



University of Fort Hare
Together in Excellence

**Design and Implementation of a Multi-Agent Opportunistic
Grid Computing Platform**

A dissertation submitted in fulfilment of the requirements for the degree

Master of Science

In

Computer Science

By

Raymond Muranganwa

Supervisor: Prof M Thinyane

To:

My family for redefining purpose:-

Declaration

I, Raymond Muranganwa declare that this dissertation and work presented herein is my own and has been generated by me as the result of my own research.

[Dissertation Title]

Design and Implementation of a Multi-Agent Opportunistic Grid Computing Platform

I confirm that:

1. This dissertation has not been submitted to any other institution for a degree qualification;
2. This work was done only while in candidature for a research degree at this University;
3. I have acknowledged main sources of help.

Signed:

Date:

Abstract

Opportunistic Grid Computing involves joining idle computing resources in enterprises into a converged high performance commodity infrastructure. The research described in this dissertation investigates the viability of public resource computing in offering a plethora of possibilities through seamless access to shared compute and storage resources. The research proposes and conceptualizes the Multi-Agent Opportunistic Grid (MAOG) solution in an Information and Communication Technologies for Development (ICT4D) initiative to address some limitations prevalent in traditional distributed system implementations.

Proof-of-concept software components based on JADE (Java Agent Development Framework) validated Multi-Agent Systems (MAS) as an important tool for provisioning of Opportunistic Grid Computing platforms. Exploration of agent technologies within the research context identified two key components which improve access to extended computer capabilities. The first component is a Mobile Agent (MA) compute component in which a group of agents interact to pool shared processor cycles. The compute component integrates dynamic resource identification and allocation strategies by incorporating the Contract Net Protocol (CNP) and rule based reasoning concepts. The second service is a MAS based storage component realized through disk mirroring and Google file-system's chunking with atomic append storage techniques.

This research provides a candidate Opportunistic Grid Computing platform design and implementation through the use of MAS. Experiments conducted validated the design and implementation of the compute and storage services. From results, support for processing user applications; resource identification and allocation; and rule based reasoning validated the MA compute component. A MAS based file-system that implements chunking optimizations was considered to be optimum based on evaluations. The findings from the undertaken experiments also validated the functional adequacy of the implementation, and show the suitability of MAS for provisioning of robust, autonomous, and intelligent platforms. The context of this research, ICT4D, provides a solution to optimizing and increasing the utilization of computing resources that are usually idle in these contexts.

Acknowledgements

I offer my deepest regards and thanks to all who contributed positively to the success of this research.

I thank God for providing in abundance.

With immense gratitude, I acknowledge the support of my supervisor, Prof. Mamello Thinyane. His interest, research ideas, constructive feedback and making time for consultations made the research enjoyable. This supervision realized that it's not always about getting there but the person one becomes in the process.

Many thanks to the Head of Department, Mr. M.S Scott who gave me an opportunity to be part of the team; and all staff members who were involved directly or indirectly; especially Prof. K. Sibanda, Mr. Z. Shibeshi and Mr. S. Ngwenya. I shall continue with Ms. N. Moorosi for bringing me close to the high performance systems domain; the skills gained are invaluable.

Great appreciation to my colleagues and friends; being there was enough; you inspired me to keep on during tough times.

I express my love and gratitude to my parents and sisters for their prayers, motivation and support.

Lastly, I acknowledge the financial support provided through the Telkom Centre of Excellence, Department of Computer Science.

Contents

1 Introduction.....	1
1.1 Background	1
1.2 Research Context.....	2
1.3 Research Rationale	2
1.4 Problem Identification.....	3
1.5 Aims and Objectives	4
1.6 Conclusion.....	5
2 Research Design.....	6
2.1 Design.....	6
2.1.1 Philosophy and Approach.....	7
2.1.2 Strategy	7
2.1.3 Methods	8
2.1.3.1 Research Methodology	8
2.1.3.2 System Development Methodology.....	8
2.1.4 Agent Oriented Software Engineering	10
2.1.4.1 GAIA.....	11
2.1.4.2 MESSAGE.....	11
2.1.4.3 Tropos	12
2.1.4.4 JADE MAS Methodology.....	12
2.2 Conclusion.....	14
3 Distributed Computing.....	15
3.1 Opportunistic Grids	15
3.1.1 Opportunistic Grids Characterization.....	15
3.1.2 Volunteer Computing	17
3.1.2.1 Folding@home	17
3.1.2.2 Storage@home.....	18
3.1.2.3 SETI@home.....	18
3.1.2.4 Distributed.net.....	18
3.1.2.5 Bayanihan	19
3.1.2.6 Javelin	19
3.1.2.7 VC Challenges	19

3.2 Grid Computing.....	20
3.3 Cloud Computing.....	20
3.3.1 The Cloud Architecture	21
3.3.1.1 Essential Services.....	21
3.3.1.2 Services and Deployment Contexts	22
3.3.2 The Cloud Realization	22
3.4 Distributed Storage.....	23
3.4.1 Google File System	24
3.4.1.1 GFS Architecture	24
3.5 Conclusion.....	25
4 Multi Agent Systems.....	26
4.1 Agent Technology	26
4.2 Multi-Agent Systems Rationale	26
4.3 FIPA Compliance.....	27
4.4 MAS Communication	27
4.4.1 KQML and KIF	28
4.4.2 FIPA ACL.....	28
4.5 MAS Coordination	29
4.5.1 MAS Negotiation Strategies	29
4.6 MAS Challenges	31
4.7 MAS Development Platforms	31
4.7.1 Aglets.....	32
4.7.2 Anchor	32
4.7.3 Zeus	33
4.7.4 JADE MAS.....	33
4.7.4.1 JADE Architecture.....	33
4.7.4.2 Agent Tasks: Behaviours	34
4.7.4.3 JADE ACL.....	35
4.7.4.4 Debugging Tools.....	36
4.7.5 Platform Evaluation.....	37
4.8 MAS Development Bottlenecks.....	39
4.9 Multi-Agent Distributed Computing.....	40

4.10 Code Mobility in Distributed and Multi-Agent Systems	41
4.10.1 Mobile Agent.....	41
4.10.1.1 Advantages of MA Paradigm.....	41
4.10.2 Remote Evaluation	42
4.11 Knowledge Presentation and Reasoning for MAS.....	42
4.11.1 Rule Reasoning.....	42
4.11.2 Rule Engines.....	43
4.11.2.1 SweetRules.....	44
4.11.2.2 F-OWL	45
4.11.2.3 Drools and Jess	45
4.12 Conclusion.....	45
5 MAOG Implementation Context and Requirements	46
5.1 The Siyakhula Living Lab (SLL) Context	46
5.2 MAOG Services and Requirements	47
5.2.1 Compute Component.....	48
5.2.2 Storage Component	48
5.2.2.1 Disk Mirroring File-System.....	48
5.2.2.2 C-AP File-System	49
5.2.3 Non-Functional MAOG Requirements	49
5.3 Conclusion.....	50
6 The MAOG System	51
6.1 Compute Component.....	51
6.1.1 Mobile Agent Platform.....	51
6.1.1.1 MA Architecture	51
6.1.1.2 System Specifications	53
6.1.1.3 Agent Interactions.....	53
6.2 Storage Component.....	56
6.2.1 System and Requirement Analysis	57
6.2.2 Agent Identification.....	58
6.2.3 Agent Tasks	58
6.2.4 Storage Modules	60
6.2.5 DM File-System	60

6.2.5.1 Upload Service: 1 st Iteration	60
6.2.5.2 Upload Service: 2 nd Iteration	63
6.2.5.3 Upload Service: 3 rd Iteration	65
6.2.5.4 Download Service	67
6.2.6 C-AP File-System	68
6.2.6.1 Dynamic vs. Static Chunking	69
6.2.6.2 C-AP Upload Service	69
6.2.6.3 C-AP Download Service	71
6.3 Conclusion	73
7 MAOP Services	74
7.1 MA Component	74
7.1.1 Processing User Agent-PUA	74
7.1.2 Node Processing Agent-NPA	75
7.1.3 Processing Resolver Agent-PRA	76
7.1.4 Mobile Agent-MA	76
7.1.4.1 Processing Mobile Agent	77
7.1.5 Integration Exceptions	79
7.1.5.1 JADE and Drools: Exceptions	79
7.1.5.2 JADE and Jess: Exceptions	82
7.2 Storage Component	85
7.2.1 DM File-System: Upload Service	85
7.2.1.1 SCUA	85
7.2.1.2 RA	86
7.2.1.3 SA	88
7.2.2 DM File-System: Download Service	89
7.2.2.1 SCUA	89
7.2.2.2 RA	90
7.2.2.3 SA	90
7.2.3 C-AP File-System: Upload Service	91
7.2.3.1 RA	91
7.2.3.2 SA	93
7.2.4 C-AP File-System: Download Service	94

7.2.4.1 RA	94
7.2.4.2 SA	94
7.3 Conclusion.....	95
8 Results and Analysis	96
8.1 Compute Component.....	96
8.1.1 Evaluation	96
8.2 MAOG File-Systems.....	98
8.2.1 Measurement Criteria	99
8.2.2 Experimentation.....	100
8.2.3 DM Service Evaluation	101
8.2.3.1 DM U_T1 Analysis.....	103
8.2.3.2 DM UTAT Analysis	106
8.2.3.3 DM U_RTT Analysis.....	108
8.2.3.4 DM DRTT Analysis.....	110
8.2.4 C-AP File-System Evaluation	111
8.2.4.1 C-AP U_TAT Analysis.....	111
8.2.4.2 C-AP URTT Analysis	113
8.2.4.3 C-AP DRTT Analysis	115
8.2.5 C-AP vs. DM File-System.....	117
8.3 Security.....	119
8.4 Conclusion.....	120
9 Conclusion and Future Work	121
9.1 Summary	121
9.2 Findings.....	121
9.3 Contributions	123
9.4 Limitations	125
9.5 Future Work	125
9.6 Publications	126
Bibliography	127
Appendix A - Compute Implementation Details	139
A. MA Platform.....	139
A.1 Processing User Agent-PUA	139

A.2 Node Processing Agent-NPA	141
A.3 Processing Resolver Agent-PRA	142
A.4 MobileAgent.java	144
Appendix B - Storage Implementation Details	148
B. Storage Component	148
B.1 DM File-System: Upload Service	148
B.2 DM File-System: Download Service	154
B.3 C-AP File-System: Upload Service	158
B.4 C-AP File-System: Download Service	161

List of Figures

Figure 1: The Research Onion [15]	6
Figure 2: Deductive Approach.....	7
Figure 3: Iterative Development	9
Figure 4: JADE MAS Methodology [22]	13
Figure 5: Cloud Definition [49]	21
Figure 6: GFS Architecture [54]	25
Figure 7: JADE Architecture	34
Figure 8: Message Object	35
Figure 9: Message Receive	36
Figure 10: RMA.....	36
Figure 11: Rule Engine Components.....	43
Figure 12: SLL Connectivity	47
Figure 13: MA Architecture.....	52
Figure 14: MA Use Case.....	53
Figure 15: MA Infrastructure.....	54
Figure 16: Node utilisation ERD	55
Figure 17: MA Sequence Diagram	56
Figure 18: Storage Use Case.....	57
Figure 19: Agent Diagram	58
Figure 20: Storage Component Architecture	59
Figure 21: File Upload Service (1 st iteration)	62
Figure 22: Sniffer Agent interaction capture	63
Figure 23: File Upload Service (2 nd Iteration)	65
Figure 24: File Upload service (3 rd Iteration)	66
Figure 25: File Download Service	68
Figure 26: File Download Service	70
Figure 27: File Download Service	72
Figure 28: MA Compute Project.....	74
Figure 29: PMA GUI	75
Figure 30: Keeping track of source node.....	77
Figure 31: Initialising Drools in JADE.....	80
Figure 32: Drools Engine Instance	80
Figure 33: Migration Exception.....	81
Figure 34: Migration Exception.....	81
Figure 35: Drools 5.6.0 in JADE	82
Figure 36: Starting a JADE Runtime	83
Figure 37: Jess Instance	83
Figure 38: FileNotFoundException	83
Figure 39: Rule File to Bytes	84
Figure 40: IOException.....	84

Figure 41: Serialisation Exception.....	84
Figure 42: Output Stream.....	85
Figure 43: DM project	85
Figure 44: File Chooser	86
Figure 45: Upload Status	86
Figure 46: Handling SCUA Requests	87
Figure 47: Upload Service Implements	88
Figure 48: Storing Downloaded File.....	89
Figure 49: File Retrieval	91
Figure 50: Chunking a file	92
Figure 51: Atomic Append Approach.....	93
Figure 52: Chunk Retrieval.....	95
Figure 53: Configuring Java	97
Figure 54: CNP Mechanism.....	97
Figure 55: Loading a PMA	98
Figure 56: Elementary Intervals.....	99
Figure 57: Upload Interactions	102
Figure 58: Download Interactions	102
Figure 59: DM_U_T1	103
Figure 60: RA Error Log.....	103
Figure 61: Regression Analysis	104
Figure 62: DM_U_T1 vs. File size Plot.....	104
Figure 63: Covariance	104
Figure 64: Pearson Coefficient	105
Figure 65: Significance Test	105
Figure 66: DM_U_TAT.....	106
Figure 67: Covariance	106
Figure 68: DM_U_TAT vs. File-size Plot	107
Figure 69: Test for Significance	108
Figure 70: DM U_RTT vs. File size Plot.....	108
Figure 71: Covariance test	109
Figure 72: Test for Significance	109
Figure 73: DM_D_RTT vs. File-size Plot	110
Figure 74: Covariance.....	110
Figure 75: Significant Test.....	111
Figure 76: C-AP U_TAT vs. File-size Plot	112
Figure 77: Covariance.....	112
Figure 78: Significance Test	113
Figure 79: Covariance.....	113
Figure 80: C-AP_U_RTT vs. File size Plot.....	114
Figure 81: Significance Test	115
Figure 82: C-AP DRTT vs. File Size Plot	115
Figure 83: Covariance.....	115

Figure 84: Significance Test	116
Figure 85: DM & C-AP Mean URTT	117
Figure 86: DM&C-AP Mean U_TATs	118
Figure 87: DM&C-AP Mean DRTT	118

List of Tables

Table 1: Grid Survey.....	16
Table 2: Grids and OGs Analysis [45].....	20
Table 3: Grid and Cloud Compared.....	23
Table 4: Message Parameters [93].....	35
Table 5: Agent Platforms [80]	38
Table 6: DM Upload Service Responsibilities.....	61
Table 7: Updated Agent Roles.....	63
Table 8: File Download responsibilities	67
Table 9: C-AP Responsibilities.....	69
Table 10: C-AP responsibilities.....	71
Table 11: PMA TAT.....	98
Table 12: DM_U_TAT Correlations	107
Table 13: Pearson Coefficient.....	109
Table 14: Pearson Coefficient.....	111
Table 15: Pearson Coefficient.....	112
Table 16: Pearson Coefficient.....	114
Table 17: Pearson Coefficient.....	116

List of Abbreviations

ACC	Agent Communication Channel
ACL	Agent Communication Language
AID	Agent Identifier
AMS	Agent Management System
AOT	Agent-Oriented Techniques
API	Application Program Interface
C-AP	Chunking with Atomic Append
CAP	Community Access Point
CNP	Contract Net Protocol
CPU	Central Processing Unit
DA	Database Agent
DAI	Distributed Artificial Intelligence
DAN	Digital Access Node
DF	Directory Facilitator
DM	Disk Mirroring
DRTT	Download Round Trip Time
DSS	Distributed Storage Systems
DTAT	Download Turnaround Time
FIPA	Foundation for Intelligent Physical Agents
GFS	Google File System
HTTP	Hypertext Transfer Protocol
IaaS	Infrastructure as a Service
ICT	Information and Communications Technology
ICT4D	Information and Communication Technologies for Development
IIOP	Inter-Object Resource Broker Protocol
IMTP	Internal Message Transport Protocol
JADE	Java Agent Development Platform Multi-Agent System)
KIF	Knowledge Interchange Format
KQML	Knowledge Query Manipulation Language
KR	Knowledge Representation
KRR	Knowledge Representation and Reasoning
LAN	Local Area Network
MA	Mobile Agent
MAC	Media Access Control
MAs	Mobile Agents
MAOG	Multi-Agent Opportunistic Grid
MAS	Multi-Agent Systems
MTP	Message Transmission Protocol
MTS	Message Transport Service
NCL	Node Cumulative Load
NIST	National Institute of Standards and Technology
NPA	Node Processing Agent
OG	Opportunistic Grid
OOT	Object-Oriented Techniques

OSL	Ordinal Sharing Learning
OWL	Web Ontology Language
PaaS	Platform as a Service
PMA	Processing Mobile Agent
PMAGUI	Processing Mobile Agent Graphical User Interface
PRA	Processing Resolver Agent
PRS	Production Rule Systems
PUA	Processing User Agent
RA	Resolver Agent
RDF	Resource Description Framework
RI	Research Infrastructures
RMA	Remote Monitoring Agent
RMI	Remote Method Invocation
RTT	Round Trip Time
RuleML	Rule Modeling Language
SA	Storage Agent
SaaS	Software as a Service
SCUA	Storage Component User Agent
SLA	Service Level Agreement
SLL	Siyakhula Living Lab
SS	Storage Server
SSL	Secure Socket Layer
SWRL	Semantic Web Rule Language
TAT	Turnaround Time
UML	Unified Modelling Language
URL	Universal Resource Locator
URTT	Upload Round Trip Time
UTAT	Upload Turnaround Time
VC	Volunteer Computing
VSAT	Very Small Aperture Terminal
WiMAX	Worldwide Interoperability for Microwave Access

Chapter 1

1 Introduction

This chapter presents the research overview, context, objectives and rationale. The problem identification section present challenges faced with emerging enterprises in hosting specific systems and services. The discussions unfold open research questions which recommend ICT4D infrastructures as alternative computing resources realized through MAS. Research aims and objectives are laid out to complement the related questions.

1.1 Background

The rapid increase in multi-purpose access centers in recent years has heightened the use of Information and Communication Technologies (ICTs) for socio-economic development [1]. ICTs play a key role as transformational tools for growth and spur of knowledge in developing countries. A wealth of ICT4D initiatives have established formal and informal sector links defying the information and communications sharing barriers [2]. As much as there is significant optimism in ICTs as enablers for socio-economic sustenance in developing countries, research has shown a considerably slow ICT4D uptake and drive [3], [4]. Lack of project implementation blueprints for Information and Communications Technology (ICT) initiatives is a major contributing factor [5]. However, researchers remain resolute that ICTs can play a crucial role in bridging the technology gap [6], [7].

The development of roadmaps for utilizing Research Infrastructures (RIs) to include ICTs for scientific research has provided and opened up opportunities for increased innovation. RIs are physically distributed resources and services used by scientific communities for top-level research. The infrastructures empower researchers by offering access to facilities irrespective of ownership and location. The view of expanding ICT-based electronic infrastructures such as Opportunistic Grids (OGs) and data centres support development thus strengthening the industrial base of knowledge and technological expertise.

As computers are encompassing all facets of our lives with technological advancements, demand is being felt for low-cost computational and storage resources. Commercial services such as Amazon Elastic Compute Cloud (Amazon EC2) [8] and Google drive offer these on-demand

resources. Few services are built around ICTs compared to commercial applications; this is even true in developed countries where technology is readily available [2]. Cooperative structuring of ICTs foster the advent of approaches built on shared use of resources in diverse socio-economic landscapes and in low-resource contexts. This creates sustainable partnerships for provisioning of crucial services such as of e-Learning, e-Health and e-Government.

Exploration of Opportunistic Grid (OG) e-infrastructures exposed vast possibilities in seamless integration of machine resources. OGs integrate computing resources from geographically dispersed networked locations [9]. The vision for large-scale, transparent and pervasive sharing of resources in diverse administrative domains can be a reality as a result of OGs. This research focused on the design and implementation of a grid solution that harness resources from idle non-dedicated computers in an ICT4D context. The platform as envisaged brings effective computing inclusive of processing power and/or storage capabilities to ordinary users and institutions as a substitute for expensive high performance systems.

1.2 Research Context

This study involved an investigation on the effectiveness of integrating low-cost commodity resources into a converged ICT solution within an ICT4D research project (called Siyakhula Living Lab) undertaken at a rural community of Dwesa in South Africa. Dwesa is a marginalized rural community in the former Transkei region of the Eastern Cape about 400 kilometers from East London [10]. The Siyakhula Living Lab (SLL) is an ICT4D initiative that thus far principally uses schools in the community as test beds for deploying ICT infrastructures in the area [11]. These ICT service centers (formally called Digital Access Nodes) were established on ICT collaboration models. The models assume community participation in all project stages thereby facilitating development in areas formerly lacking ICT services and infrastructures [12]. The SLL is seeking to capitalize on the potential of ICTs towards socio-economic development in marginalized rural communities.

1.3 Research Rationale

The current network deployed in Dwesa consists of a Worldwide Interoperability for Microwave Access (WiMAX) backbone network which connects about 16 digital access nodes (largely schools). This provides a high-speed broadband island within the network with a back-haul link

to the Internet via Very Small Aperture Terminal (VSAT). Each of the Digital Access Nodes (DANs) consists of a number of computers used by the community to access network services. These computers are generally under-utilized mainly because they are only accessible during school hours. The computers are usually idle because the students rarely use them and they are not always accessible to the community even when the students are not using them. They are typically used for very basic applications/services which don't fully utilize the available capacity (e.g. processor and storage). These idle computing resources provide a perfect opportunity to be utilized in an OG setup. This research considers the MAOG solution on this type of network. The research seeks to determine its feasibility and its technical adequacy. A successful deployment of a grid in a rural marginalized community, such as Dwesa, would open up a plethora of possibilities and opportunities. The grid solution in the research context enables computing to previously marginalised backgrounds to participate in Internet-based volunteer networks. By utilizing such an ICT facility, community members assume the role of producers of services rather than being consumers primarily.

1.4 Problem Identification

An assortment of high performance systems (i.e. supercomputers) have been developed to satisfy storage and floating point calculation demand of specific applications and projects (e.g. in earthquake prediction and visualization simulations) [13]. While these distribute enhanced computing resources, they nonetheless quickly become obsolete and cost organizations money in anticipated downtimes [14]. The prospect of hosting such systems is unachievable for selected users and enterprises due to high capital injection for short-lived life cycles as a main factor. The costs scale from hardware, physical space, temperature-controlled facilities and maintenance which prove this option not viable for large-scale adoption in non-enterprise contexts. On the course to understand and investigate how standard ICT resources can be an alternative computing resource in an OG setup context, the following research questions were explored in this research:

1. Can low-cost commodity computers in ICT contexts be exploited for an OG setup?
2. What is the most appropriate distributed computing design and implementation for an ICT4D OG?

OGs are typically based on utilization of heterogeneous resources which present challenges in integrating the shared compute and storage resources. Primarily, it is mandatory to use these resources only when they are idle. Also, the resources have erratic uptimes which disrupt grid processes. A candidate technology for the implementation of the OG platform in an ICT4D context is the MAS technology. MAOG proposes a scalable, adaptive and context aware approach to dynamic resource identification and allocation in distributed heterogeneous uncertain settings. As such, in addressing question two above, this research undertakes an evaluation of the agent technology as a potential tool to ease the development of distributed systems to include OGs in factual scenarios identified. Linked to the second research question, the following sub-questions have been identified:

3. Is a MAS solution a feasible technology for implementation of the platform and does it provide the necessary functional adequacy?
4. What is the most suitable MAS analysis and design methodology that can be utilised in the implementation of such a MAS system?
5. How can agents reason about their execution environment to adapt the system to dynamic environment changes? And what is the most suitable rule engine to implement rule reasoning in MAS?

The design and implementation of software systems using agent-oriented approaches is unquestionably challenging. A selection of domain independent concerns have to be identified which range from how autonomous agents in dispersed environments interact and agents having shared or conflicting goals. This document examined how ICT infrastructures can be utilised for commodity grids to address key issues in particular enterprises operating on constrained budgets. The adoption and validation of the MAS technique in addressing grid implementation challenges were analysed.

1.5 Aims and Objectives

The main aim of this research was to determine the suitability and feasibility of MAS in the provisioning of a distributed infrastructure which exploits idle shared computing resources. The related sub-objectives are listed further on:

1. Determine the effectiveness of deploying an OG platform in the low-resourced contexts;

2. Undertake an extensive evaluation of the selected grid solution.

While this research is conceptualized in low-resources context, it is equally applicable to any setting where idle computing resources can be combined into an OG infrastructure.

1.6 Conclusion

The main research concepts have been discussed in this chapter. To realize a MAOG system, the underutilised machines in Dwesa were identified as a potential resource base for an OG setup. Although the agent technology has exciting features such as scalability and context awareness in distributed application development; addressing domain independent concerns was highlighted as the main challenge.

Chapter 2

2 Research Design

This chapter details the study's design based on the research onion metaphor. In this chapter research questions and objectives are also turned into a research project. Discussions on the research philosophy, approach and method will outline the design assumptions. Special attention will be given to how MAS are unified with opportunistic computing to devise strategies for utilizing shared machine capabilities in an ICT4D context. Literature in this chapter also address research question 4 and motivates a MAS methodology that can be utilised for the MAOG system development.

2.1 Design

Research activities contribute to discovery and confirmation of scientific knowledge. Techniques for acquiring and analysing data represent only the final phase in a research design. Identifying a research approach that is comprehensive and elaborate was crucial for this research. Hence the Saunders et al research onion metaphor was adopted to guide the research design and implementation process [15]. The research onion (Figure 1) is composed of six layers which convert research questions and objectives into a research project.

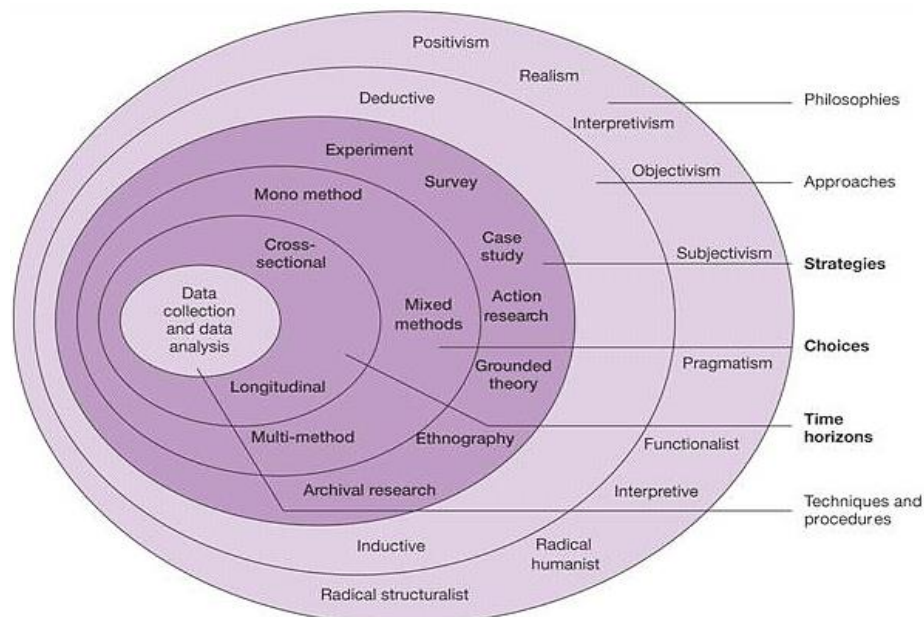


Figure 1: The Research Onion [15]

The questions and objectives inform on choice of research philosophy, approach, strategy and time horizons on which projects can be undertaken. Considering the research onion concepts develops a coherent research design that is justifiable [15]. The onion layers examined by this research are presented in proceeding subsections.

2.1.1 Philosophy and Approach

The research philosophy influences the course of the entire research. It incorporates assumptions on suitable research methods which interpret the project views [16]. Positivism [17], [18] was observed as a philosophical perspective consistent with aims, objectives and nature of this research.

Positivism assumes that the observer and instruments are independent of the objectively given reality. Positivism analyses phenomenon by making assumptions based on quantifiable measures and hypothesis testing [16]. That is, positivists utilize deductive methods to test theories thus gaining predictive understanding of these phenomenon [16]. The deductive method life cycle in Figure 2 include: (1) development of a theory from previous findings (i.e. literature); (2) derivation of hypothesis from theory; (3) making observations; and (4) confirmation of a claim or rejecting hypothesis.

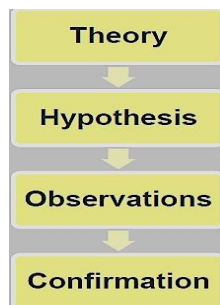


Figure 2: Deductive Approach

2.1.2 Strategy

After identifying the research philosophy and approach; the experiment strategy was chosen to guide this project based on positivistic and deductive interpretations. Experiments create and manipulate relationships between variables to check how system scenarios affect conclusions or results [16]. Experiments were relevant in answering research questions of the form “how” or “why” and investigating the performance of selected system components.

2.1.3 Methods

A mixed methods approach was considered in this layer. The technique was used to divide this study into two; research and system development methodologies to address specific research questions outlined in chapter 1.

2.1.3.1 Research Methodology

The research examined opportunistic computing as a justifiable model to utilize ICT infrastructures using Dwesa as a case study. Distributed computing surveys were conducted. The questions (introduced in Section 1.4) answered in the related work section include:

1. Can low-cost commodity computers in ICT contexts be exploited for an OG setup?
2. What is the most appropriate distributed computing design and implementation for an ICT4D OG?

A survey on these questions and related technologies achieved the following milestones:

- i. Motivated the choice for Opportunistic Grid Computing;
- ii. Clarified the research's Opportunistic Grid Computing perspective;
- iii. Contrasted opportunistic computing with other mainstream distributed computing models;
- iv. Considered the benefits of public resource computing in enterprises;
- v. Proposed service types that can be realized on low-cost commodity resources.

Multiple visits to Dwesa were conducted in order to evaluate the ICT infrastructures and their usage patterns. ICT4D field concerns such as unreliable power and erratic telecommunications infrastructure that may affect an OG solution were confirmed during those visits.

2.1.3.2 System Development Methodology

The system development methodology was used to guide the design and implementation of the MAOG system's compute and storage components. The approach recommended the significance of the agent technology in implementing specific distributed services. The research questions reviewed to explore MAS in opportunistic computing include:

1. Is a MAS solution a feasible technology for implementation of the platform and does it provide the necessary functional adequacy?

2. What is the most suitable MAS analysis and design methodology that can be utilised in the implementation of such a MAS system?
3. How can agents reason about their execution environment to adapt the system to dynamic environment changes? And what is the most suitable rule engine to implement rule reasoning in MAS?

An agent oriented software engineering methodology by Nikraz et al (introduced in Section 2.1.4.4) was adopted for the analysis and design of MAOG services. The methodology incorporates iterative development enabling developers to reconsider each stage flexibly.

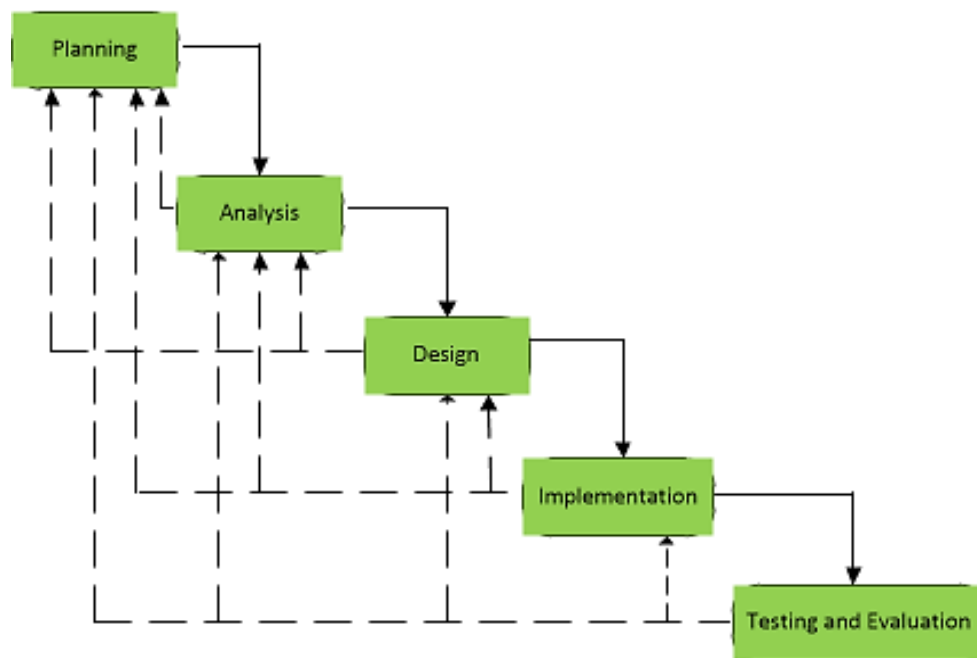


Figure 3: Iterative Development

The system development methodology integrated the planning and implementation stages into the agent software engineering methodology:

1. **Planning:** The step evaluated programming paradigms, rule engines and the agent-based option as logical for this research. Literature review was conducted on:
 - i. Agent Technology;
 - ii. MAS Rationale;
 - iii. MAS Communication;
 - iv. MAS Coordination;

- v. MAS Development Platforms;
- vi. Agent Oriented Software Development;
- vii. Knowledge Presentation and Reasoning in MAS.

From literature suitable rule engines, a design methodology and a MAS platform (JADE MAS) were identified for the MAOG system implementation. The agent technology was also established as viable in distributed applications based on related work.

2. **Analysis:** In the analysis stage research problems were introduced without solution considerations. The following stages were included:
 - i. Agent type identification;
 - ii. Agent responsibility formulation;
 - iii. Acquaintance identification.

Use case diagrams and agent relationships deliverables shaped the foundation of the analysis stage.

3. **Design:** The solutions to problems identified in the analysis step were considered in this step. Agent interaction specification was the main activity. For agent types and their responsibilities identified; related agent acquaintances relationships were defined. The agent sequence diagrams designed enabled for easy transition into the implementation stage.
4. **Implementation:** The MAOG compute and storage services were implemented using the JADE platform. The Drools (Section 4.11.2.4) and Jess (Section 4.11.2.1) engines were experimented with a specific compute component. Due to Drools integration issues (documented in Section 7.1.5.1), Jess features were integrated for rule based reasoning.
5. **Testing and Evaluation:** The MAOG services were setup in an experimental environment and evaluated.

2.1.4 Agent Oriented Software Engineering

Agent-Oriented Techniques (AOT) combine uniform approaches in the development of flexible systems with complex behaviour [19]. That is, AOT support system analysis, design and implementation stages with a distinct uniform theory namely that of agents. Agent participant identification and behaviour refinement are considered in the design phases.

AOT foundation is deeply rooted in Object-Oriented Techniques (OOT). Specifically, AOT from an engineering perspective is a specialization of OOT. However, according to Shoham [20], OOT and AOT differ in the following ways: (i) the internal states of agents in AOT are structured by mental notions (i.e. beliefs, goals and intentions) and entities communicate using a universally defined communicative language; and (ii) OOT describe a system as composed of modules which communicate and have separate methods of handling messages. Due to these underlying variations, agent systems are analysed, designed and implemented differently.

Utilising a suitable development methodology in projects is crucial as it reduces system development time. Most existing MAS design approaches are centred on top-down and object-oriented methodologies. Using the two simultaneously has limitations, as agents and objects are generalised differently; thus should be considered at separate levels [21], [22]. Since MAS provide a means of problem solving in certain domains where some techniques lack; a comprehensive software development methodology was imperative for this research. GAIA, MESSAGE, TROPOS and the JADE MAS methodology are examples of guidelines used in agent-oriented development.

2.1.4.1 GAIA

GAIA [23] is a theoretical framework to guide the development of MAS from analysis to design. The approach is focused on the macro-level (societal) and micro-level (agent) aspects of systems. GAIA has two analysis and three design models which omit specification collection (i.e. requirement gathering) and implementation. Although the models are established; their illustration is for a subset of the concepts necessary for agent oriented analysis [23]. The methodology was extensively adopted but encountered its fair share of limitations in its practicality to real world multi-agent contexts. Critically the approach was suitable for analysis and design of closed MAS in which agents cooperate to achieve objectives in closed systems. This factor sets it apart from current agent-based system setups owned by different stakeholders and which interact for self-benefit or collaboration [23].

2.1.4.2 MESSAGE

MESSAGE (Methodology for Engineering Systems for Software Agents) [19] is a software engineering approach which cover analysis and design of agent systems. The knowledge level

entities in MESSAGE are classified into Concrete Entity (consisting of agents, organisation, roles and resource entities), Activity (task and interaction) and Mental State Entity (goal) classes. The method “extends Unified Modelling Language (UML) by contributing with agent knowledge level concepts and diagrams with notations to view them” [19]. From a structural perspective “MESSAGE entities are objects with operations and attributes expressed by methods”; the behavioural view illustrates them as state machines. Although MESSAGE extends UML to manage agent interaction; the approach doesn’t have agent technology concepts at its centre.

2.1.4.3 Tropos

Tropos [24] methodology applies agent concepts in all phases of development. Tropos is founded on the following ideas [25]:

1. Agents and related mentalist concepts are used in all software development phases.
2. Requirement analysis facilitating for an understanding in the system environment set-up.

Principally Tropos consists of [25]:

- Early requirement analysis which analyses a problem by studying its organisational setting;
- Late requirement analysis: the system is described in its environment inclusive of its functionality;
- Architecture design: the different subsystems of the global architecture are defined.

Emphasis on requirements analysis distinguish Tropos from current MAS methodologies [25]. The methodology hasn’t been used for developing a fully-fledged MAS and lacks tools which support the transition between phases [25].

2.1.4.4 JADE MAS Methodology

Nikraz et al proposed a methodology (shown in Figure 4) for analysis and design of MAS using JADE [22]. Generic software engineering aspects are covered in the analysis stage with the latter focused on the JADE platform. Contrary to current methodologies which extend object-oriented methods, the JADE methodology is centred on agents specifically and the agent paradigm abstractions.

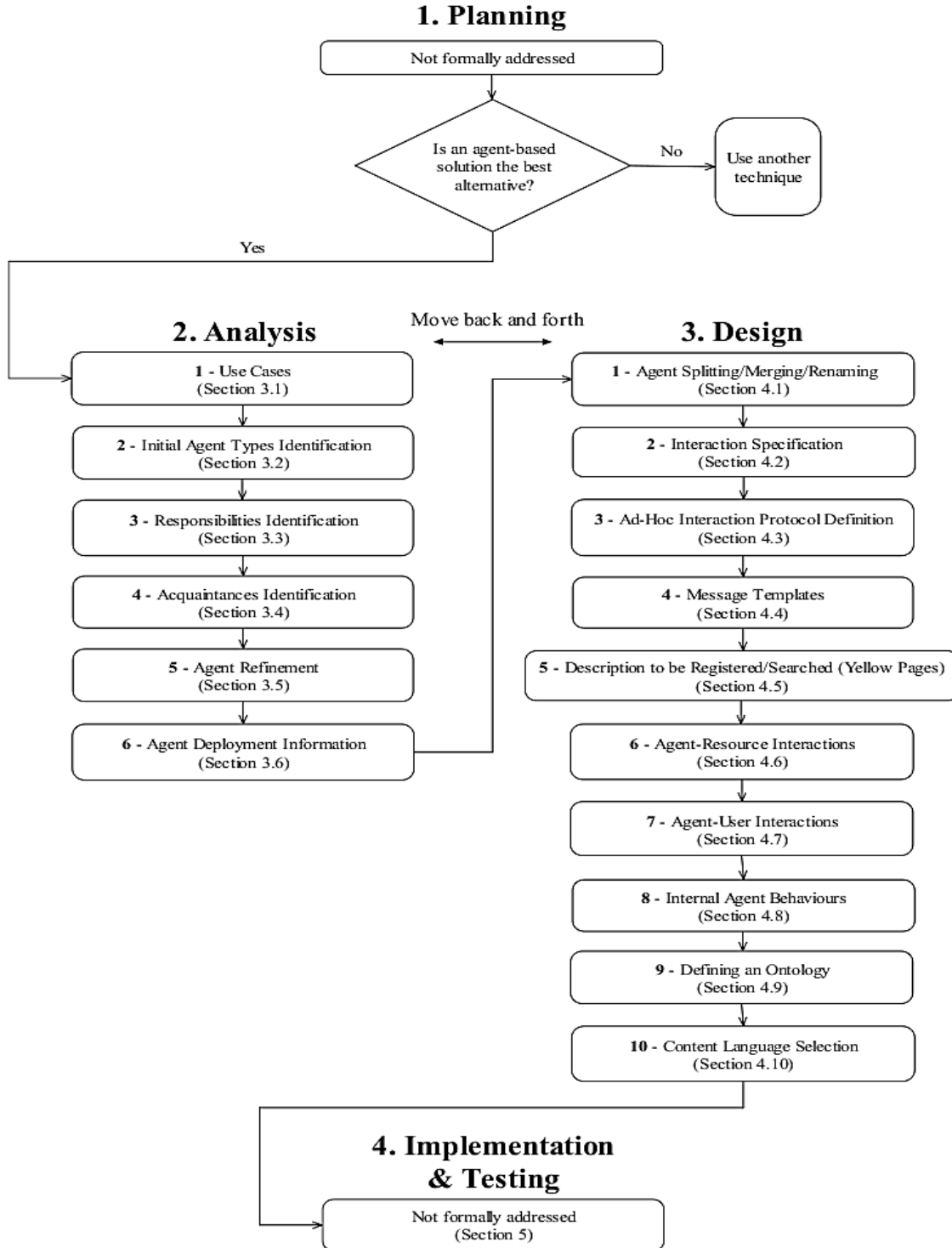


Figure 4: JADE MAS Methodology [22]

Additionally, the top-down and bottom-up approaches included account for system capabilities (e.g. legacy systems and people) and application requirements. This enable designers who are new to JADE and the agent computing field to grasp quickly important concepts in MAS

development [22]. The Nikraz et al methodology was selected to guide the MAOG compute and storage services development. The comprehensiveness of the methodology in the analysis and design stages of MAS based on JADE motivated its adoption for this research. Also, its support for top-down and bottom approaches enable people and target infrastructures in the research context to be incorporated flexibly to realise a MAS infrastructure with real life application.

2.2 Conclusion

The research onion discussed in the previous sections provided an overview of all research stages. This chapter presented the study in terms of the research and system development methodologies. A MAS development methodology explained in terms of JADE was selected to drive the development of the compute and storage services. The following chapters introduce the distributed computing domain, MAS concepts and the ICT4D context in which the research is undertaken.

Chapter 3

3 Distributed Computing

The literature foundations for this research are laid in this chapter and the relevant mainstream paradigms. Background on the research topic is provided to show contributions the study offers based on research conducted in the related area. The literature in this chapter address questions 1 and 2 introduced in Section 1.4. The main objective is to determine the effectiveness of deploying an OG platform in low resourced contexts. Concepts in OGs and how they differ from other computing paradigms are discussed. In this regard, this section elaborates the OG perspective to confirm the research's computing standpoint.

3.1 Opportunistic Grids

OGs also known as Desktop grids are distributed infrastructures which join idle resources from users on the Internet to work on computational and storage resource constrained projects [26], [27]. Computing in this model depends on ordinary resources to attain its goals, contrary to other distributed systems wherein resources are dedicated for specific computational tasks. The volunteered machines share a quota of their resources to form a geographically distributed computing solution. Trends in production of multi-core processors in computers with enhanced storage capabilities for affordable market value are evolving on a globally scale. A number of these machines are connected by high speed connectivity which makes each computer a potential volunteer node.

3.1.1 Opportunistic Grids Characterization

OGs are classified as either volunteer or enterprise systems based on platform support, scalability and nature of resource providers [28]:

1. **Platform:** OGs are web and/or middleware based with respect to the platform running on the resource provider. In web-based, volunteers download applications (e.g. java applets) using web browsers and process applets using shared machines. The middleware approach however requires resource providers to set-up applications which offer the required service.

2. **Scalability:** Scalability classify OGs in two: Local Area Network (LAN) and Internet based [29]. Internet-based are “characterized by anonymous resource providers, connectivity issues, malicious resources and high security risks” [29]. LAN-based on the other hand are governed by policy frameworks and have consistent connectivity. Usually Volunteer Computing (VC) is internet-based and enterprise grids are LAN-based.
3. **Resource Provider:** Resource types identify how resources are shared in distributed systems. Volunteer and enterprise are the two primary support scenarios. VC infrastructures rely on public users’ participation, whereas in enterprise computing; users share resources involuntarily and are usually within a university or corporation.

Table 1 shows an overview of available OG systems in related classifications as discussed.

System	Platform (Based)	Scalability	Resource Provider
Bayanihan [30]	Web or Middleware	Internet	Volunteer
Boinc [31]	Middleware	Internet	Enterprise
Condor	Middleware	LAN	Enterprise
SETI@home [32]	Middleware	Internet	Volunteer
Distributed.net [33]	Middleware	Internet	Volunteer
Entropia [34]	Middleware	LAN or Internet	Enterprise or Volunteer
QADPZ [35]	Middleware	LAN, Internet	Enterprise
SZTAKI [36]	Middleware	LAN, Internet	Enterprise
Javelin [37]	Web	Internet	Volunteer
Folding@home [38]	Web	Internet	Volunteer

Table 1: Grid Survey

The research focused on a platform founded on computing resources shared from different heterogeneous environments. The resource provider characteristic was important in motivating our design and approach in this research. From connected factors, VC was considered as appropriate for this study. Although enterprise grids overcome volatility (i.e. robustness, security and reliability) concerns; they are limited in computational power compared to the virtually unlimited resources in VC [29].

3.1.2 Volunteer Computing

VC is established on computational and participative pillars. The computational aspect deals with allocation and management of machine resources (e.g. storage and processing), with the latter focused on motivations to contribute computer resources in projects [39].

Human factors in VC are important for distributed systems to achieve specific design goals. Evidence from studies conducted on why people volunteer resources suggested the following [40]:

1. **Support for scientific goals:** Computer resources are shared to support research goals (e.g. such as in curing diseases and extra-terrestrial life search);
2. **Credit:** Some resource providers are into computer benchmarking and use VC as a platform to publicize their machine performance.

To date a number of VC applications have produced resources comparable to a selection of supercomputers and commercial file hosting services [41]. Related projects with millions of users offering unmatched computing power and storage are currently being used in medicine, bioinformatics, climate studies, astrophysics and molecular biology [2]. SETI@home [42] for instance managed to gather 2.5 million years of processor time in a 7 year operation period [41]. Examples of recognized VC projects include: Folding@home [43], Storage@home [44], Distributed.net [33], Bayanihan [30] and Javelin [37].

3.1.2.1 Folding@home

Folding@home is an active project that harvests processing cycles for folding simulations to understand protein mis-folding oriented diseases and therapies [38]. The solution searches for an alternative source of computational power to cater for “the combination of detail needed in the simulations coupled with long timescales required to compare experiments” [38].

To share processor cycles, resource providers install client software and define when folding can occur. A work unit is processed as follows [38]: (1) An installed client requests an assignment server to assign it to a work-server; (2) client downloads a work-unit and computational core required for processing from the web-server; and (3) client returns feedback on completion. Although Folding@home is formalised in protein folding, the system has potential value in different domains.

3.1.2.2 Storage@home

Storage@home is a “distributed storage infrastructure intended to solve the problem of backing up and sharing petabytes of scientific results using a model of volunteer managed nodes” [44]. Traditional approaches to backing up data were not scaling with Folding@home generated data which increased at a scale of 2 terabytes per month [44]. Contributions of ten gigabytes per computer multiplied by thousands of computers were projected. The system anticipated volunteers who participate in the community for at least 6 months earning points in the process. A penalty was proposed for participants who exit without notice.

3.1.2.3 SETI@home

SETI (Search for Extra-Terrestrial Intelligence) is a scientific study which aims to determine the existence of life outside earth [32]. An approach, radio SETI, which use radio telescopes to listen on specific radio signals known not to occur naturally is used to provide evidence on extra-terrestrial activity [42].

SETI@home enable anyone with an Internet connection and a computer to participate in radio SETI data analysis when their computers are idle. The process is made possible through a client program with a screen saver behaviour running on the volunteer’s machine. The screen saver fetches data from a centralised server, analyses and reports on results. When a node is recalled, the screen saver suspends and resumes analysis only when the machine reassumes the idle state. The project hasn’t identified evidence of extra-terrestrial life yet, but has certainly established the viability of public resource computing [42].

3.1.2.4 Distributed.net

Distributed.net pursues processing challenges by exploiting combined idle processing cycles from member machines [33]. RSA security firm’s utilisation of the infrastructure to evaluate vulnerabilities in encryption schemes attracted more participants to join [45]. Initial effort to break the “RC5-56 portion of the RSA Secret-Key Challenge, a 56-bit algorithm” by the company for a prize was suspended due to “SYN flood attacks by participants on the server” [46]. A new independent effort, Distributed.net focused on harnessing the power of home computers towards academic and public interest projects was then developed. The deployment

moved for global distributed computing through participation, contribution of expertise, processing power and bandwidth.

3.1.2.5 Bayanihan

Project Bayanihan shaped by the Filipino tradition of communal unity and cooperation “makes it easy for ordinary people with little technology to cooperate in solving parallel problems” by sharing their processing power [30]. Minimising effort and expertise in sharing nodes motivated for a world-wide computing network.

In addition to Distributed.net achievements, Bayanihan introduced a web-based VC component. The technology enabled developers to code “platform independent parallel applications in Java and post them as web applets” where volunteers only require a web browser and a few operations to join a computation [30]. When economic models are integrated, a survey in [30] suggested a shift from the Bayanihan barter system to a commercial approach in which computers become a commodity people can buy or trade in.

3.1.2.6 Javelin

Javelin is a Java-based project composed of brokers, clients and hosts [37]. A broker entity “coordinates the demand and supply of computing resources”; with clients representing processes requesting machine resources from hosts. The role of a client or host is dynamic in certain situations. “A machine may serve as a Javelin host when it’s idle, while being a client when its owner needs additional computing resources” [37]. By selecting a known broker Universal Resource Locator (URL), “volunteers automatically share their machine capabilities towards parallel computations” [37].

3.1.2.7 VC Challenges

This section highlights some issues faced in realizing VC applications. Some concerns in existing projects are [28]:

1. **Volatility:** Shared nodes are not dedicated. They constantly join and exit projects during work-flow operations. Hence uptime intervals are periodic and unpredictable.
2. **Security:** Volunteer systems should be secure in-order to attract people to share their machines.

3. **Failure:** Projects are prone to faults due to their size and open nature. These scale from nodes failure, data corruption and faulty network links.
4. **Scalability:** For public resource computing to be effective, speedups similar to available computing technologies should be offered.

3.2 Grid Computing

Grid computing advanced through developments in parallel and high throughput computing [28]. The paradigm coordinate for problem solving using multiple-institutional resources to deliver transparent and secure access to machine resources [47]. From definition, grids and OGs have similarities. An OG can be viewed as a type of grid, but the models differ based on resource types, connectivity, dedication, trust and failure. The detailed analysis of grid computing and OG classes is shown in Table 2.

Item	Desktop Grid (DG)		Grid
	Internet-based (Volunteer DG)	LAN-based (Enterprise DG)	
Resource	Desktop * Anonymous resource provider	Desktop * within a corporation, university, etc.	Supercomputer, cluster, scientific instruments, database, storage,
Connection	-Non dedicated poor bandwidth -Immediate presence (connectivity) -Consider firewall, NAT, Dynamic address	-Non dedicated intermediate bandwidth -More constant connectivity than volunteer DG	Dedicated high speed, bandwidth
Heterogeneity	High heterogeneous * Need resource grouping	Intermediate heterogeneous * Less heterogeneous than volunteer DG	Low heterogeneous
Dedication	-Non-dedicated -High volatile * Need an incentive mechanism	-Non-dedicated -Low volatile (non-business hours) * Need an incentive mechanism	Dedicated * Be able to use reservation
Trust	Malicious volunteer * Need result certification	Low trustworthy resource provider	High trustworthy resource provider
Failure	Unreliable (faulty)	Unreliable * More reliable than volunteer DG	More reliable than desktop grid
Manageability	Individual-based administration * Totally distributed to individual * Difficult to manage	Individual-based administration * More controllable than volunteer DG	-Domain-based administration * Professional administrator
Application (job)	-Independent (mainly) -Computation-intensive (mainly) -High-throughput (mainly)	-Independent (mainly) -Computation-intensive (mainly) * Data-intensive (possible) -High throughput (mainly)	-Independent or dependent -Computation or data-intensive -High performance (mainly)

Table 2: Grids and OGs Analysis [45]

3.3 Cloud Computing

Cloud computing is a distributed model in which services are offered over the Internet on a pay as you go basis [48]. The infrastructure is used within an organization as private clouds or leased as utility computing services. In cloud environments software and infrastructure are offered as services through technologies like web services and virtualization. By using these technologies,

abstracted storage, computational power and networking resources are seamlessly offered to end users.

3.3.1 The Cloud Architecture

The National Institute of Standards and Technology (NIST) accepted definition of cloud computing is based on essential characteristics, service models and deployment models [49]. Their architecture (in Figure 5) suggested a logical language to interpret the cloud based on its main use cases.



Figure 5: Cloud Definition [49]

3.3.1.1 Essential Services

Cloud services have the following features which relate to or differ from traditional distributed computing practices [49]:

1. **On-demand-self-service:** The service allow resource requestors to run computing services without direct interaction with the service provider.
2. **Broad network access:** Enable heterogeneous devices and software services to be accessed over the network through standard mechanisms.
3. **Resource Pooling:** Service provider resources are combined to aid in the consumer multi-tenant model. Consumers can therefore access resources from an abstract level without control or knowledge of physical resource whereabouts.
4. **Rapid elasticity:** Resources can be rapidly and elastically provisioned and this presents an illusion of unlimited capabilities accessible at any time and quantity.

5. **Measured Service:** Resource metering enable transparency between service providers and consumers through optimized resource usage.

3.3.1.2 Services and Deployment Contexts

Software, Platform and Infrastructure (as a service) are the main cloud services [49] with more specialization and variations being offered for specific application contexts (e.g. Backend as a service and Payments as a service). Software as a Service (SaaS) allow consumers to access service provider applications through client devices. Consumers can deploy and enjoy administrator privileges over their applications on Platform as a Service (PaaS) without underlying cloud environment control. In Infrastructure as a Service (IaaS), the resource requestor has control over storage, processing and networking to include deployed applications [49].

Irrespective of the service models, four cloud deployment setups are used to cater for specific requirements [49]:

1. **Public Cloud:** Public clouds are made available to the general public as a utility computing service.
2. **Private Cloud:** The clouds mainly meet the daily needs of an organization and are normally isolated from the public.
3. **Community Clouds:** Organizations with shared requirements can set up community cloud to support a specific goal.
4. **Hybrid Clouds:** A hybrid cloud infrastructure is a fusion of two or more cloud deployment models.

3.3.2 The Cloud Realization

Discussions in the distributed computing community have either established grid and cloud computing as same phenomenon or clouds as being merely an extension of grid computing. This is an excerpt from IBMs whitepaper by Kourpas [50]:

“Grid computing allows you to unite pools of servers, storage systems and networks into a single large system so you can deliver the power of multiple-systems resources to a single user point for a specific purpose. To a user the system appears to be a single enormous virtual computing system.”

Is Cloud Computing then just another synonym for Grid computing?

Both models progress the vision for seamless access to pooled multi-computational resources. However, there are differences on what makes a grid and cloud considering factors such as virtualization, business and programming model. Table 3 shows the differences between grid and cloud computing as discussed in [51]:

Characteristic	Cloud Computing	Grid Computing
Business Model	-Utility computing service model	- Project oriented
Utilisation	-Implement virtualisation to compute several tasks concurrently	-A single task is allocated to multiple servers to execute
Programming Model	-Mesh-up's and scripting are used as workflow technologies to integrate services and applications	-Technologies used in parallel programming are mainly used.

Table 3: Grid and Cloud Compared

The virtualisation technology separates cloud computing from grids. The technology provides an abstraction which unifies compute, storage and networking as a pool of resources allowing for services to be implemented on top [51]. This maximises computing power and resolve challenges faced in grid computing where computations continuously communicate (e.g. in parallel computing).

3.4 Distributed Storage

Distributed Storage Systems (DSS) are computer networks which provide reliable access to data redundantly stored on a model of distributed nodes [52]. Analysis of bulk data sets to drive scientific discoveries in research have improved the development of high-end DSS. An alternative approach to these high-end storage services in recent years now harnesses the storage potential from commodity workstations in the same way idle Central Processing Unit (CPU) cycles are integrated in a number of VC projects [53]. The Google File System (GFS) is an example of a scalable storage infrastructure that has realised the utility of inexpensive commodity hardware [54]. To date the GFS is a useful platform for running the Bigtable system,

which is currently hosting a number of Google applications such as Google Maps, Google Code Hosting and MapReduce [55], [56].

3.4.1 Google File System

GFS is a distributed system which allocates storage facilities to multiple clients. The commodity hardware based file system offers a storage platform for Google's research and data processing needs [54]. The design space examined projected application workloads in traditional distributed file systems. The following are essential to the system [54]:

1. Since cheap components are utilized; "constant monitoring, error detection, fault tolerance and atomic recovery" [54] mechanisms were incorporated to handle system failures;
2. Design assumptions such as I/O operations and block sizes were reconsidered to manage exponential growths in data sets;
3. File access patterns by analysis programs and data streams motivated for atomic record appends as a performance optimization strategy.

In addition to appends (which allow a number of clients to append data to a single file with preserved append atomicity), GFS also offers snapshot, a mechanism to checkpoint current states before experimenting with mutations [54].

3.4.1.1 GFS Architecture

The master, multiple clients and chunk-servers each typically running on a Linux distribution are integral GFS components [54]. The client's fixed-sized chunks identified by global chunk handles are stored in chunk-servers as Linux files. Since GFS employs atomic record appends; chunk data reads and writes are mainly depended on chunk handles and byte offsets. Replication of individual chunks at a factor three increased the system's fault tolerance and data redundancy. To consistently handle client requests, GFS's master retains access control information, file to chunk mappings and chunk location file metadata [54].

Traditional file writes required a client to specify the data and byte offset to write data. The redefined functionality however limits the client to data specification and isolates the client from back-end byte offset logic. The GFS's design redefinition reduced on complicated and expensive synchronization as was experienced with traditional writes. GFS doesn't provide assurance on

replicas being byte wise identical but only that data is written more than once at a GFS defined byte offset [54].

For a file chunked and appended; Figure 6 shows the interactions in a read operation. A client translates the filename and byte offset parameters to appended chunks into a chunk index. The client then requests chunk-server replica locations from the master using the chunk index. Leveraging the master returned data; the client queries for replicas in proximity. Chunk location data is not saved persistently by the master but requested through master control messages send each time chunk-servers join the cluster.

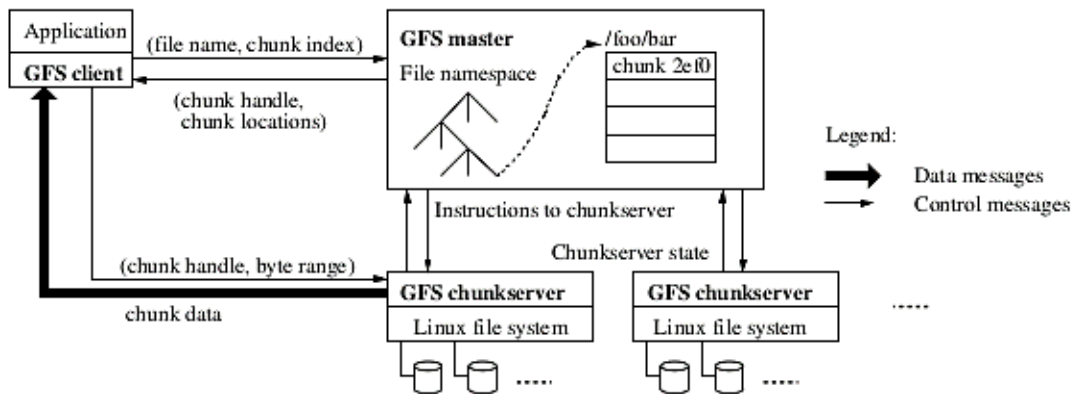


Figure 6: GFS Architecture [54]

3.5 Conclusion

This chapter presented and justified the research's opportunistic computing perspective and its classification in distributed computing. OGs were concluded as different from grid computing in view of resource types, connectivity, dedication and trust characteristics. The VC paradigm was then accepted as appropriate based on the nature of resource providers in the ICT4D context within which the research is conducted. Examples VC projects and DSS which integrate compute and storage resources were then revised. The cloud's business model, programming model and virtualisation technologies distinguished the model from all distributed computing approaches considered.

Chapter 4

4 Multi Agent Systems

This chapter presents the MAS overview in terms of agent communication strategies, coordination protocols and architectures. Literature on MAS approaches will be presented to motivate the agent technology. The second part of the chapter will discuss on knowledge presentation and reasoning in MAS. To explore ways in which MAOG services can adapt to dynamic environments, rule based system concepts are analysed.

4.1 Agent Technology

Agents with goal and task interaction patterns in competitive and cooperative scenarios are known as MAS. MAS offer reliable means of natural understanding, design and implementation of complex distributed software. There are separate views on the definition and concepts around agents. Genesereth [57] described agents as programs which interoperate using an expressive language. Based on this idea, MAS were acknowledged as a pool of autonomous code communicating using an Agent Communication Language (ACL). For MAS to achieve specific objectives, Wooldridge and Jennings emphasized for proactive, responsive and social characteristics in agents [58]:

- Pro-activeness classify agents as entities with goal-directed behaviour;
- Responsiveness specify agents' ability to perceive and act to environment changes;
- The social attribute integrates interactivity between agents (and possibly humans).

The above properties summarize the weak notion of agents [22]. An area of active research and widespread controversy known as strong agency assumes further humanistic and mental properties (e.g. belief, desire and intention) [59].

4.2 Multi-Agent Systems Rationale

Distributed Artificial Intelligence (DAI) studies the creation and application of MAS in pursuit of specific goals [60]. The domain has established inter-disciplinary concepts in artificial intelligence, sociology and computer science. The study highlights the following incentives in MAS [61]:

1. MAS offer parallelism, robustness and scalability. These are important in the integration of knowledge sources and processing of data sets which cannot be handled by centralised systems;
2. MAS build around artificial intelligence, psychology and sociology which propose interactivity and intelligence as deeply coupled. MAS appreciate the coupling in both ways; that is, interactivity allow agents to increase their intelligence and equally, intelligence facilitates the efficiency of agent interactivity;
3. Exploring MAS from DAI helps understand agents from complex social phenomena (i.e. emergent behaviour and collective intelligence);
4. Presently powerful computers and applications are becoming tightly linked with innovations in long-range networks. MAS provide innovative ways for managing connected computing infrastructures with insights from interconnecting existing legacy systems.

4.3 FIPA Compliance

The Foundation for Intelligent Physical Agents (FIPA) [62] is a standardisation board which promotes the effort of regulating agent technologies. FIPA is described as bundled up expertise which is easily included in “complex systems with a high degree of interoperability” [63]. The FIPA97 specifications defined normative rules which enabled for interoperability and management of societies of agents [63]. Of importance was the agent platform reference model which classifies key roles or agents required for platform management services. The Agent Management System (AMS), Agent Communication Channel (ACC) and Directory Facilitator (DF) are key agents/roles identified into the agent platform. The ACL for inter-agent communication through message passing was also defined by “setting out the encoding, semantics and pragmatics of the messages” [63].

4.4 MAS Communication

Agents require a universal language to define agent views and requirements. Specifically they interact by using unique languages called Agent Communication Languages (ACLs). ACLs originated from the need to model frameworks for agents to convey information in distributed computing environments [64]. The communicative languages typically exist in the logical layer above the transport protocols. Communication concerns at the data and message level are

handled by transport protocols with ACLs addressing communication on the social and intentional layer [65]. The design basis of ACLs evolve around heterogeneity, coordination, cooperation, interoperability, transparency and performance [64]:

1. Heterogeneity principle: Agents must communicate irrespective of the underlying environment;
2. Coordination and cooperation highlight the need for unique ACLs in complex problem solving. The ACL model should provide means of exchanging information on agent knowledge and its environment;
3. Interoperability assures the need for ACLs in agent interoperability;
4. The complexity of underlying ACL specifications should be hidden from MAS. Transparency underscores the need for ACL APIs which deprive agents from specific details and sets interactions to a higher abstraction;
5. Performance states that it's binding for ACL implementations to use system resources efficiently (i.e. CPU and Memory).

The development of ACLs progressed from Knowledge Query Manipulation Language (KQML) and Knowledge Interchange Format (KIF) to the most recent FIPA ACL.

4.4.1 KQML and KIF

KQML [66] is the first inter-project ACL proposed by the Advanced Research Projects Agency's Knowledge-Sharing Effort consortium. The knowledge sharing initiative comprise of two components: the KIF which describes message content; and KQML dedicated to system component interactions at runtime. Communication (message ID, sender and receiver), message (performatives and message format) and content (ontology, content language, and message content) are layers in KQML [64]. KQML support communication between agents with reserved primitives called performatives with message content description defined by KIF.

4.4.2 FIPA ACL

The FIPA ACL is the most adopted and studied ACL which includes various characteristics of KQML. FIPA ACL is lightly similar to KQML but differ in the syntax used to classify reserved primitives. Messages in FIPA ACL are viewed as communicative acts which aim to achieve certain actions [65]. FIPA ACL semantics based on the "speech act theory interprets human

natural language as requests, suggestions, commitments and replies” [67]. These communicative acts describe communication as a function achieved by the act of communicating [68].

4.5 MAS Coordination

Agent negotiation is a decision making mechanism in which MAS jointly search for mutually agreed solutions to problems in collaborative and competitive situations. These kinds of interactions enable a group of agents to act and participate in a rational way. Agents require interaction because:

- Agent goals can conflict with specific actions;
- Agents possess varying abilities and knowledge;
- Some system goals are achieved through collaborative effort.

A number of agent coordination strategies are proposed for task allocation. The English Auction, Dutch Auction and CNP customized for specific domains complements a list of FIPA protocols [69]. For cooperative problem solving in MAS applications, the CNP is the most utilized [70],[71].

In the CNP coordination approach, agents assume manager and/or contractor roles. “A manager provides a task to be processed, with contractors being agents with capabilities to solve the problem” [48]. The agent responsibilities in the protocol are not defined prior. “Any agent can be a manager by issuing call for proposals specifying the task allocation criteria” [48]. The premise of CNP stems from the fact that if an agent doesn’t have adequate resources to solve an allocated problem using native expertise, it decomposes the problem and discovers alternative agents.

4.5.1 MAS Negotiation Strategies

Negotiation strategies enable for flexible access to services in distributed systems. Wong and Yu [72] introduce an architecture for multi-product supplier selection considering synergy between products. Selecting suitable suppliers enhance performance since services are provided at the right time. If purchases are in bulk, the design highlighted the possibility of synergy between products which affect supplier choice. The efficiency of model was introduced in three phases

which include: product synergy determination, supplier pre-selection and negotiation-based final selection [72].

An agent based negotiation mechanism for data storage and product information was discussed in [73]. The e-commerce automated strategy addressed high data organization costs in cloud environments. Agents with specific requirements were utilized by buyer and seller participants which facilitated for a fast and reliable bilateral negotiation process. Users passed hash coded requirements to secure the negotiation process. The report focused on the preliminary application of MAS negotiation without the e-commerce specifics [73].

Zhang and Ren explored the Bayesian approach to agent preference prediction in bilateral multi-issue negotiation. In competitive MAS, self-interested agents may hide their preferences which complicate mutually beneficial negotiations. As per the paper, the Bayesian theory analyses historical opponents' offers to predict preference over negotiation issues. A counter offer proposition algorithm was incorporated to facilitate in MAS mutual offers. The Bayesian approach reduced the negotiation time and integrated utility to agents that implement the functionality from conducted evaluations [74].

An et al [75] present a MAS based negotiation approach to dynamic resource allocation in distributed settings such as clouds. Buyers and sellers interact simultaneously with representative agents allowed to decommit from an agreement at a cost. The cost for agreement decommitment improved the resource allocation mechanism. The use of bilateral bargaining and defining heuristics to aid decision making provided a limited number of closed form results [75].

A basis for intelligent Service Level Agreement (SLA) bilateral bargaining between SaaS brokers and multiple resource providers was introduced for cloud infrastructures [76]. The research introduced SaaS brokers on behalf of customers to provide a one-stop-shop for offering customer service. An investigation on counter offer generation strategies and decision making heuristics introduced how the techniques are important in implementing specific goals [76].

Maclaren et al [77] discuss how MAS and CNP based SLA negotiations in grid computing optimise infrastructures for efficient job scheduling. The CNP bidding mechanism enhanced grid scheduling workflows since busy agents need not bid for a contract. To cater for emergent

failures in dynamic and heterogeneous distributed settings, SLA renegotiation mechanisms were incorporated [77].

The MAOG perspective to resource negotiation differs from some strategies discussed. In the same way, this research adopts the CNP to identify available agent components and therefore idle shared resource capabilities with a different context of application.

4.6 MAS Challenges

The construction of purely goal directed or reactive agent based distributed applications is fairly achievable. Challenges arise in implementing MAS which appreciate the balance between goal-directed and reactive behaviour [78]. DAI then address questions on when and how agents should cooperate or compete to meet design objectives [78]. Two routes based on the micro (agent) and macro (group) levels are used to examine these questions [78], [79]:

- Bottom-up: Searches for “agent-level capabilities which result in interaction at the overall group level”;
- Top-down: Searches for “specific group-level conventions or norms” which constrain the interaction at individual agents’ level.

Further concerns arise on expressing logical relationships between the micro and macro levels. The micro-macro problem present problems in MAS considering [78]:

- how communication is enabled by ACL communication languages;
- how decision making can be activated by utilizing knowledge provided by other agents;
- how reasoning on the state of the interaction environment can be integrated;
- The balance between local computation and communication.

Integrating solutions to these concerns in the MAOG solution through an agent development methodology enabled for a solution with problem solving capabilities at the same time balancing on the goal-directed and reactive behaviors.

4.7 MAS Development Platforms

To implement complex agent systems, MAS development platforms are required. In early years, lack of environments where agents can communicate to achieve desired goals presented

obstacles to the proliferation of agent technologies [80]. Notwithstanding the availability of a plethora of agent platforms, there is currently no definite consensus or a universal approach to agent development in literature. The application contexts of multi-agent platforms depend on characteristics such as standards compliance and scalability. There are roughly three classes of agent platforms: those which specialize in internal agent reasoning, those that focus on inter-agent communication, and Mobile Agents (MAs) [65]. Examples of agent development environments include: Aglets, Anchor, JADE and Zeus.

4.7.1 Aglets

Aglets [81], [82] is an environment for implementing MAs in Java. The Aglets Core and Proxy are the main platform components. All agent internal methods and variables are confined to the Aglets Core; and the Proxy acts as an interface to the Core. Developing standalone MAs is administered by the Aglets Workbench. The Aglets Building Environment comes with Tahiti server and Fiji (Agent web launcher). Tahiti mainly provide mechanisms for agent dispatch and mobility [83].

For agents to communicate, synchronous and asynchronous message passing methods are implemented. The lack of good security mechanisms and scalability are major issues associated with Aglets. The downside results in the state of Aglets not being saveable on any host, and interoperability issues with other platforms [80].

4.7.2 Anchor

The Anchor [84] project developed by Lawrence Berkeley National Laboratory offers secure management and transmission of MAs in distributed settings. Anchor based on Aglets abstracts agents as Java objects which migrate between networked hosts encapsulating state and code. Executions resume on reaching remote hosts. Within Anchor, the agent server run-time environment conducts critical system functions (e.g. agent creation). The run-time environment addresses trust, code integrity, fault tolerance and secure communication issues. The Secure Socket Layer (SSL) and Akenti [85] provide functionality for mutual authentication and access control on resources accessed respectively.

4.7.3 Zeus

Zeus [86], [87] simplifies the implementation of cooperative agent-based applications. The platform complies with FIPA specifications, is open source and implemented in Java. The approach views an agent as composed of three layers [86]:

- definition layer - an agent is viewed as an autonomous reasoning component;
- coordination layer - the agent is considered social;
- The organisational layer is focused on agent associations.

The platform presents agent coordination, rational agent theory and knowledge representation concepts to practical concerns in constructing MAS. Research documented in [80], however certified the platform's lack of support for agent mobility as its main disadvantage.

4.7.4 JADE MAS

JADE MAS is a Java software for constructing peer-to-peer multi-agent applications. JADE implements distributed interoperable systems through compliancy with FIPA standards [88], [89]. The following characteristics are offered by the framework [90]:

1. **Agents are autonomous and proactive:** An agent has a single thread of execution which is useful in agent life cycle control and automatic resolution of actions to perform;
2. **Agents are loosely coupled:** Agents communicate through asynchronous message passing. An agent which initiates a communication addresses a receiver using an Agent Identifier (AID). This eliminates the sender-receiver object reference dependency;
3. **The system is peer-to-peer:** An agent can join, leave or discover other agents in the same platform by querying white and yellow page services. Individual agents can initiate communication and can equally be objects of incoming messages.

4.7.4.1 JADE Architecture

The FIPA97 specification is the basis of JADE. The toolkit consists of runtime instances (containers) distributed over the network [48]. The main-container is a unique runtime instance which represents the bootstrap point of any platform. A single main-container exists in a platform to register other containers. If a separate main-container is initialised elsewhere on the network, it constitutes a standalone platform to which other containers can possibly register [48].

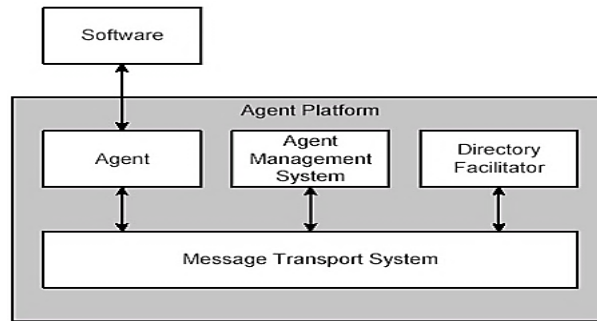


Figure 7: JADE Architecture

The “main-container is not a bottleneck in a platform, but it remains a single point of failure” [90]. Two agents/roles hosted by the main-container are initialised each time JADE is executed [90]:

1. **AMS**: the AMS administrates access to and use of a platform. Agents created in a platform are required to register with AMS to acquire a valid AID.
2. **DF**: the agent implements a yellow page service. It is used by agents for service registration. Agent subscriptions to be notified on specific platform service modifications can also be defined.

4.7.4.2 Agent Tasks: Behaviours

Agent objects have a set of behaviours which execute specific actions. JADE behaviour scheduling is non-pre-emptive. Developers hence resolve when behaviours execute or switch to give precedence to others. The ability to activate and block specific methods in JADE makes the scheduling process flexible. The following are JADE’s abstract behaviour classes [91]:

1. **OneShotBehaviour**: OneShotBehaviour is an operation designed to complete in one execution step;
2. **CyclicBehaviour**: CyclicBehaviour execute continuously until an agent terminates. The behaviour is suitable for functionality which executes in the background and wait for specific requests;
3. **TickerBehaviour**: A TickerBehaviour class is implemented in an agent object to perform actions which execute periodically as defined by a defined time interval.
4. **GenericBehaviour**: A GenericBehaviour executes sequential operations based on a status value. The behaviour is useful when interacting agents are dependent on knowledge

provided by each party. In particular, the approach is commonly used to implement the CNP negotiation mechanism (e.g. JADE book-trading example [92]).

4.7.4.3 JADE ACL

JADE complies with FIPA ACL message specifications which define mandatory performatives required of all ACL messages [93]. Sender, receiver and message content parameters are also defined. The attributes in Table 4 can be defined in an ACL message.

Parameter	Category of Parameters
performative	Type of communicative acts
sender	Participant in communication
receiver	Participant in communication
reply-to	Participant in communication
content	Content of Message
language	Description of Content
encoding	Description of Content
ontology	Description of Content
protocol	Control of conversation
Conversation-id	Control of conversation
reply-with	Control of conversation
in-reply-to	Control of conversation
reply-by	Control of conversation

Table 4: Message Parameters [93]

ACL messages are implemented as objects. The send method (Figure 8) is used to forward a created message. Since agent communication is based on asynchronous message passing, active agents are assigned mailboxes for storing inbound and outbound messages.

```

ACLMessage out_bound = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
out_bound.addReceiver(storageagent[0]);
out_bound.setContent(FILENAME);
out_bound.setConversationId("Save_File");
out_bound.setReplyWith("out_bound"+System.currentTimeMillis());
myAgent.send(out_bound);

```

Figure 8: Message Object

Once a message reaches the preferred destination, the receive method reads the desired message from a message queue. Defining a performative constructor (e.g. `ACCEPT_PROPOSAL`) in a message template helps return messages matching a required pattern.

```

MessageTemplate receive = MessageTemplate
    .MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
ACLMessage msg = myAgent.receive(receive);
if (msg != null) {

```

Figure 9: Message Receive

4.7.4.4 Debugging Tools

The development of platforms distributed across multiple hosts is simplified through an assortment of debugging tools. The Remote Monitoring Agent (RMA), Sniffer Agent, Introspector Agent and Dummy Agent are examples of debugging agents used in implementing distributed JADE MAS applications:

- **Remote Monitoring Agent:** The RMA (Figure 10) provides a graphical management console for monitoring and managing platforms. The visual agent is composed of three node types: agent platform, agent and container. The tool integrates a tools menu in which Dummy, Sniffer and Introspector agents can be launched [94].

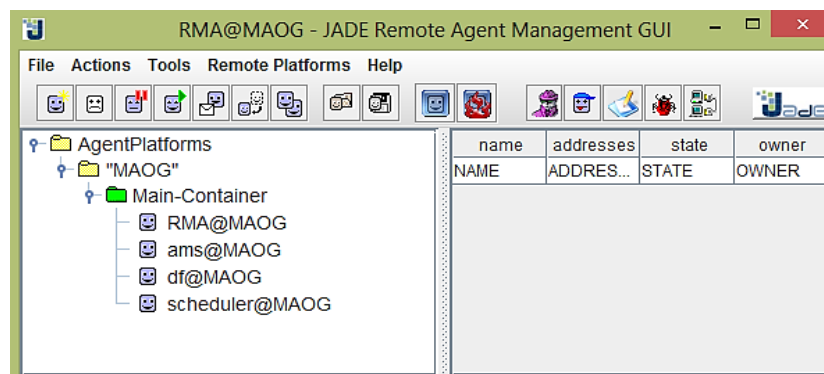


Figure 10: RMA

- **Sniffer Agent:** The agent allow developers to analyse interactions between agents. To intercept communication between targeted agents, a Sniffer Agent subscribes with the AMS to receive notifications on specific platform events [94].

- **Introspector agent:** An Introspector Agent monitors agent life-cycles and message queues. The functionality provide useful information on agent functionalities and behaviours active in a platform [94].
- **Dummy Agent:** The Dummy Agent is useful mainly in the development stage. It tests agent behaviours by sending custom ACL message stimuli [94].

4.7.5 Platform Evaluation

The use of MAS toolkits in distributed systems reduce development problems by isolating low-level implementation constraints. Adopting suitable MAS platforms from various agent toolkits depends on the problem domains. This section evaluates the MAS development environments reviewed in Section 4.7 based on defined metrics. The guideline is composed of [80]: standard compliance, communication techniques, license, security, migration techniques and agent mobility.

- **License:** Although open source toolkits have compromises on features compared to commercial environments; their availability allow for a larger developer base with access to its functionality;
- **Standard Compliance:** Standardizing agent platforms augments interoperability in agent-based applications. If an agent environment supports a reliable standard, its scalability and utility increases;
- **Communication Technique:** Asynchronous communication is advantageous compared to synchronous communication in MAS;
- **Security:** Security in agent toolkits make agent-based applications more attractive in handling mission critical services;
- **Agent Mobility:** Agent mobility in distributed applications typically reduce network traffic, increases responsiveness and supports disconnected computing (i.e. in MAs) [95];
- **Migration Technique:** Remote Method Invocation (RMI) enable function calls to remotely located subroutines. Compared to a selection of remote execution approaches (e.g. sockets), RMI consume more resources and time.

Table 5 shows the classification of agent platforms reviewed. Aglets and Anchor implement resourceful agent migration techniques based on sockets. But poor security coupled with lack of

standardization in both platforms affects their interoperability and scalability. Though Zeus is readily available, has good security and complies with FIPA specifications; the platform lacks agent mobility capabilities.

Agent Development Toolkits →	Aglets	Anchor	JADE	Zeus
Features↓				
Licence	Open-Source	Available in BSD license	Open-Source	Open-Source
Standard Compliance	MASIF	SSL, X.509	FIPA,CORBA	FIPA
Communication Technique	Synchronous , Asynchronous	Asynchronous	Asynchronous	Asynchronous
Security	Poor	Strong security	Good	Good
Agent Mobility	Weak	Weak	Not-so-weak	Supported
Migration Technique	Socket	Socket	RMI	Not supported

Table 5: Agent Platforms [80]

From a high level, JADE is favourable based on the evaluation metrics defined. Also, since communication is vital for MAS to achieve specific objectives; the way agent platforms handle communication and message services is important. The JADE Message Transport Service (MTS) transparently selects a transport mechanism and an optimum protocol which achieves the least message passing communication cost in MAS implementations [96].

Through the Message Transmission Protocol (MTP) and Internal Message Transport Protocol (IMTP) JADE interfaces, additional protocols can be added to the already supported Java RMI, Hypertext Transfer Protocol (HTTP) and Internet Inter-Object Resource Broker Protocol (IIOP) which increases the flexibility and scalability of agent systems. Apart from the communication services benefits, the ACC is integral by integrating system caches which eliminate the JADE main-container as a platform bottleneck [96].

Moreover, JADE provides homogeneous add-ons which are network and Java version independent. That is, the JADE run-time offers these add-ons for all Java environments (e.g.

Java Platform Enterprise Edition, Java Platform Micro Edition and Java Platform Standard Edition). This feature enable designers to develop and reuse identical application code for different platforms (e.g. Java mobile devices and computers) [96].

From the guideline and technical characteristics discussed, JADE was considered as a balanced agent platform to implement the MAOG distributed services. Therefore the MAOG platform will be built on the JADE MAS.

4.8 MAS Development Bottlenecks

MAS have the potential to improve the practice of designing and implementing distributed applications. Characteristic problems are however linked with agent systems directly attributed to agent-oriented software features. Mainly, MAS pursue specific system objectives while maintaining consistent interaction with defined execution environments. Incorporating such context awareness presents problems in designing software agents with stable support for both proactivity and reactivity. Additionally context sensitive decision making in MAS may result in uncertainty on which objective the agents pursue and methods to achieve the chosen objectives [97].

While agent sub-systems in MAS are modularised as attaining specific objectives for the parent system, the effects of their interactions are unpredictable. Primarily, the sub-systems resolve at run-time on objectives which require interaction; and which agents to interact with. Hence interaction aspects such as number, pattern, timing and outcome cannot be projected in advance. Secondly, emergent behaviours due to collective interactions which cannot be decomposed in terms of individual component's behaviour yield unexpected individual and group behaviour [97].

Less commitment in understanding the pragmatics of MAS development has been evident considering the proliferation of agent technologies [98]. For the MAOG system viability, pragmatic areas of agent system development such as social, conceptual, “analysis and design, micro (agent) level, macro (society) level and implementation” [98] are considered in this research.

4.9 Multi-Agent Distributed Computing

MAS have been extended to manage different distributed resources. Mobigrid, a framework for MAs in grid environments based on InteGrade [99] was presented by Barbosa and Goldman [100]. Aglets in the framework included support for encapsulating applications processed using a network of workstations. To support computations, a manager component was incorporated to keep track of agents submitted. From findings, the MA characteristic enabled InteGrade to reach zero idleness and offer transparency to machines through reduced performance loss.

An agent system for energy resource scheduling in power systems with distributed resources was proposed by Khambadkone et al [101]. The technology was applied to offer reliability and efficiency in integrating alternative energy sources. Results from simulations show that the system enabled for management of micro-sources with minimum operational cost.

Liu et al [102], proposed Ordinal Sharing Learning (OSL), a novel multi-agent reinforcement learning method for load balancing in Grids. Due to complex and dynamic grid environment characteristics, the approach avoids scalability issues by implementing multi-agent coordination with limited communication. The simulation results validate OSL as comparable to some centralized scheduling algorithms.

A minimalist decentralized algorithm for resource allocation in grid environments was suggested by Galstyan et al [103]. The idea was considered in a system of heterogeneous reinforcement learning agents which share resources for computational needs. Agents in the system only received job completion times without direct communication between them. The experiments recommended the effectiveness of reinforcement learning in improving quality of resource allocation in heterogeneous distributed systems.

There is considerable attention in MAS approaches for problem solving. Similar to the MobiGrid framework approach, this research extends the MA paradigm to process computationally intensive applications but differ in the agent development platform used. In addition, the anticipated MA module contains a rule based reasoning feature that enable workflows to be relocated when nodes are recalled which prioritises resource provider activity on their shared nodes.

4.10 Code Mobility in Distributed and Multi-Agent Systems

Code mobility is the ability of distributed systems to relocate code or objects from one host to another. The related mobile code technologies are classified as weak or strong mobility based on their ability to migrate the state of an executing thread. Strong mobility transfers code, data and execution state across to remote hosts where execution resumes therein. Weak mobility on the contrary allow only code and data to be moved [104]. Mobile code technologies don't unfold new functionality per se, but orchestrate for faster and flexible means of developing distributed applications [104]. MA and Remote Evaluation (REV) paradigms are mobile code model examples [48].

4.10.1 Mobile Agent

A MA is an executable code that moves amongst networked hosts according to the MA itinerary to achieve certain actions on behalf of its creator [104]. The code and data state transfer during a migration unlike the execution state. The program execution in this scenario suspends and waits for a resume state on migration [105]. To include support for execution state saving, research highlighted modifications to virtual machines, instrumentation of application source code and byte code; and modification of Java platform debugger architecture as the four basic approaches that can be utilized to capture the state of Java threads [106].

4.10.1.1 Advantages of MA Paradigm

The following are advantages offered by the MA paradigm in application development [80], [106]–[108]:

1. **Asynchronous and autonomous execution:** Deployment of MAs with embedded tasks that require continuous open connections result in low latency savings. MAs can be invoked into networks where they operate independently and synchronously without constant monitoring.
2. **Fault tolerance and robustness:** The agent's reactivity allow for construction of fault tolerant and robust distributed applications.
3. **Bandwidth consumption:** Network bandwidth usage is minimized as agents move computation code to data which reduces intermediate results passing.

4. **Heterogeneity:** MAs offer optimum conditions for seamless system integration as they are transport and hardware layer independent.
5. **Dynamic adaptability:** The agents through embedded functionality can perceive changes in their execution environment and react autonomously to these changes.

4.10.2 Remote Evaluation

REV [109], [104] conceptualize distributed systems as composed of machines connected by a communication link. In REV, a client sends instructions to a remote server with access to resources required [48]. On completion, results are returned to a requestor machine. REV offer flexibility in creating custom services, execution of complex tasks and is easier to implement compared to MA based systems (i.e. that require state and code management) [104]. In relation to client-server models, REV suggest that remote nodes do not only receive processing requests from a client but also instructions required for performing the operations [105].

4.11 Knowledge Presentation and Reasoning for MAS

Knowledge Representation and Reasoning (KRR) describe how symbolic rules are used to present knowledge. The acts of thinking using this knowledge introduce various aspects to the reasoning process. A number of Knowledge Representation (KR) techniques have been developed over the years, and these include formalizations from Artificial Intelligence and Web computing domains (e.g. Web Ontology Language, Ontology Inference Layer and Description Logic). A gap between what can be represented in theory and practical still exists and there is constant and continuing exploration of novel KR techniques. There is also increasingly more research exploring the coupling of KRR and MAS towards the development of knowledge-based intelligent MAS.

4.11.1 Rule Reasoning

Production Rule Systems (PRS) are computer programs that consist of a set of rules (productions or simple patterns) about predefined behaviours. PRS focus on solving problems by performing rule based reasoning making use of expert knowledge composed of "if-then" statements stored in rule bases [110].

A simple rule which represent knowledge about a specific domain is structured as follows:

If <conditions>
Then <conclusion>

The <conditions> specifies preconditions of a simple pattern with the <conclusion> representing the action taken. A rule is said to be triggered when the rule's precondition matches the current state of the world [111]. If a rule's conclusion statement is reached, it is classified as fired.

PRS were the first Artificial Intelligence software with potential to match the decision-making capabilities of human experts. An inference engine and knowledge base (facts and rules) subsystems make-up the core of a rule engine. The inference engine's sole purpose is to match rules in the knowledge base to well-known facts (data about current state or knowledge) to deduce new facts with the latter (knowledge base) concerned with representation of facts and rules.

Rules and facts are stored in the production memory and working memory respectively. The facts are declared in the working memory where they are altered regularly. In practical scenarios, rules can enter a state of conflict when a rule system with large sets of rules and facts result in many rules being true for the same fact assertion. In such cases the conflicting rules execution order is managed by an Agenda [112]. The Agenda is a rule system component that utilize conflict resolution strategies to determine which rules, “out of those that apply, have the highest priority and should be fired first” [113].

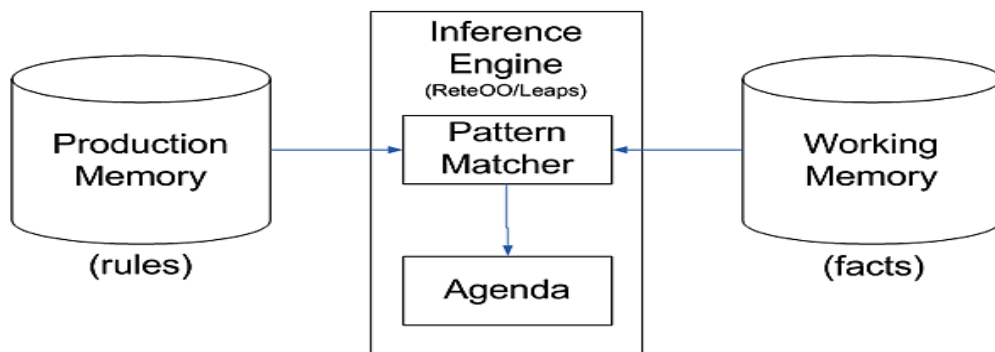


Figure 11: Rule Engine Components

4.11.2 Rule Engines

Rule engines are computer programs that deliver KRR functionality and execute a defined cycle made up of three states: match rules, select rules and execute rules [114]. An engine searches for

all rules satisfied by working memory contents. The various rule matches identified for execution are jointly referred to as the conflict set. The instantiation of the rule is a representative of the rule and a subset of matching data items [114].

When a conflict set is identified, it is converted to the select rules state where a selection strategy is invoked to determine rules to execute. The selected instantiations of a rule are then moved to the execute rules state where selected rules are fired [114]. There are two types of inference methods [111]:

1. Forward chaining: is data driven; it initializes with availability of data in the working memory and uses inference rules to obtain more data until a goal is reached. Pattern matching is conducted in the working memory until if clause (antecedent) known to be true is found. The engine then executes the then clause (consequent), subsequently pushing the new information to its data.
2. Rule engines utilizing backward chaining search inference rules until a rule with a consequent that matches a desired goal is identified. “If the rule antecedent is not known to be true, then it is added to the list of goals” [114]. The pattern matching method is goal-driven since the lists of goals determine rules to be selected [114].

Rule engines utilize algorithms such as Rete, Leaps and Treat. Rete is widely used in several applications due to its efficiency in pattern matching. Rete algorithm [115] is implemented by building a network of nodes and “creating an acyclic network of the rule premises; the so-called Rete network” [116]. The algorithm allows state saving in matching and re-computes changes only for modified facts. The matching process state is updated only as facts are added and removed. Due to the state saving functionality, fewer facts are added or removed which translates to a faster matching process.

4.11.2.1 SweetRules

SweetRules [117] is a set of tools for semantic web rules and ontologies revolving around the Rule Modeling Language (RuleML) standard. The tools can be easily merged with distributed rule-bases/ontologies due to their interoperability with various ontology languages such as the Semantic Web Rule Language (SWRL) and Jena [118]. Recent SweetRules revisions incorporate support for scalable backward and forward chaining.

4.11.2.2 F-OWL

F-OWL [119] is a Web Ontology Language (OWL) and Resource Description Framework (RDF) engine implemented using Prolog logic programming language and a Flora-2 extension (i.e. providing the F-logic frame-based layer). Primarily, F-OWL is a combination of an OWL engine and a frame based system that is utilized for reasoning with OWL ontologies. An OWL importer in F-OWL reads OWL ontology thereby extracting the RDF triples. After conversion of the extracted RDF triples into an F-OWL's supported format, the triples are fed into the F-OWL engine. Flora rules defined in flora-2 language are then utilized for ontology consistency check and knowledge extraction via resolution [119].

4.11.2.3 Drools and Jess

Drools [120] is the leading open source business rule management system and also a rule engine that reacts to data changes and affords enhanced querying capabilities. Jess [117] also implemented in Java develops software which can reason based on supplied declarative rules. A Jess scripting environment for Java object creation, method initializations and Java interface implementation is offered.

Drools and Jess extends the Rete algorithm (i.e. ReteOO) in pattern matching compared to available inference engines. From Section 4.11.2, Rete algorithms state saving characteristics translates to a faster matching process. Due to Rete support, Drools and Jess rule engines were selected for rule based reasoning functionality.

4.12 Conclusion

The sections introduced MAS and rule based reasoning concepts in detail. From a survey JADE was established as ideal for implementing the MAOG platform. The CNP was selected for task and resource allocation negotiation from literature. Drools and Jess engines were identified as attractive for integrating context awareness in MAS through the Rete algorithm support. The chapter ended with a discussion on rule based technologies which adapt MAS to dynamic uncertain environments. The next chapter introduces the MAS compute and storage services.

Chapter 5

5 MAOG Implementation Context and Requirements

This chapter introduces the grid solution through the fusion of reviewed technologies. The first sections present the ICT4D context in which the research is conducted including its connectivity and nature of resource providers. The chapter ends with a discussion on the prototypes and generic requirements expected of the grid system.

5.1 The Siyakhula Living Lab (SLL) Context

Section 3.1 “Opportunistic Grids” highlighted the different usage contexts within which OGs have been deployed. In the case of MAOG, the SLL ICT4D project is the primary target context of implementation. This project provides the basis for the formalization of the requirements for MAOG system based both on the environment factors (e.g. available computing infrastructure and resources) and also usage profile of the computing resource in these contexts.

The SLL project is based in a rural community called Dwesa which is located in South Africa. The project seeks to explore novel approaches in addressing societal challenges [5], [121]. The primary approach towards addressing these societal challenges is through the deployment of networked DANs, through which the community is able to access Internet based information and services. The seventeen (17) deployed DANs largely consist of computer laboratories that have been deployed at specific schools.

The SLL model relies on a wireless broadband island realised through a blend of fixed and mobile WiMAX links connected to the Internet through VSAT technologies [122]. Alvarion BreezeMAX WiMAX technologies were used to build the wireless local access loop and inter-connecting the points of presence since fixed line infrastructures lack in the region [11]. The Community Access Point (CAP) gateway running a Point-to-Point (PPP) client over Ethernet (PPPoE) enables machines within schools to access the wider local network. Once a PPP client for a school authenticates with an access concentrator and establishes a link; outgoing traffic (e.g. VoIP traffic, Internet) is then routed to the next hop [11]. The SLL network structure is shown in Figure 12.

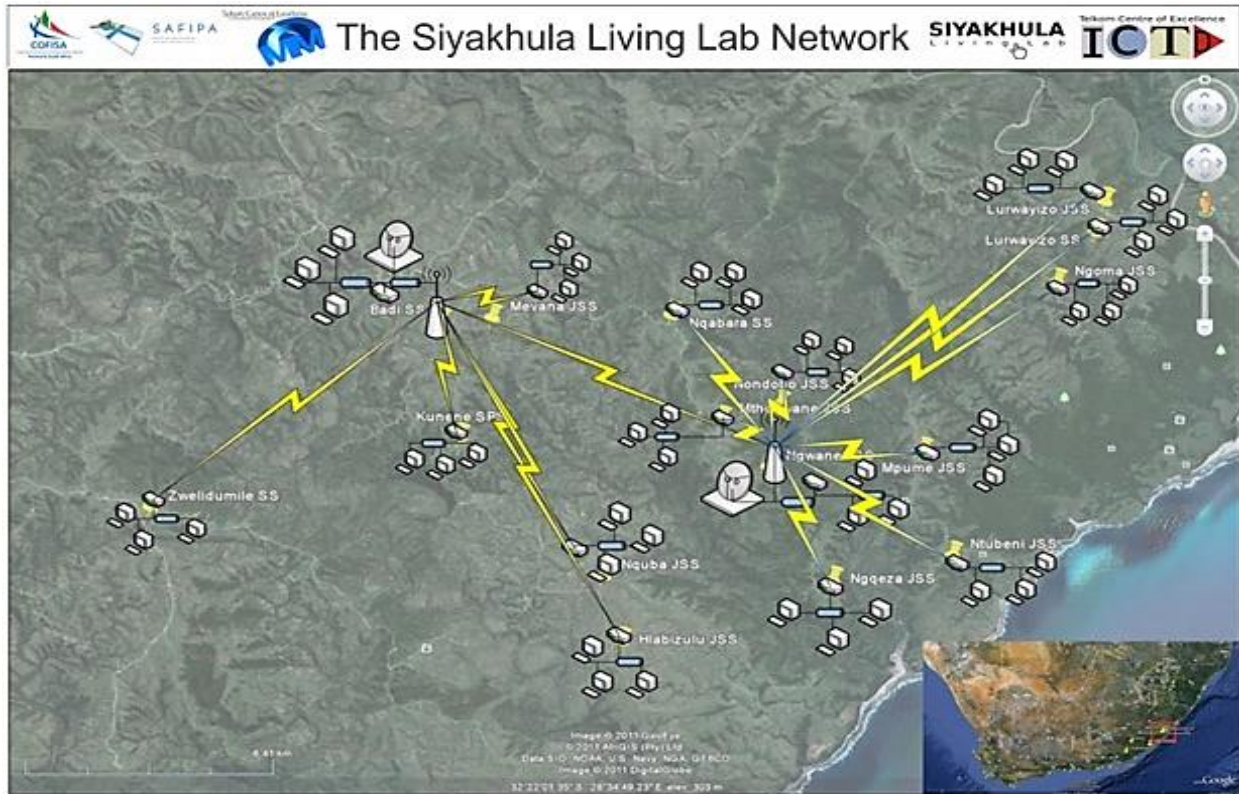


Figure 12: SLL Connectivity

5.2 MAOG Services and Requirements

A MAOG system which offer transparent and pervasive grid user access to shared idle resources in ICT4D contexts is presented in this work. The VC paradigm can open up computing to previously marginalised backgrounds to participate in the power of Internet-based volunteer networks and services.

The evolving trends in high performance computing and DSS (introduced in Section 3.4) motivated the compute and storage services as crucial services that can be incorporated in the MAOG system. The services were implemented and evaluated to recommend optimum techniques for the Dwesa ICT4D context. Demonstrating the viability of the grid solution through the fusion of introduced technologies was crucial in designing phase.

The compute and storage components identified for the MAOG system are listed further on:

1. Computational Component
 - a. MA compute component

2. Storage Component

- a. Disk Mirroring (DM) file-system
- b. Chunking with Atomic append (C-AP) file-system.

5.2.1 Compute Component

The MA and REV paradigms (introduced in Section 4.10) address limitations prevalent in classic distributed computing architectures. The REV approach adopts the client-server model. The model presents limitations considering the context within which the compute component is formulated. The client in REV functions properly by sending instructions to servers (nodes) and receiving responses. That is, the handshake in REV is continuous, where each request/response requires a complete round trip across the network [108]. The method isn't suitable for a VC context as it incorporates some performance overheads.

In the MA paradigm the client doesn't communicate with servers but migrates to the nodes. Buchanan et al [108] discuss the following merits offered by the MA paradigm in distributed applications:

- The migrations reduce bandwidth problems and eliminates repetitive request/response handshakes by moving a transaction from client node to the server;
- The MA model solves against intermittent network connections as agents can be created for offline computing and communicate results when applications are back online.

Inclusive of the above highlighted characteristics; a MA allows system level functionality (e.g. rule based reasoning) to be integrated. In view of the discussed merits, the MA paradigm was utilised for the design and implementation of the compute component.

5.2.2 Storage Component

This section introduce the MAS based file-systems which join shared disk space into one distributed storage resource.

5.2.2.1 Disk Mirroring File-System

DM in storage systems “replicates logical disk drives onto separate physical hard disks to ensure continuous availability” [123]. DM can also achieve data mirroring which makes exact copies of files available on separate nodes in the same system. In node failures, the DSS can recover data

flexibly from volumes in proximity. Apart from high data availability through redundancy, mirroring allow for concurrent reads which can significantly improve performance in specific system circumstances [123]. The DM file-system was hence designed to explore error recovery characteristics offered through data replication to address erratic shared storage availability.

5.2.2.2 C-AP File-System

The HTTP language's file and drag-and-drop APIs utilize chunked uploading to address problems associated with large file uploads. Uploads in excess of several gigabytes on unreliable networks often leads to failure and increased upload times. To target users with slow Internet connectivity, the APIs break files into fragments and send the chunks to upload servers.

The GFS utilizes chunked uploading to optimize on file size and network usage. The GFS also extends the atomic append technique to optimize on file storage in distributed nodes. Contrary to traditional storage approaches which write multiple data fragments in a region of storage; chunk appends reduce client synchronizations resulting in high read speeds on accessing the individual chunks. The C-AP file-system builds on the GFS's optimizations (introduced in Section 3.4.1). Considering the SLL network, sending file uploads as a series of chunks to volunteered nodes can enhance file upload and download transfer speeds.

5.2.3 Non-Functional MAOG Requirements

This section outlines the MAOG non-functional requirements without emphasis on the technology used. In software engineering, non-functional requirements are measures used to validate system functionality [124]. The requirements are considered in design trade-offs when designers specify structural and behavioural system aspects [125]. The following non-functional requirements are inclusive for the compute and storage components:

1. **Ease of setup:** Shared compute and storage resources are integral to the system and should be easy to set-up.
2. **Requesting a service:** The platforms should allow users to submit computational applications and uploads easily.
3. **Heterogeneity:** The system services should be platform independent.
4. **Autonomy:** System workflows should prioritise resource provider activity on shared nodes.

5. **System Uptime:** The systems should be readily available to provide required services.
6. **Response and Turnaround time:** the system should keep at minimum the time taken to implement requested services.
7. **Security:** Message integrity and confidentiality are integral security measures in the MAOG services.

5.3 Conclusion

The context within which this research is conducted was presented in this chapter. The connectivity framework as introduced can support an integrated commodity computing resource from shared resources in DANs. Specific compute and storage services including motivations were hence presented to propose techniques in utilising these infrastructures. Generic non-functional requirements universal for the identified services were then listed at the end.

Chapter 6

6 The MAOG System

In this chapter, the compute and storage services founded on the MAS methodology accepted in Section 2.1.4.4 are introduced. The architectures, agent participants and acquaintance relationships present the identified compute and storage services using the agent technology. The MAOG services' use case and sequence diagrams introduce the functional requirements of key components identified.

6.1 Compute Component

The MA compute component which utilizes idle shared CPU cycles through VC is considered and designed in this section. The main goal was to come up with a MAS infrastructure which solves computationally intensive tasks using volunteered processor capabilities. The process of designing the MA compute service is discussed in the following sub-sections.

6.1.1 Mobile Agent Platform

The MA platform's architecture and its agent delegation model are discussed in the following sub-sections. The CNP (introduced in Section 4.5) selects an optimum platform with readily available processing cycles from a selection of shared nodes. The service also considered resource provider activity on their shared nodes to avoid computations getting into their way. The platform assumes that shared resources and their representative agents are connected via the SLL network with easily detectable failures.

6.1.1.1 MA Architecture

The platform is composed of autonomous and collaborative agents with knowledge of their deployment environment. The architecture defined by JADE and its features is shown in Figure 13. Processing User Agent (PUA), Processing Mobile Agent (PMA), Processing Mobile Agent Graphical User Interface (PMA GUI), Node Processing Agent (NPA) and Processing Resolver Agent (PRA) agent types with different capabilities and view of the system were designed. The agents obtain and share knowledge required in solving compute oriented problems.

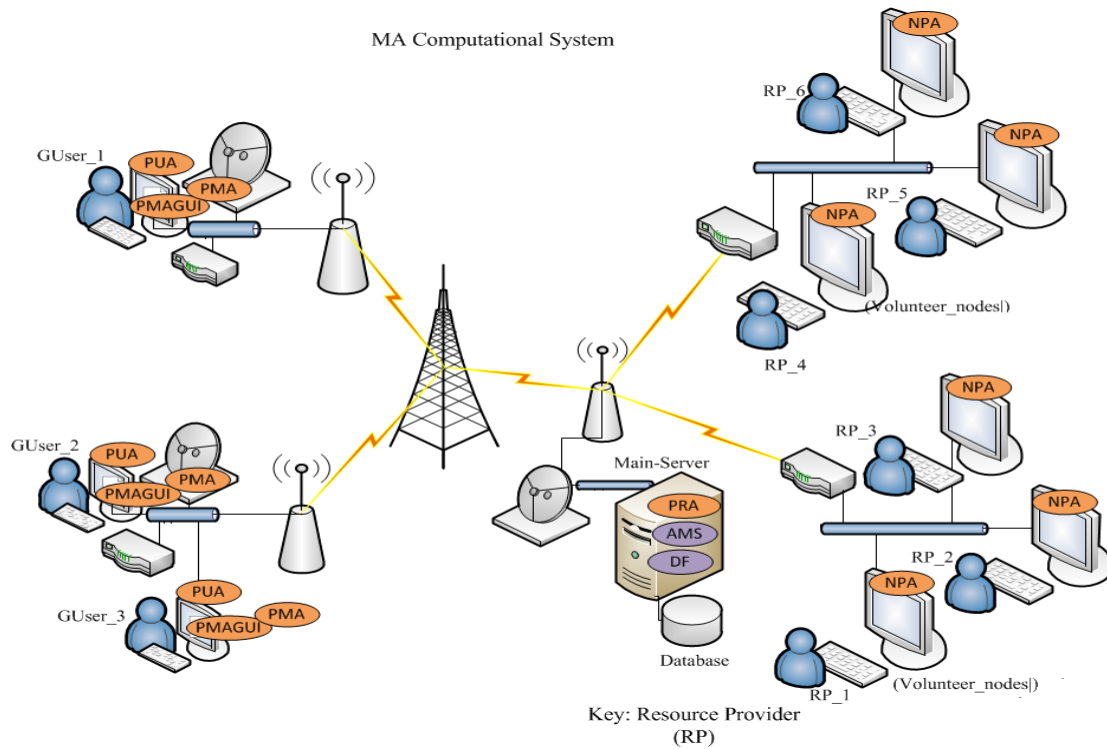


Figure 13: MA Architecture

The following are the descriptions of the main agent types and their responsibilities in the infrastructure:

1. **PUA**: From a high level; PUA request processing resources matching a specific criterion on behalf of users. Initiating the CNP mechanism, evaluation of proposed PRA bids and contract allocation are the main primary execution steps;
2. **PMA**: This agent encapsulates the user problem and the rule based reasoning functionality to be validated;
3. **PMAGUI**: The agent provides a graphical user interface for users to invoke, load and deploy PMAs;
4. **Database**: The database stores shared node system information and CPU utilization patterns provided by NPAs;
5. **NPA**: NPA detects node processor utilization dynamics and profiles the information in a database;
6. **PRA**: The module identifies the SLL back-end shared nodes to the MA platform. In responds to call for proposals, PRAs determine the aggregate processor load for shared computers in their respective platforms and return the values as a bids to requestor PUAs.

6.1.1.2 System Specifications

This is discussed in consideration of two participating parties; users and resource providers. Users execute their applications using resource provider machines. Both parties have access to modules which define their roles in the platform. The non-functional requirements in Section 5.2.3 are equally applicable in this platform. The functional requirements which outline the main behaviorally related interactions performed by the participants in dialog with the system include:

1. The CNP negotiation strategy should recommend an optimum computational platform;
2. A feature for selecting nodes, application loading and deployment should be integrated;
3. The system should monitor resource provider activity during problem solving to relocate work-flows in an event of a node recall;
4. The system must provide feedback on the results to users.

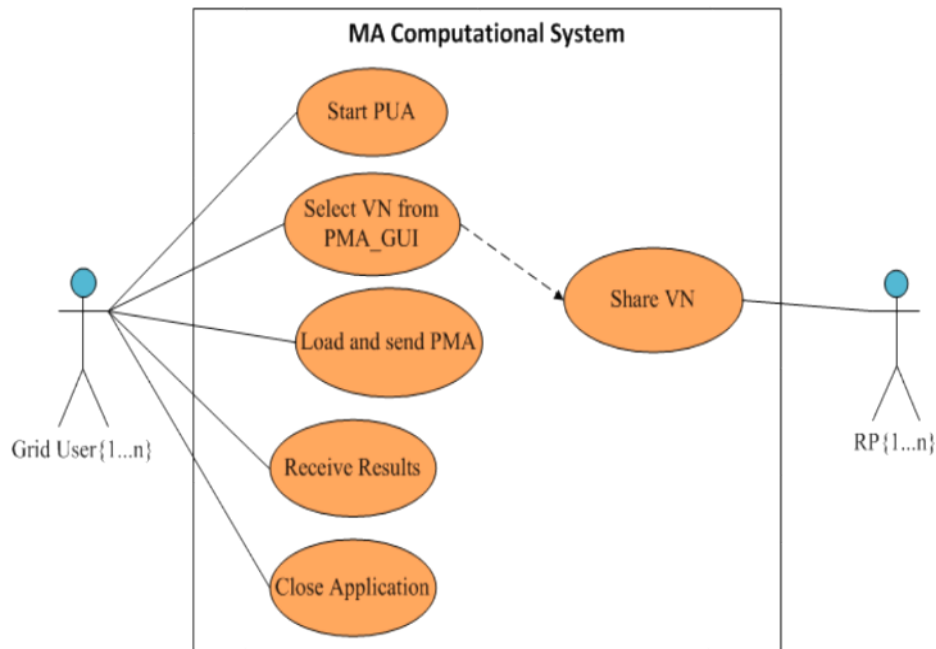


Figure 14: MA Use Case

The actions performed by the participants in line with the expected functional requirements are shown in Figure 14.

6.1.1.3 Agent Interactions

The MA component acquaintance interactions in problem solving are discussed in this section. To assess the CNP approach, three MA platforms are connected by a proxy-server. The resultant

converged MA infrastructure and its sequence diagram are shown in Figure 15 and Figure 17 respectively. The following conditions were defined: (1) The proxy adds separate platforms through their PRAs and (2) a resource provider shares a resource by initializing a NPA thereby joining a node. The parameters shown in Figure 16 are updated in node_cpu_util_infos after a defined interval.

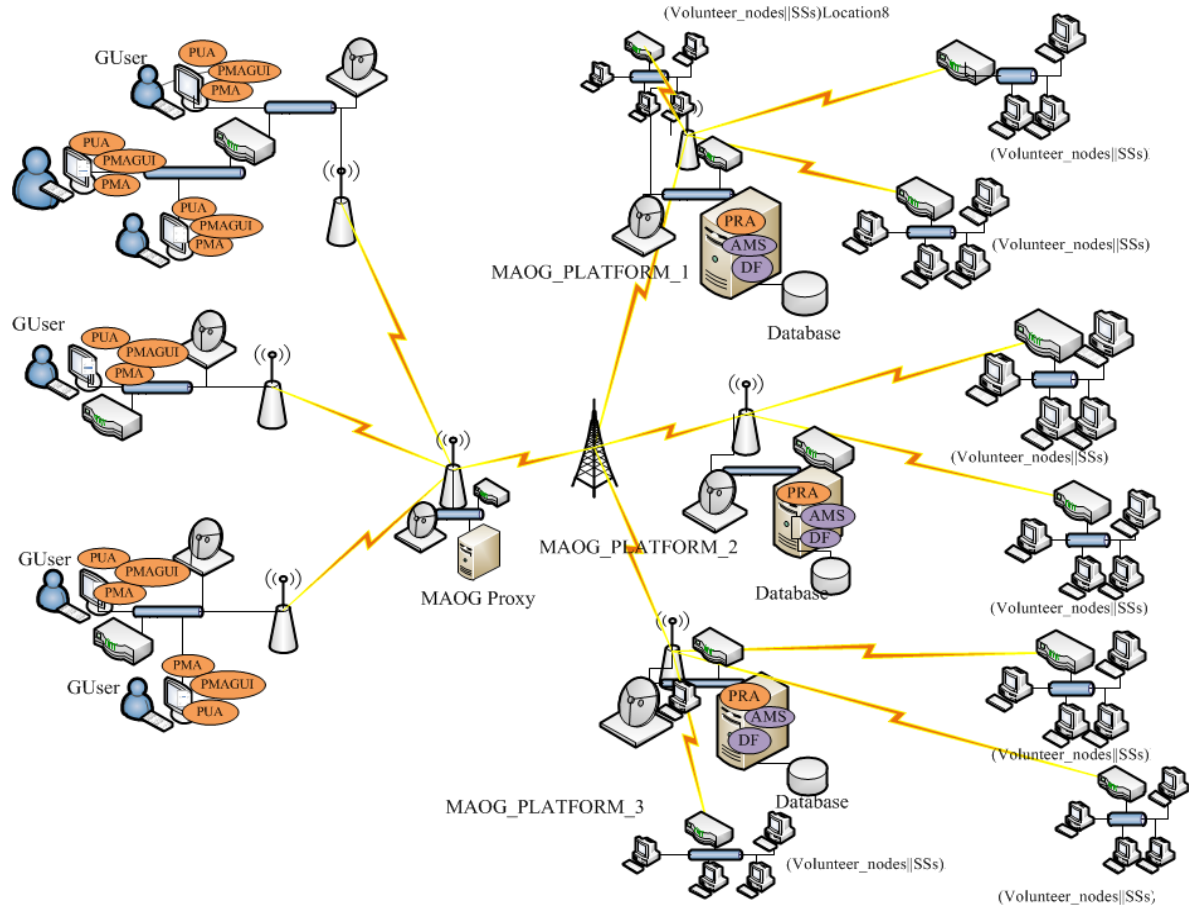


Figure 15: MA Infrastructure

When a user initializes a PUA to request shared resources from the compute service; the user oriented agent searches the DF for registered PRAs and encapsulates the identities in INFORM_SCHEDULER_SERVICE. Call for proposals are then submitted to PRAs to identify separate Node Cumulative Load (NCL) values of shared nodes managed in the identified platforms. Provided the proposals are received, individual PRAs query local databases (node_cpu_util_infos); calculates the NCL of shared nodes in the platforms and returns bids (PROPOSE_AGGREGATE_LOAD_INFO) stating the mean NCLs to the PUA.

node_cpu_util_info	
PK	<u>mac_address</u>
	pc_name ip_address ma_platform ma_architecture os_version comm_virtual_memory_size free_physical_memory_size process_cpu_time system_load_average system_cpu_load

Figure 16: Node utilisation ERD

The requestor PUA then evaluates the bids and forwards an offer to an identified PRA with the least NCL mean (i.e. PRA_2 in Figure 17). Turnaround times in processing workflows can be reduced considerably by assigning a task to a platform with the least NCL. A suitable PRA then provides identities of shared nodes in the related platform on accepting a contract.

The PUA invokes a PMAGUI when node identities are received enabling a grid user to load a PMA, select node of preference and deploy the PMA to evaluate encapsulated problems. When the PMA processes complete on the selected shared node without node recall or interrupt; the PMA migrates back to report on computation results (e.g. transition return (result), Display (results continual)).

If Node_1 in Figure 17 is recalled during application processing for instance, the PMA requests for alternative nodes registered from the AMS. The AMS then checks and returns all nodes identified to the PMA. The PMA then suspends its execution on the current node and moves its application and data to an alternative node (e.g. Node_2). Results are then returned to the user (e.g. transition return (result), Display (results_interrupted)) on successful completion.

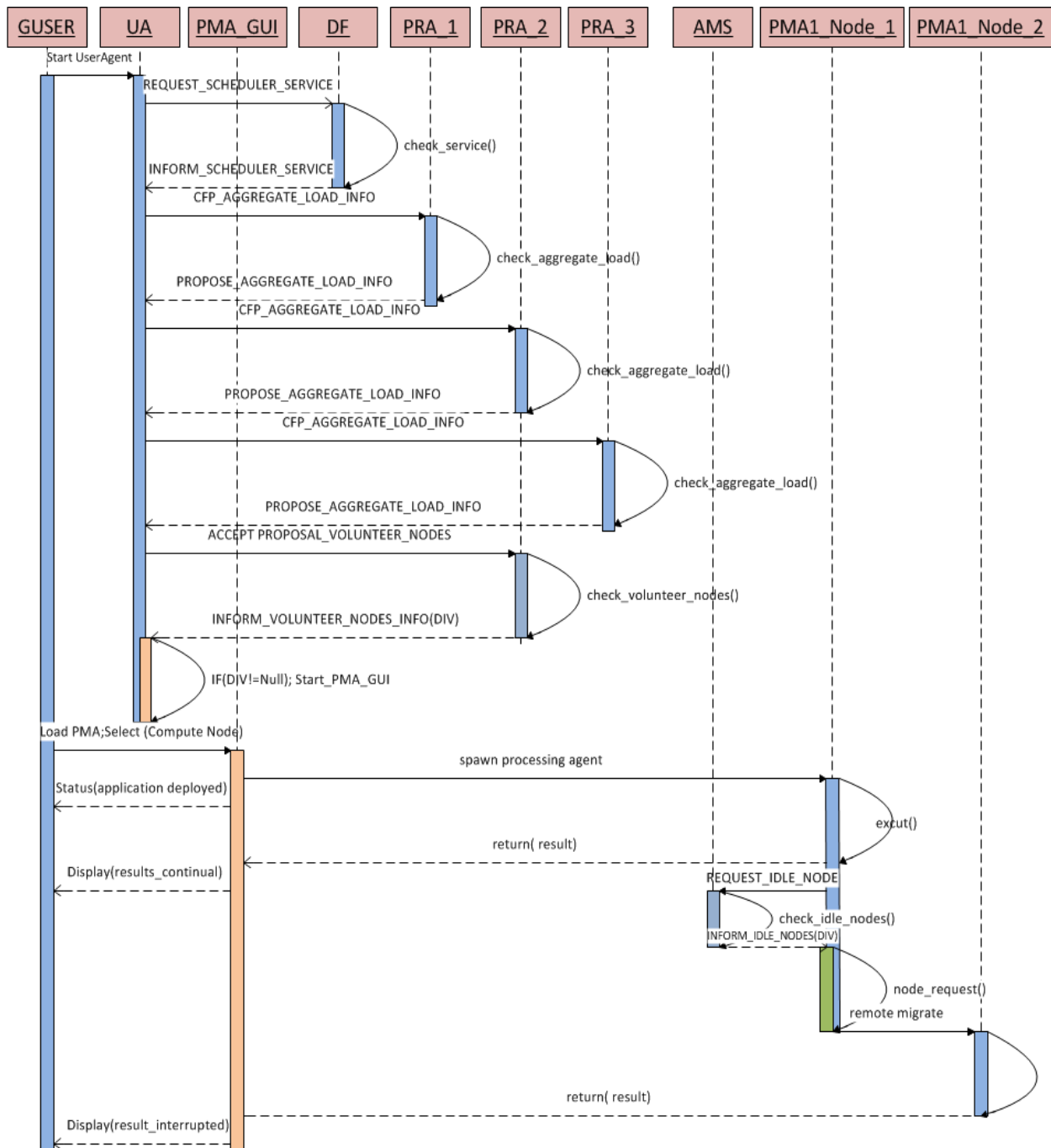


Figure 17: MA Sequence Diagram

6.2 Storage Component

The DM and C-AP file-systems were designed to explore ways of integrating idle storage resources. The terms shared resource and Storage Server (SS) are functionally interchangeable and refer to shared nodes with free disk space.

6.2.1 System and Requirement Analysis

Users request shared disk space from the distributed file hosting service by running specific system modules. The facility should enable these users to:

1. Query the system for shared machines offering free storage space;
2. Select files to upload and forward the content to identified resources;
3. Receive feedback on the storage operations.

Likewise an allowance for a user to retrieve the uploaded file without knowledge of where the file is stored should be incorporated. In view of users and resource providers involved in the realisation of the storage component; the following generic requirements are considered:

1. There should be simplicity in allowing resource providers to share their resources towards the project;
2. Potential users should have seamless access to the storage system.

Participants with a machine and an active connection can interact with the system either as volunteers or users based on the system modules they execute. Established on the system requirements highlighted, a primary list of potential system interaction scenarios was laid out. The modelled use case diagram is shown in Figure 18.

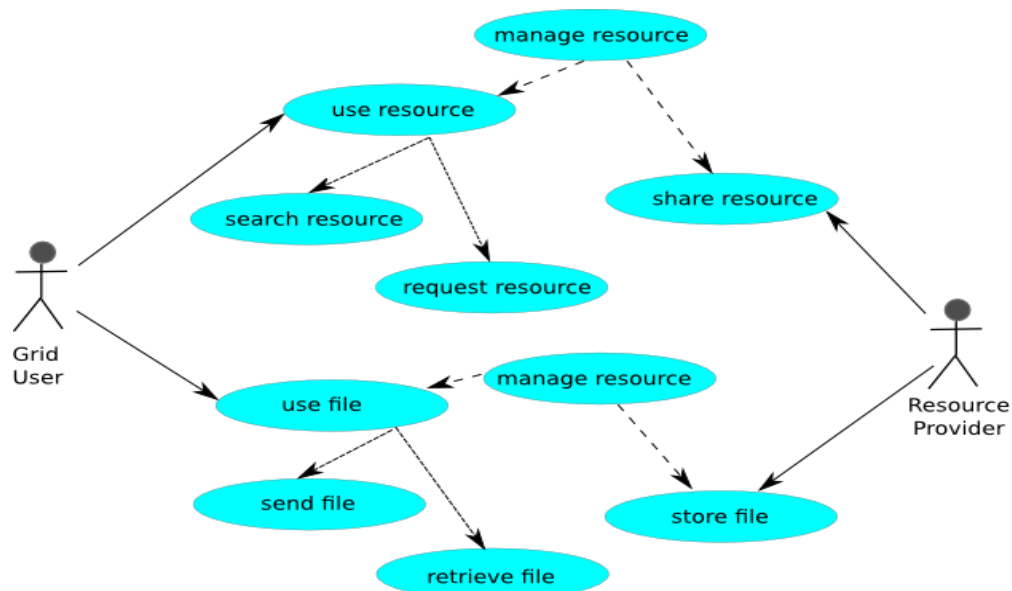


Figure 18: Storage Use Case

6.2.2 Agent Identification

This section identifies agents in the operation scenario as identified by the functional requirements. In the agent identification process, the agent diagram is the key deliverable and unlike in use cases the approach differentiates between human and external system components. A typical agent diagram consists of [22]:

1. Agent Types: Represent the actual agents as circles;
2. Humans: Represent people who interact with the system. They are identified by an actor symbol;
3. Resources: Represent external components which contribute towards a MAS under development (represented as rectangles);
4. Acquaintances: Symbolise association between linked system components. Links in an agent diagram are restricted to agents and resources/humans. Agents to agent relationships are considered further in the design process.

Figure 19 is an agent diagram approach to system analysis.

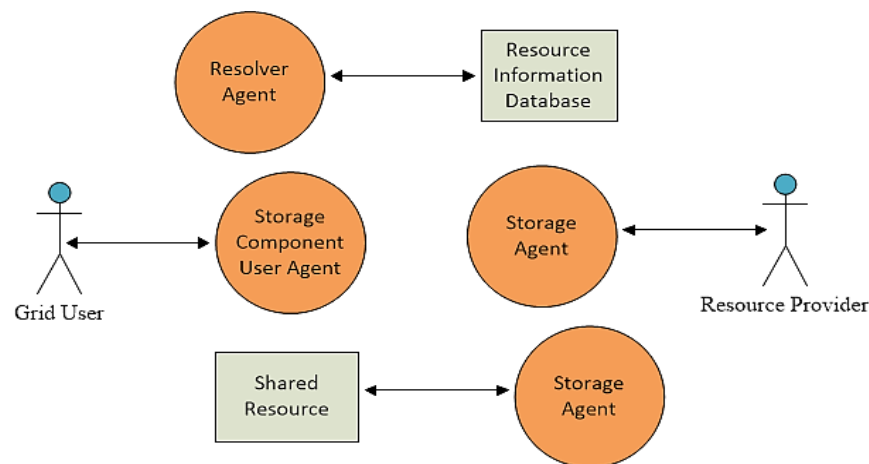


Figure 19: Agent Diagram

6.2.3 Agent Tasks

The storage architecture applicable to the DM and C-AP file-systems is shown in Figure 20. The capabilities of agents in the architecture are explained first in order to appreciate the platform dynamics.

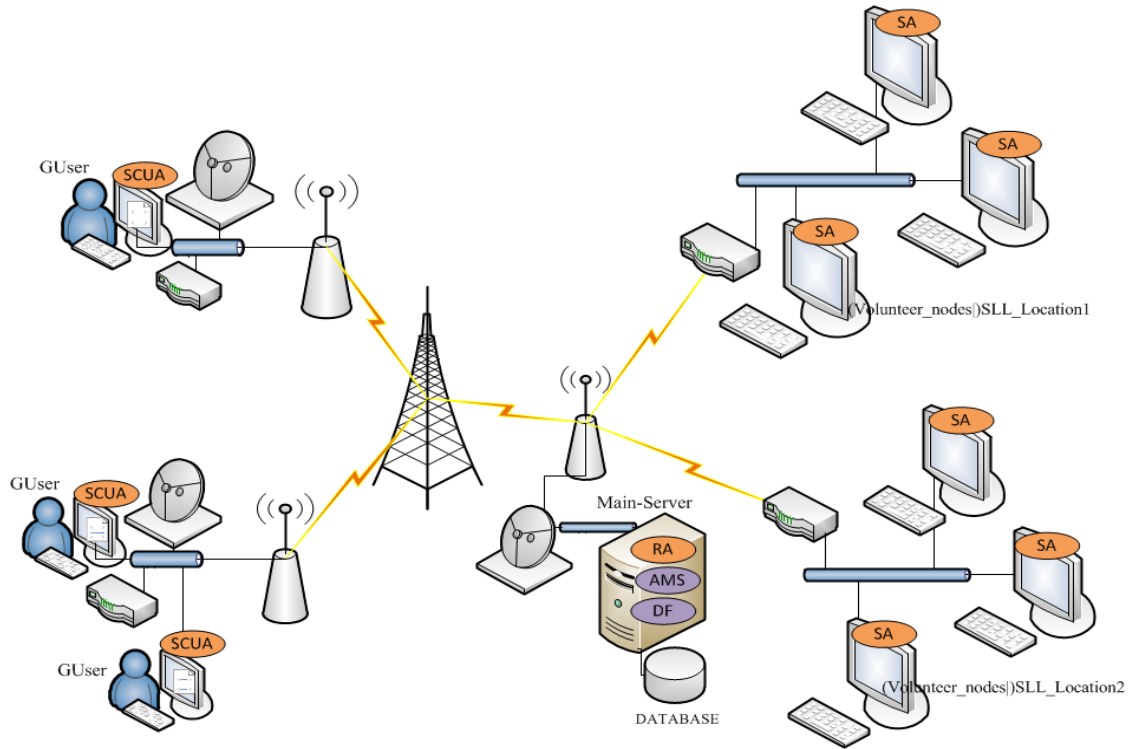


Figure 20: Storage Component Architecture

There are four agent types with different responsibilities in the storage infrastructure exclusive of platform service roles: Storage Component User Agent (SCUA); Resolver Agent (RA); Database Agent (DA)/database and Storage Agent (SA):

1. **SCUA:** SCUA acts as a gateway for users to request storage services from the file-systems;
2. **RA:** The agent handles storage requests and identifies suitable SSs;
3. **SysDA:** The system database stores the SS system information on the underlying disk utilisation. The data types stored are: PC name; Internet Protocol (IP) address; media access control address (MAC-Address); platform; architecture; operating system version; total swap space; free swap space; total disk space; free disk space; and usable disk space;
4. **LDA:** The Locations Database stores parameters to uploaded files;
5. **SA:** SAs identify SSs to a storage platform. The agent is executed by resource providers to donate their machines. In particular, the SA capabilities are summed up as follows:
 - a. The agent is responsible for uploading/downloading a file to/from SS;

- b. The agent collects and stores the SS hardware stack information to a centralised SysDA;
- c. SA publishes the shared nodes capability to DF enabling for SS service identification.

6.2.4 Storage Modules

This section discusses the storage designs and their communication models to achieve the emphasized goals. The main events discussed are focused on the DM and C-AP file system approaches to file storage introduced. The following were defined as pre-conditions:

1. Resource providers register their SAs to an active session;
2. Each SA's AID and service names are defined as "*storageagent*" + *MAC address*. The Media Access Control address (MAC address) is unique for each shared computer. Associating the agent name and its DF registered service with the address, eliminates conflict in service and agent name naming conventions; supports unique identification of specific SA to SS relationships; and enable simplicity in developing agent functionality;
3. SAs register and update their SS information with a SysDA after a defined time interval;
4. The RA (broker service) is registered with the DF;
5. A user's SCUA can identify the broker service in platform through the DF.

6.2.5 DM File-System

6.2.5.1 Upload Service: 1st Iteration

The DM file-system's agent acquaintances and responsibilities are presented in an informal and intuitive way in Table 6.

File Upload Steps (in Figure 21):

1. An initialized SCUA searches for a registered broker service (RA) from the platform's DF. A request for a SS with largest usable disk space from an identified RA is then forwarded;
2. The RA receives the request and queries for a SS with the largest usable disk space from SysDA. On extracting an identified optimum SS MAC address, the RA attaches the

string “*storageagent*” creating a “*storageagent*”+*mac_address* variable type matching the SA agent and service names;

Agent	Acquaintances and Responsibilities
SCUA	a) Initiate user request for a file upload service; b) Let the grid user select a file to upload; i) Convert selected file into acceptable message format; c) Forward the selected file to identified SA for upload; d) Notify user on the status of upload
RA	a) Receive storage request from the SCUA; b) Search for optimum SS from the DA/database; c) Return optimum SS/SA AID to the SCUA.
DF	a) Register SA and RA services.
SysDA	a) Collect SS system information provided by SA.
SA	a) Populate SS system information into SysDA; b) Receive SCUA file upload requests; i) Write received file into SS and return feedback.

Table 6: DM Upload Service Responsibilities

3. As “*storageagent*”+*mac_address* match with a unique SA service; RA uses the variable to determine the availability and identity of a SS specific agent. If the SA is active, RA then returns the SA AID to the SCUA;
4. The SCUA fetches and decodes the feedback. If a SS is available (*e.g. If (SA_AID! =Null)*), a file chooser is triggered enabling users to select a file to upload. The SCUA in turn converts the selected file into a byte data type supported. On conversion, the file is forwarded as a request to an identified SA;
5. The SA receives the request and writes the file to a SA’s defined SS directory. A reply is then returned to the SCUA specifying the service handler identity (SA) and the remote Filename defined;
6. The SCUA then shows the SA AID and remote file name parameters to an upload.

Design Limitations: 1st Iteration

The first iteration file upload service was captured by the sniffer agent as in Figure 22. At A in Figure 22, the SysDA is queried for an optimum SS; B returns the SA AID to handle the upload service; and stage C generalise the file chooser instance, file conversion and file forwarding to a SA.

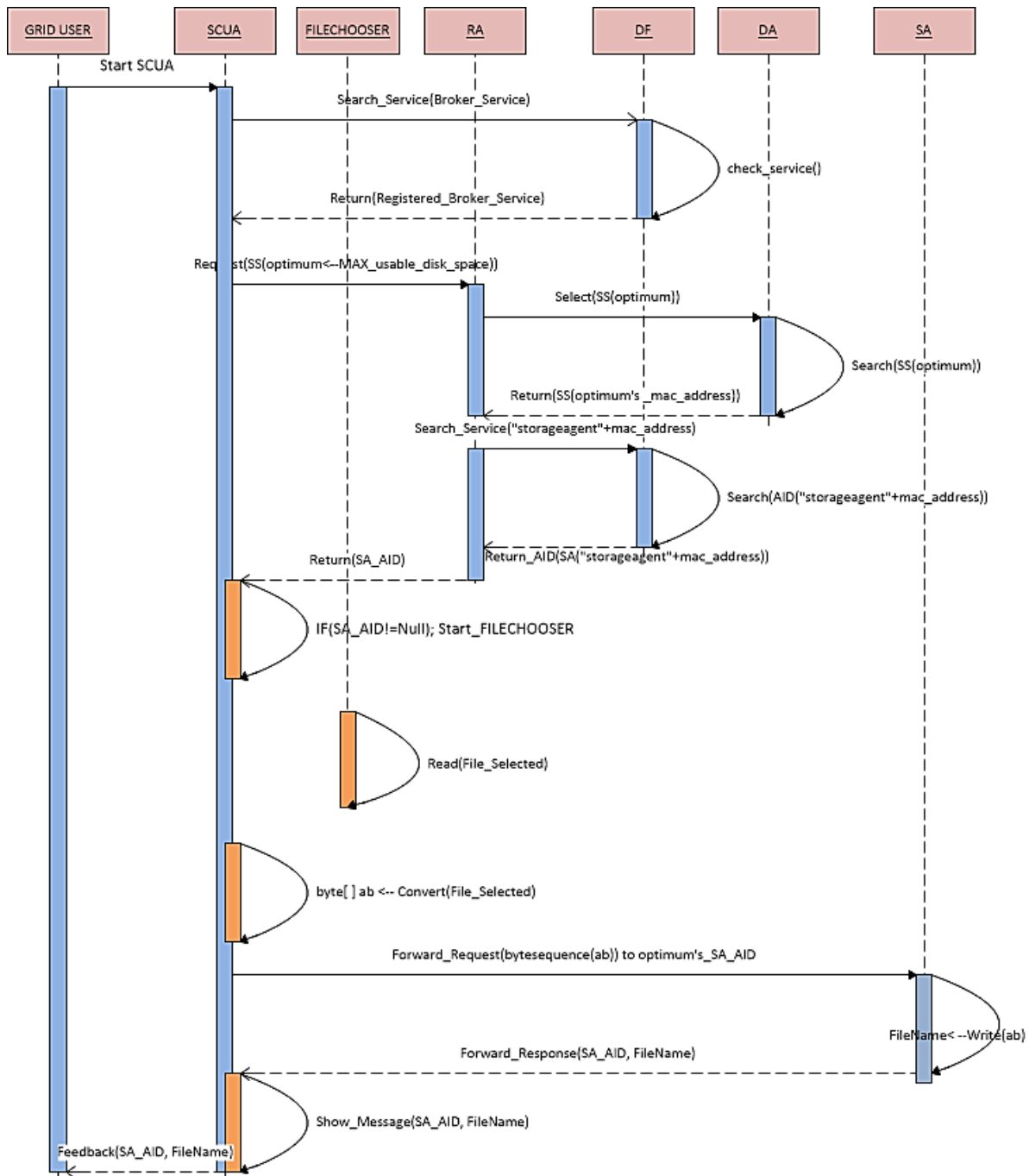


Figure 21: File Upload Service (1st iteration)

The design lacked redundancy in file storage since a single SA AID parameter is returned. Sending an upload to RA for the module to resolve and handle the upload request to the SA was viewed as convenient. This restricts the user machine participation in the service. It was also important to deprive the user from backend technical processes and parameters to make the system more user-friendly. These issues were considered in the second iteration.

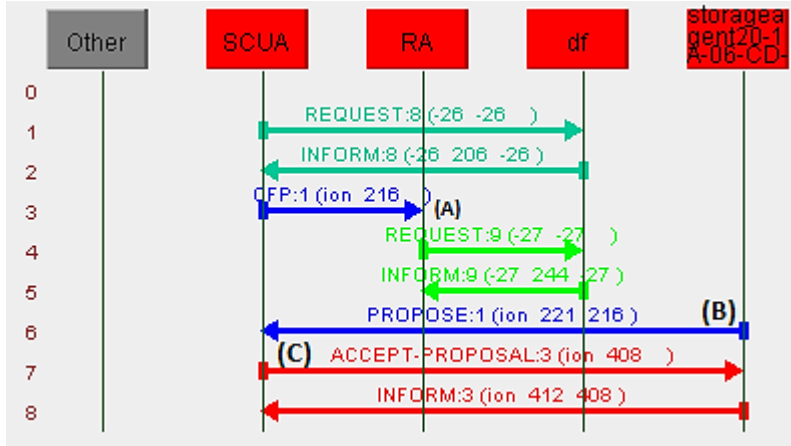


Figure 22: Sniffer Agent interaction capture

6.2.5.2 Upload Service: 2nd Iteration

The second iteration file upload service resolved shortfalls highlighted in the previous section. The updated acquaintance and responsibilities are shown in Table 7. The changes are highlighted in italics.

Agent	Acquaintances and Responsibilities
SCUA	a) Requests upload service on behalf of user. b) Let the grid user select a file to upload; i. Convert selected file into accepted message format; c) <i>Forward file to RA;</i> d) <i>Receive RA upload status and show upload feedback.</i>
RA	a) <i>Receive SCUA uploaded file;</i> b) Search for optimum SSs from SysDA; c) <i>Forward file to identified SSs/SAs for storage;</i> d) <i>Save file upload parameters in a database;</i> e) <i>Return upload status to SCUA;</i>
DF	a) Register SA and RA services;
SysDA	a) Store SS system information provided by SAs;
LDA	a) <i>Store file SS storage parameters;</i>
SA	a) Populate SS system information in SysDA; b) <i>Handle RA upload requests;</i> i. Write file in SS directory c) <i>Return uploads parameters to RA.</i>

Table 7: Updated Agent Roles

To offer redundancy in file uploads, specific task delegation steps were reassigned.

The following assumptions were also redefined:

1. Indirect interaction between SCUA and SA through the RA;
2. The SCUA's AID parameter is preserved in the RA to SA upload round-trip interactions to uniquely identify the sender and file upload locations in LDA;

File Upload Steps:

1. When a grid user initiates a SCUA, the agent invokes a file chooser to select a file. The SCUA then converts the selected file into supported content. To query for storage resources, SCUA searches the DF for broker services registered. From an identified RA, the SCUA requests (N) SS with the largest usable disk space available from the identified RA (N is the number of SS tagged as optimum). The SCUA AID and the file payload are also specified in the request;
2. The RA receives the request from the SCUA and determines if the query matches its primary capabilities. If conditions are satisfied, RA then retrieves the payload and queries (e.g. Select (N*optimum_mac_addrs)) the SysDA for MAC addresses affiliated with (N) optimum resources. Provided the $N * SAs$ are online, RA encodes the file payload and sends the requests to these SAs;
3. Typical RA requests are handled by SAs in the following way:
 - a. The request is decoded to identify the file payload;
 - b. The payload is extracted, validated and saved under a unique alias (Filename) in the SS directory;
 - c. Each SA attaches the SS MAC address and saved file (Filename) identifier; and returns the parameters to RA.
4. The RA receives the (N) SA parameters and saves the data in LDA uniquely identifiable by the source SCUA AID. The file upload status is then returned to SCUA.

Design Limitations: 2nd Iteration

Selecting SSs based on largest usable disk space available in the design didn't fully optimise the storage capabilities offered. Considering a First Come First Served (FCFS) scheduling scenario in which small file uploads are prioritised on SS with largest usable disk space compared to large file sizes. The assumption results in specific SS being underutilised and over-utilized based on the disk space available. Also, the sequence flow didn't keep track of the user's source file

name. Keeping track of the parameter enables the platform to reconstruct the exact file name and its contents on download request. These recommendations were incorporated in the next iteration.

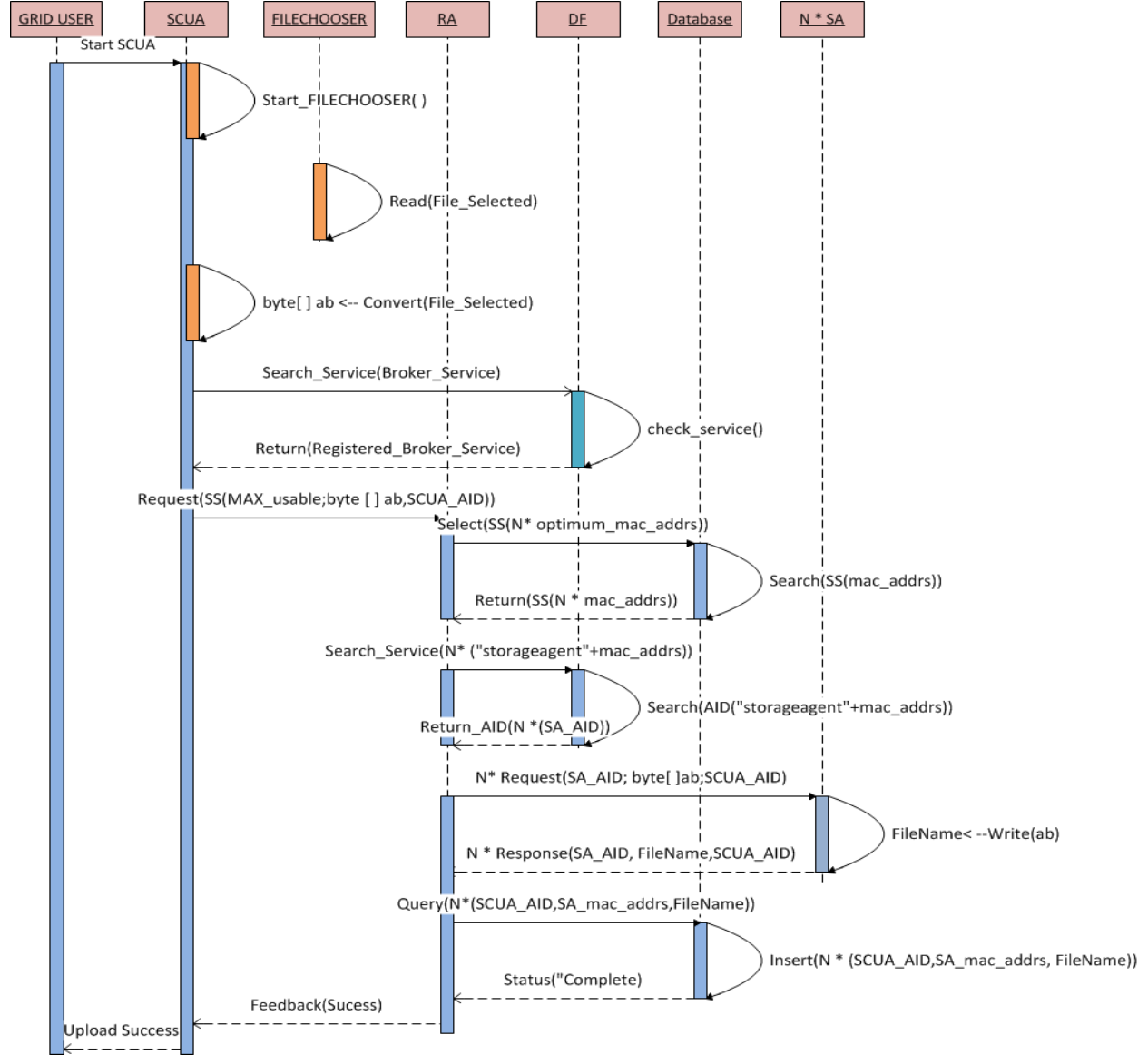


Figure 23: File Upload Service (2nd Iteration)

6.2.5.3 Upload Service: 3rd Iteration

The third iteration merged a resource mapping function to identify SSs suitable for an upload contrary to the approach where SSs with largest available storage are selected. The smaller the file upload size the less free usable disk space required was the defined mapping criteria. The inverse was assumed for a larger file size. The SCUA's AID and an additional source filename

parameter were included in RA to SA upload round-trip interaction. The acquaintance and responsibility table defined in the previous iteration was maintained. The updated sequence diagram and steps which address the highlighted shortfalls are shown in Figure 24.

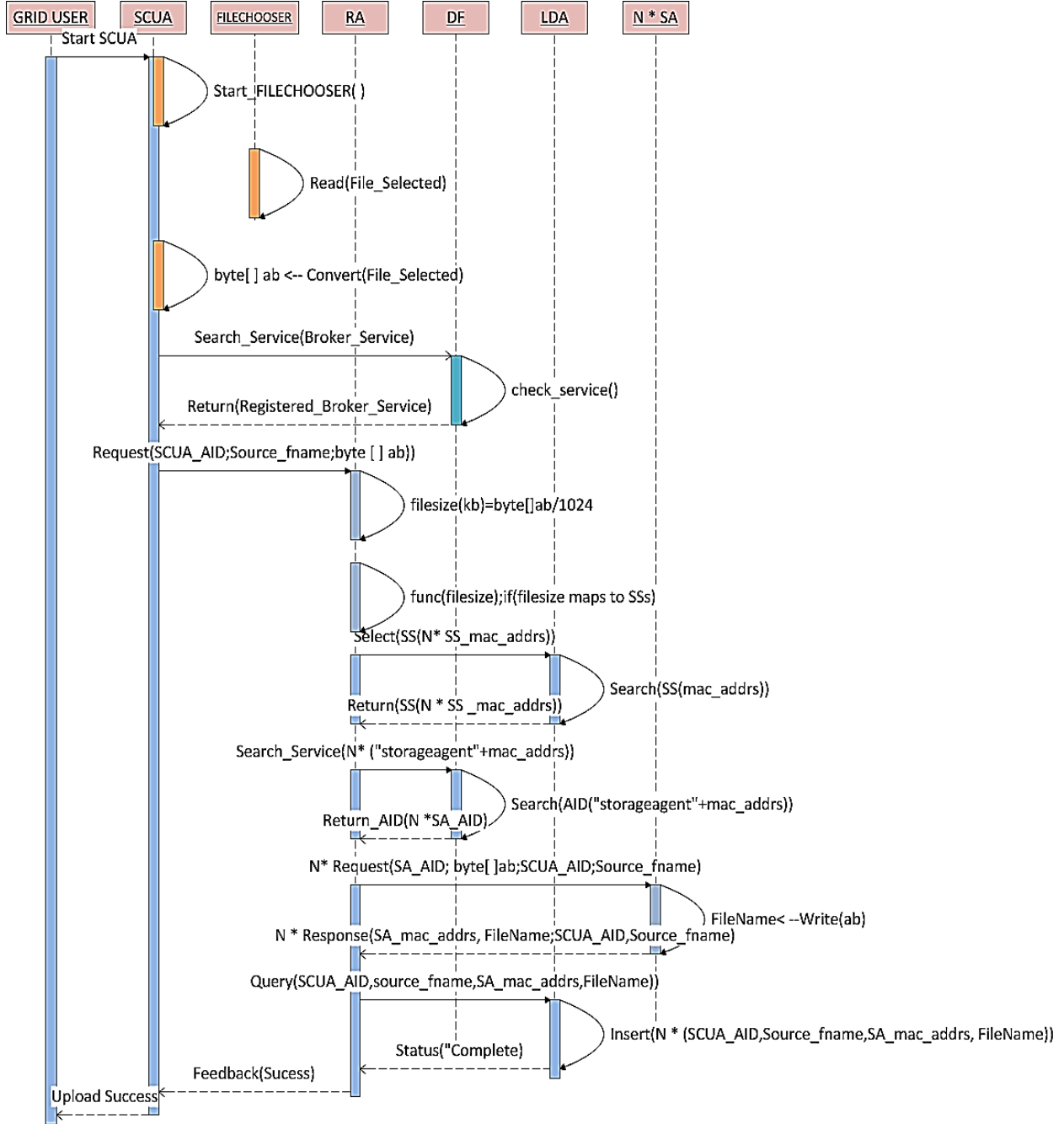


Figure 24: File Upload service (3rd Iteration)

File Upload Steps:

1. A grid user selects a file to upload through an invoked file chooser. SCUA then sends the payload, its AID and an additional source filename parameter to RA;
2. RA acknowledges the SCUA request, retrieves the payload and calculates the file size. The file size variable is parsed into a function which identifies suitable SSs from SysDA. The identified SS MAC addresses are appended to a string as in the first iteration; RA then sends the file payload, SCUA AID and an added source filename to SAs identified for storage;
3. The SA processes steps 3a and 3b in Section 6.2.5.2 and returns file storage parameters to RA. The parameters are then saved in a LDA with the SCUA AID as the primary key;
4. An upload status is then returned to the requestor.

6.2.5.4 Download Service

This service downloads a file uploaded in Section 6.2.5.3. The events are shown in Figure 25. A file can be read from any of the redundant node/SS identified in the upload process.

Agent	Acquaintances and Responsibilities
SCUA	a) Request file on grid user's behalf; b) Forward SCUA AID to RA c) Receive the file from RA and write the file locally d) Show download status and local file path
RA	a) Receive locations identifier to the redundantly stored file b) Check for locations where the file is redundantly stored c) Request file from identified SAs/SSs d) Return file to SCUA
DF	e) Register RA and SA services
LDA	f) Provide locations to stored files and parameters;
SA	a) Handle RA file download requests i. Retrieve and return file to RA.

Table 8: File Download responsibilities

A download is handled by SCUA, LDA, DF and SA agents in the following way:

1. On user SCUA execution, the agent sends its AID to RA. The RA extracts the SCUA AID and queries LDA to identify a record uniquely identified by the parameter. If the record exists the RA retrieves the source filename, the different SS assumed Filenames and SS MAC addresses associated with the file;

2. The RA then identifies the SA AIDs from the DF using MAC addresses extracted. If the SAs/SSs are online the RA sends requests for file retrieval to the respective SAs;
3. The SAs retrieve the file name specified in the request and return the file payload to the RA. Provided the (N) SAs feedbacks are received, the RA returns a single copy of the file to the SCUA after checking the consistency of feedbacks returned;
4. At the SCUA, the agent extracts the source filename parameter; creates the original file container and writes the file contents;
5. A download status including the path of the downloaded file is then presented to the user.

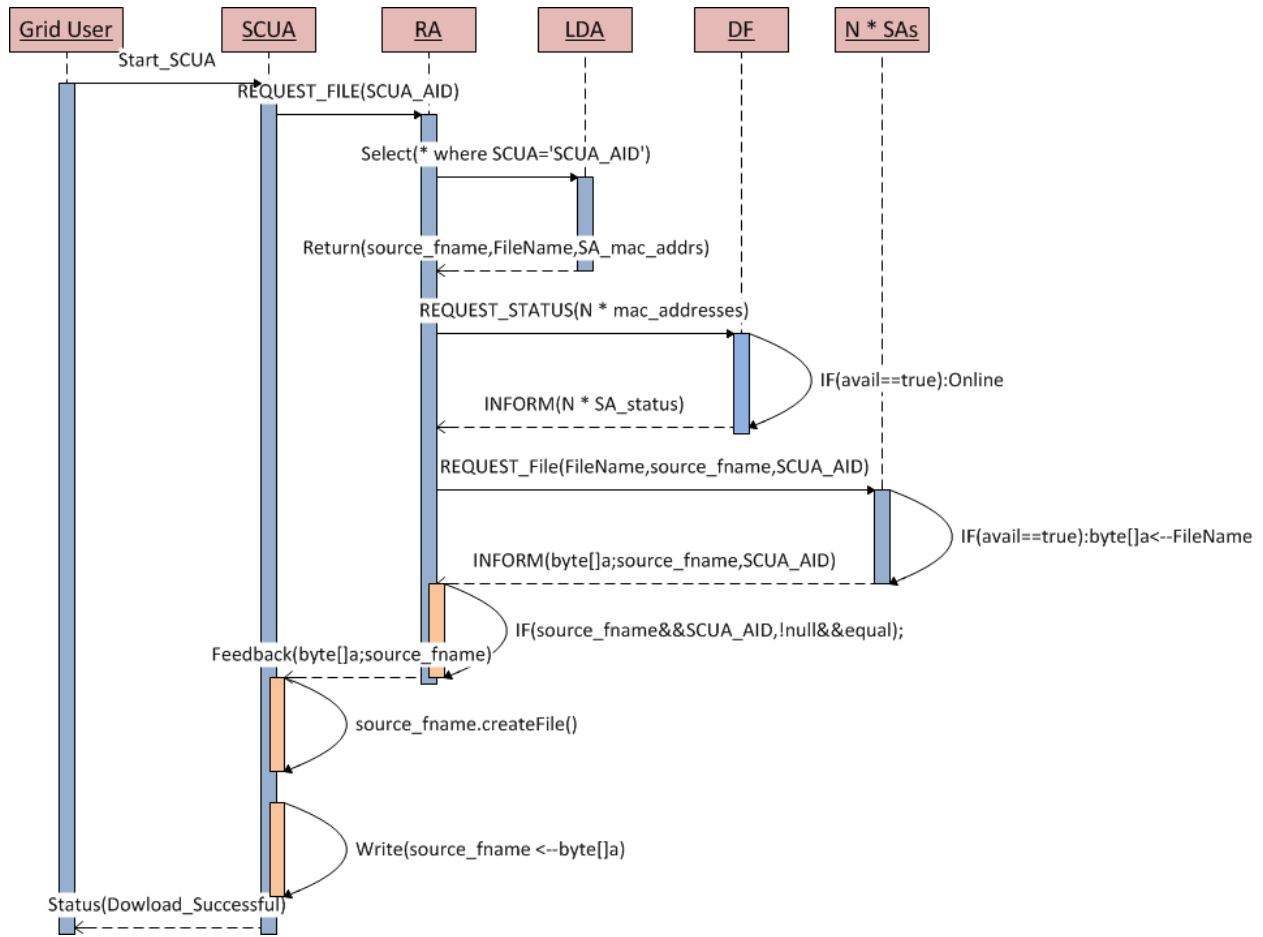


Figure 25: File Download Service

6.2.6 C-AP File-System

The C-AP file-system implements GFS's performance optimizations presented by file striping and atomic appends. This section details the file-system's storage service which achieves specific objectives.

6.2.6.1 Dynamic vs. Static Chunking

The C-AP file-system identified dynamic and static chunking methods as techniques that can be used in the design of the storage service. This research defines the phrase “dynamic chunking” as the ability of a storage service to devise a file striping coefficient based on the upload byte size. Static striping on the other hand assumes a fixed chunk size for any upload. Utilising dynamic chunking; isn’t lossless in byte conversions as was observed in design and code considerations. Considering an upload with a byte length not divisible by a coefficient deduced; this resulted in a file fragment being discarded and not accounted for. Due to ease of integration and GFS performance optimisations the static chunking approach was adopted in the design and implementation phases.

6.2.6.2 C-AP Upload Service

Agent	Acquaintances and Responsibilities
SCUA	a) Requests upload service on behalf of grid user i. Let the grid user select a file to upload b) Convert selected file into accepted message format c) Forward file to RA d) Receive RA upload status and show upload feedback
RA	a) Receive SCUA uploaded file i. <i>Split file into chunks</i> b) <i>Search for available SSs from SysDA</i> c) <i>Forward chunked files to identified SSs/SAs for appends</i> d) <i>Store SAs returned chunk locations and parameters in LDA</i> e) Return upload status to SCUA
DF	a) Register SA and RA services;
SysDA	a) Store SS system information provided by SAs;
LDA	a) <i>Store parameters to chunk locations</i>
SA	a) Save SS system information in SysDA; b) Handle RA upload requests; i. <i>Append chunks to SS local file</i> ii. <i>Compute the chunk byte offset</i> c) Returns upload parameters to RA.

Table 9: C-AP Responsibilities

The C-AP file-system has agent roles which support C-AP in handling uploads. The SCUA responsibility in DM and C-AP file-systems remain consistent. Changes (shown in italics) to the DM file-system’s agent responsibilities to realise introduced functionality are shown in Table 9. The sequences for the upload service are shown in Figure 26.

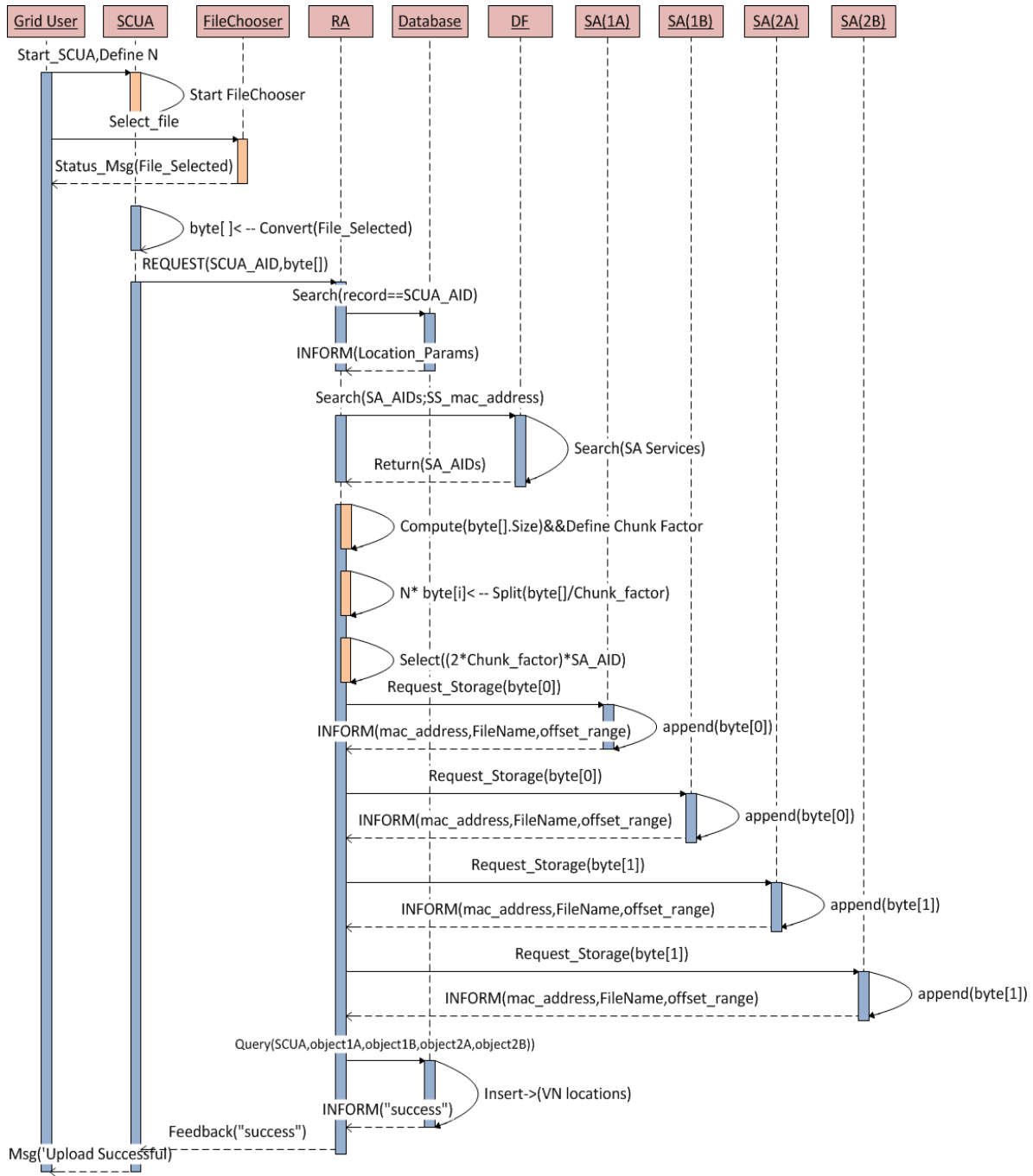


Figure 26: File Download Service

File Upload Steps:

1. The SCUA sends a request to the RA; the message contains the user selected file and its parameters;

2. The RA then computes the size of the file; determines the chunking factor and chunks the file. When this is done, the broker service identifies available SSs from the database and resolves their representative SA AIDs from extracted SS MAC address;
3. To identified SAs based on a mapping criteria, the RA pushes the chunk replicas at a specific replication level;
4. A typical SA receives the chunk, appends it to an existing file, computes the append byte offset and returns the status to the RA. The status includes the SS filename which contains the chunk, the append byte offset and the SS's MAC address;
5. Provided the RA receives all chunk append feedbacks; the parameters are saved in a database uniquely identifiable by the requestor SCUA AID;
6. The Upload status is then returned to the SCUA via the RA.

6.2.6.3 C-AP Download Service

To download a test file uploaded using the approach in Section 6.2.6.2, the message passing model which retrieves and reconstruct the user file is discussed here. The agent acquaintance roles which vary from the DM file-system's download service are shown in Table 10.

Agent	Acquaintances and Responsibilities
SCUA	<ol style="list-style-type: none"> a) Request file on grid user behalf b) Forward SCUA AID to RA c) Receive the file from RA and write the file locally d) Show download status and local file path
RA	<ol style="list-style-type: none"> a) <i>Receive chunk locations identifier from SCUA</i> b) <i>Check locations where chunks are stored from database</i> c) Request file from identified SAs/SSs d) <i>Join SA returned chunks and return file to SCUA</i>
DF	<ol style="list-style-type: none"> a) Register RA and SA services
LDA	<ol style="list-style-type: none"> a) Provide locations to stored files and the parameters
SA	<ol style="list-style-type: none"> a) Handle RA file download requests <ol style="list-style-type: none"> i. <i>Extract the chunk byte range specified by offset</i> ii. <i>Return chunk to the RA</i>

Table 10: C-AP responsibilities

File Download Steps:

1. A SCUA sends a download request to RA. The SCUA AID is encapsulated as a request parameter;

2. The RA receives the request and searches for a chunk location database record which matches the requestor's AID. When located the MAC addresses of SS with chunks are extracted and passed to the DF to identify SA AIDs. The RA then sends requests to these SAs simultaneously to request for chunk retrievals. The byte offsets and destination file names are specified as crucial parameters;
3. The SA accepts the request, accesses the SS file path and copies the chunk byte range specified by the offset; the chunk is then returned to the RA;
4. Provided all chunk retrieval feedbacks are returned successfully; the RA appends the chunks in sequence and returns the reconstructed file to the requestor (SCUA);
5. The SCUA writes the file locally and provides an upload status to the user.

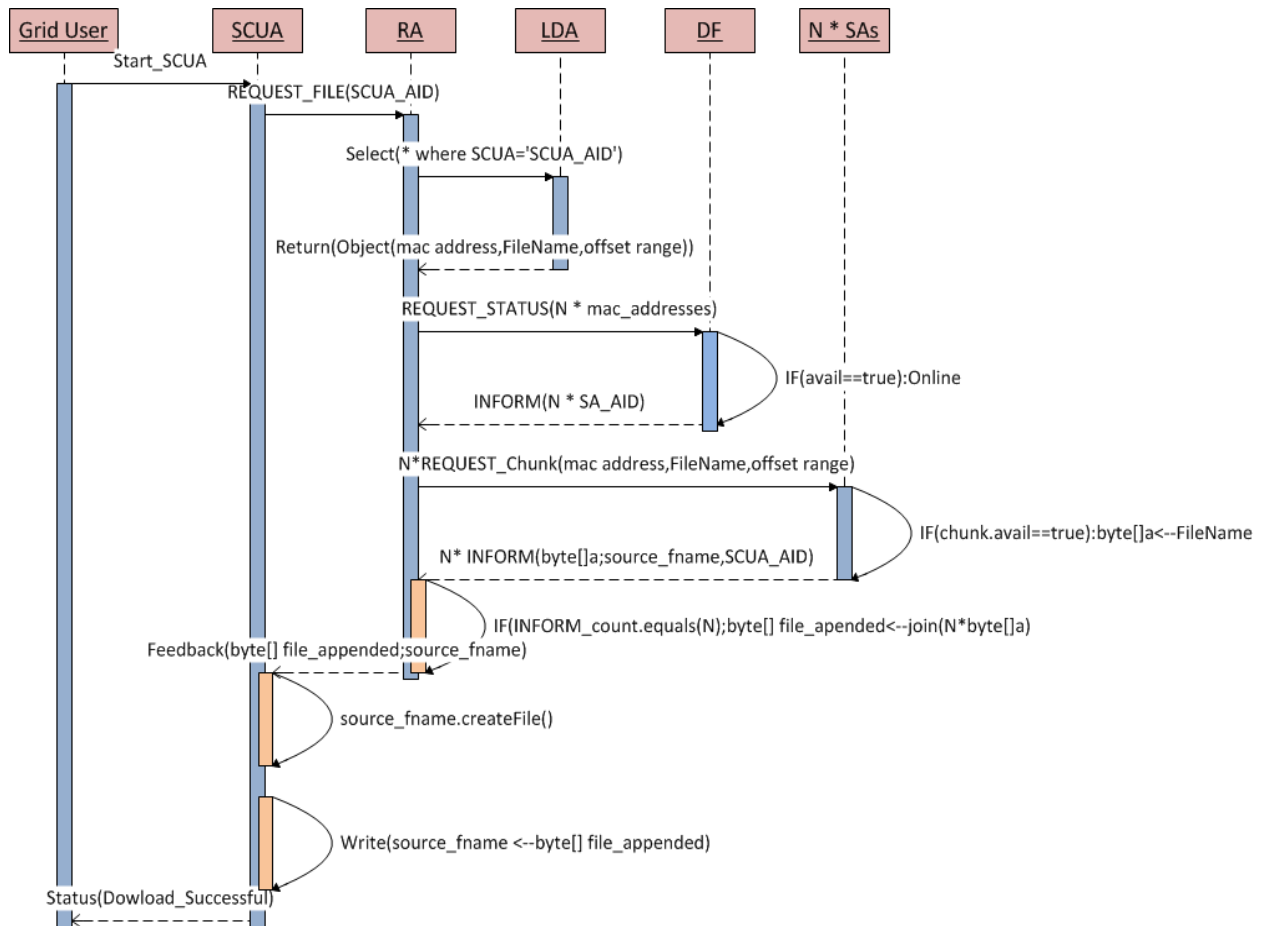


Figure 27: File Download Service

6.3 Conclusion

This chapter justified the compute and storage methods proposed at a high level. The architectures and acquaintance interactions show key patterns which highlight anticipated functionality. The CNP and rule reasoning functionalities explained incorporate concepts viable for dynamic resource identification and allocation in heterogeneous settings. The service methods were analysed and explained in MAS contexts to aid transition into the implementation stage.

Chapter 7

7 MAOP Services

The previous chapter identified key platform roles and relationships in the MAOG system. Proof-of-concept prototypes which validate the technical sufficiency of designed services are presented in this chapter. The designed components demonstrate the viability of agent technologies in Opportunistic Grid Computing. The software agent codes which implement the required functionality are presented in the following sections.

7.1 MA Component

The MA compute integrates shared processor resources into a low-cost commodity system. The CNP and Jess rule engine implements resource identification, negotiation and decision making in the distributed MAS to prioritise resource provider activity. The component assumes the converged architecture introduced in Section 6.1.1.3. The MA project and its agent participants is shown in Figure 28.

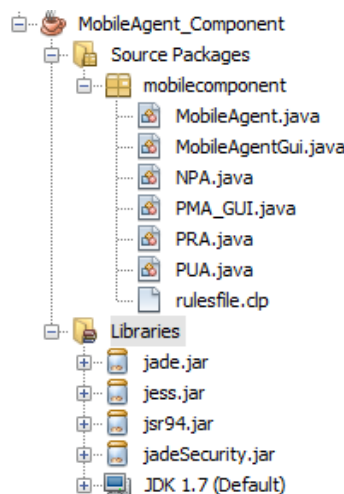


Figure 28: MA Compute Project

7.1.1 Processing User Agent-PUA

The PUA class extends a GUI agent. The agent requests and negotiates for resources to process encapsulated functionality. The following are steps involved in this process (see Appendix A.1 for associated code snippets):

1. A PUA identifies PRAs registered with a proxy-server to identify a platform with shared nodes least utilised; the PUA then sends call for proposals to identified PRAs (PRA_AIDs[i]) to request platform node utilisation information (in Appendix A.1.3).
2. The PRAs respond with bids detailing the system load averages of nodes in the respective platforms. Provided all bids are returned, PUA selects a bid with the least mean system load average and extracts the source PRA AID (in Appendix A.1.4). A contract is then allocated to an ideal PRA by sending an offer requesting the node identities in the affiliated platform (see Appendix A.1.5 for code extract);
3. If a destination PRA accepts the offer and returns the shared node identities; a PMAGUI (in Figure 29) initialises. The GUI prompts a user to load a PMA and sends the agent to process in the preferred idle node.

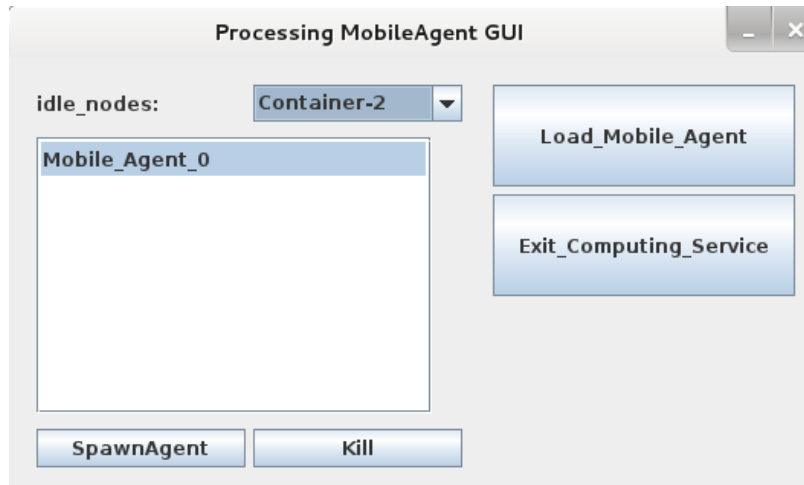


Figure 29: PMA GUI

7.1.2 Node Processing Agent-NPA

The NPA is a node system daemon. Resource providers run the component to register node's CPU utilisation parameters (in Appendix A.1.8) with a centralised database (in Appendix A.1.7). The parameters are registered (in Appendix A.1.9) for the first time on NPA start-up. To constantly update the processor utilisation metrics, a TickerBehaviour executes after a defined interval.

The most important metric is the system load average provided by the platform MXBean which is an MBean for managing and monitoring Java virtual machines. The system load average for a

node computes the sum of runnable entities queued to available processors and the number of runnable entities using the available processors averaged over a specific period of time. The system load average takes values in the 0.0 to 1.0 range. The value (0.600) was defined as the maximum value a Linux node is considered idle from experiments in view of the native Linux threads requirements.

7.1.3 Processing Resolver Agent-PRA

JADE runtime instances provide an environment for PMA to execute when deployed. A PRA identify these containers as a group of nodes. On start-up the PRA registers with a proxy-server (in Appendix A.1.11). If a container is added or removed from the platform; the PRA AMSSubscriber class listens on these events (in Appendix A.1.12) and maintains an updated list of nodes in availableContainers.

In response to PUA call for proposals, a PRA calculates the mean system load average for active nodes in a platform (in Appendix A.1.13). If the PRA is offered a contract based on a submitted bid (in Appendix A.1.14); the agent returns availableContainers as an acknowledgement for an offer (Appendix A.1.15).

7.1.4 Mobile Agent-MA

The MA integrates a reasoning component and user application. The design was driven by concepts in exploring mobility [126]. Support for mobility is achieved through APIs and methods which allow agents to decide on actions to perform independently [126]. The methods offered by JADE to manage code relocations include [126]:

- doMove: The method is called to move an agent and takes the destination as a parameter (i.e. doMove (location);
- beforeMove: The codes in method process before an agent moves to a defined location;
- AfterMove: The method is initialised on reaching the remote node.

The doClone, beforeClone and afterClone are specific for agent cloning. The MA code fragments that implement the expected functionality are introduced in the next section.

7.1.4.1 Processing Mobile Agent

The PMA class extends a GUI agent. When node identities are returned by a PRA, a user loads a PMA from the PMA GUI as in Figure 29. The void setup method (in Appendix A.1.16) executes on loading a PMA. Registration of the ontology and language is handled by the init method (in Appendix A.1.18).

The Jess rulesfile.clp file is attached as a fileinputstream to relocate with the PMA. The file contains a rule the PMA loads into a rule engine's working memory to reason about the execution environment. If a benchmark system load average load is exceeded the rule is fired and the rule component embedded recommends the PMA to relocate to an alternative idle node.

The container_array list (in Figure 30) keeps track of nodes traversed during PMA processing. The report_results_sourcenode captures the sender's JADE container for the PMA to migrate back and report on compute results. The Main-Container is added initially to restrict the agent from relocating to the main bootstrap point for processing.

```
//add sender and main container identities
container_array.add(report_results_sourcenode);
container_array.add("Main-Container");
```

Figure 30: Keeping track of source node

If a PMA is loaded, the PMA GUI SpawnAgent tab sets out the module to a selected container/shared node. A user can alternatively kill the loaded PMA using the kill button. The actions are implemented by the PMA GUI's doMove and doDelete methods (in Appendix A.1.17).

When a PMA is deployed, the afterMove method (in Appendix A.1.19) is called at the destination container/node. The method initialises the Jess rule engine instance (in Appendix A.1.21) after a defined interval and also includes code that checks if the recent migrated node is the source computer in which case the PMA prints the computation results. A Jess rule file (finalfile.clp) is written on the destination node to enable the PMA to reason about node CPU utilisation patterns.

The defined rule which reasons on the execution environment is shown further on:

```
(deftemplate rulereasoning (slot cpuinfo))
(import java.lang.Double)
(defglobal ?*var* = 0)
(defrule rule-reasoning
  (rulereasoning {cpuinfo > 0.600000})
  =>
    (bind ?*var* "migrate")
    (halt))
```

The Simpson Rule class (in Appendix A.1.20) was used to test the processing capabilities of PMA whilst reasoning on the execution environment. The application is initialised by compute_application.main method. If a destination container to which an agent migrates match the container_array.get(0) (sender container identity), the PMA reports on the compute results.

Provided a PMA rule reasoning component detects resource provider activity on the shared node during processing; the agent's rule engine fires a rule (defined in finalfile.clp) based on the fact and instructs the PMA to migrate. If the system load average value is greater than 0.600000 (motivated in Section 7.1.2), the rule is fired and the Jess engine issues a “migrate” action to the PMA. A sample PMA migration prompt on detecting user activity on the node is shown further on:

```
==> f-o (MAIN::rulereasoning (cpuinfo <Java-Object: java.lang.Double>))
==> Activation: MAIN:: rule-reasoning: f-o
FIRE 1 MAIN:: rule-reasoning f-o
Action = "migrate"
```

When the “migrate” directive is issued by the reasoning component, PMA requests for alternative nodes from the AMS (in Appendix A.1.22). The containers returned are read into Loc. Loc container identities are of the form “Container-1@kalibacktrack-Raymond”. The doMove method accepts string identities of the nature “Container-1”. A substring is introduced to trim the AMS returned variables to suite the doMove method parameter specification. The new container identifiers are then pushed into AMS_containers (in Appendix A.1.23).

The agent then selects container identities not traversed previously and selects a random node to relocate to. If processing is uninterrupted, the numerical combinations (in Appendix A.1.20) are processed until the agent migrates back to report on results (in Appendix A.1.24).

7.1.5 Integration Exceptions

Traditionally business logic (i.e. rules) in expert systems was implemented directly in application code. A number of applications still have rules tightly coupled with applications. If rules change, it's required to modify all affected parts. Inference engines have since changed the way business logic is implemented to solve problems associated with tightly coupled applications.

Pattern matching of rules in inference engines is mainly non-deterministic. An effort to parallelise the Rete algorithm in firing and/or matching stages is an area of active research. Parallel firing of rules results in deterministic execution which has some limitations. Drools rule engine has since revolutionised from 5.x to the recent 6.x series revisions which include an enhanced version of the Rete algorithm (ReteOO). ReteOO supports concurrent and parallel matching strategies.

In the following sub-sections, the research's experience in integrating a PMA with Drools and Jess is shared. The most adopted approach for rule based reasoning in JADE agents is Jess [127]. Due to the proliferation of Drools enabled applications motivated by its open source nature; this research assessed the innovation Drools can offer when included in a PMA and possibly propose it as a viable alternative for Jess.

7.1.5.1 JADE and Drools: Exceptions

A simple Drools instance which asserts a fact ("test") into the working memory was utilised. This was a preliminary test before rules and facts that match the PMA compute problem were developed. The main idea was to match the test fact with a Test Drools Reasoning rule, returning "Reason, Drools Reasoning Working!" when a rule is fired.


```

rule "Test Drools Reasoning"
    when
        message: Message (type=="test")
    then
        System.out.println      ("Reason,      Drools      Reasoning
Working!");
    End

```

Drools 5.3.0 packages were used to compile a PMA at the source node and setting up a Drools runtime instance in a remote JADE container. The remote container was initialised as in Figure 31.

```

root@msclab-Aspire-E1-572:/home/msc-lab/Desktop# java -cp http.jar:ilop.jar:jade
.jar:jadeTools.jar:knowledge-api-5.3.0.Final.jar:mvel2-2.1.0.drools4.jar:drools-
core-5.3.0.Final.jar:drools-compiler-5.3.0.Final.jar:core-3.4.2.v_883_R34x.jar:a
ntlr-runtime-3.3.jar:antlr-3.3.jar jade.Boot -container -host 172.20.56.49 -port
1099
Mar 19, 2015 9:33:33 AM jade.core.Runtime beginContainer
INFO: -----
This is JADE 3.6 - revision 6032 of 2008/05/05 14:07:10
downloaded in Open Source, under LGPL restrictions,
at http://jade.tilab.com/
-----

```

Figure 31: Initialising Drools in JADE

```

String DRL_FILE = "/mobilecomponent/testrules.drl";
//---Adding a fact to the working memory-----
void initiliseMessageObject() {
    Message msg = new Message();
    msg.setType("test");
    sessionObject = rbase.newStatefulSession();
    sessionObject.insert(msg);
}

void initialiseDrools() {
    //1. Read the DRL File and add to package builder
    try {
        pbuilder = new PackageBuilder();
        Reader reader = new InputStreamReader(HelloDrools.class.getResourceAsStream(DRL_FILE));
        pbuilder.addPackageFromDrl(reader);
    } catch (Exception ex) {
        System.out.println(ex.getMessage());
    }

    //2. Check for any errors
    PackageBuilderErrors errors = pbuilder.getErrors();
    if (errors.getErrors().length > 0) {
        System.out.println("errors exit in the packageBuilder component");
        for (int i = 0; i < errors.getErrors().length; i++) {
            System.out.println(errors.getErrors()[i]);
        }
        throw new IllegalArgumentException("knowledge could not be parsed.");
    }

    //3. Add package to rule base
    try {
        rbase.addPackage(pbuilder.getPackage());
    } catch (Exception e) {
        System.out.println("Error: " + e);
    }
}

```

Figure 32: Drools Engine Instance

On deploying PMA with the Drools logic (in Figure 32); the agent died prematurely (shown in Figure 33) on executing the afterMove method. A TickerBehaviour executed the Drools instance in this case.

```
java.lang.NullPointerException
    at mobilecomponent.MobileAgent$HelloDrools.runRules(MobileAgent.java:302)
)
    at mobilecomponent.MobileAgent$HelloDrools.onTick(MobileAgent.java:297)
    at jade.core.behaviours.TickerBehaviour.action(TickerBehaviour.java:70)
    at jade.core.behaviours.Behaviour.actionWrapper(Behaviour.java:340)
    at jade.core.Agent$ActiveLifeCycle.execute(Agent.java:1530)
    at jade.core.Agent.run(Agent.java:1468)
    at java.lang.Thread.run(Thread.java:745)
ERROR: Agent Mobile_Agent_0 died without being properly terminated !!!
State was 2
```

Figure 33: Migration Exception

```
*** Uncaught Exception for agent Mobile_Agent_0 ***
org.drools.RuntimeDroolsException: Unable to resolve class 'mobilecomponent.Mess
age'
    at org.drools.base.ClassFieldAccessorCache.getClass(ClassFieldAccessorCa
che.java:126)
    at org.drools.base.ClassFieldAccessorCache.getClassObjectType(ClassField
AccessorCache.java:48)
    at org.drools.reteoo.ClassObjectTypeConf.<init>(ClassObjectTypeConf.java
:73)
    at org.drools.common.ObjectTypeConfigurationRegistry.getObjectTypeConf(O
bjectTypeConfigurationRegistry.java:71)
    at org.drools.common.NamedEntryPoint.insert(NamedEntryPoint.java:145)
    at org.drools.common.AbstractWorkingMemory.insert(AbstractWorkingMemory.
java:886)
    at org.drools.common.AbstractWorkingMemory.insert(AbstractWorkingMemory.
java:845)
    at mobilecomponent.MobileAgent$HelloDrools.initiliseMessageObject(Mobile
Agent.java:286)
    at mobilecomponent.MobileAgent$HelloDrools.action(MobileAgent.java:272)
    at mobilecomponent.MobileAgent.afterMove(MobileAgent.java:170)
    at jade.core.Agent$1.afterMove(Agent.java:1006)
    at jade.core.mobility.AgentMobilityService$TransitLifeCycle.init(AgentMo
bilityService.java:1407)
    at jade.core.Agent.run(Agent.java:1465)
    at java.lang.Thread.run(Thread.java:745)
ERROR: Agent Mobile_Agent_0 died without being properly terminated !!!
State was 2
```

Figure 34: Migration Exception

To identify the bug, a CyclicBehaviour was integrated to execute a Drools instance on relocation. The agent died prematurely but with a detailed log (in Figure 34). Considering ReteOO optimisations in concurrent and parallel matching strategies (introduced in Section 7.1.5); the research reviewed on the algorithm's performance in development lists. Drools version 5.3.0 doesn't support concurrency majorly under the *Class Field Accessor Cache* and *Composite Class Loader* as was concluded in [128].

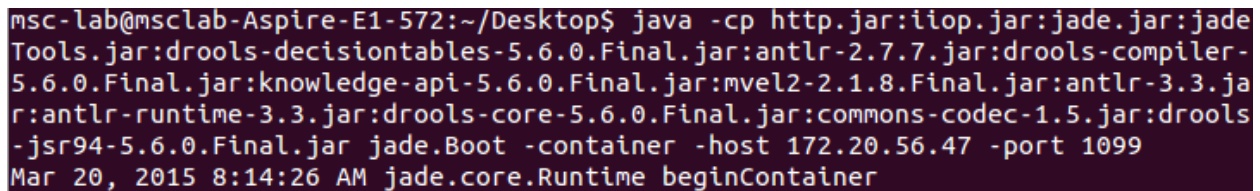
The *Class Field Accessor Cache* (shown further on) issued the exception in Figure 34.

ClassName: ClassFieldAccessorCache

This is the method which throws Exception during concurrency. Highlighted Line number throws the exception

```
public Class getClass(String className) {  
    try {  
        // Exception happens here. And This happens only during multithreading or concurrent calls.  
        return this.classLoader.loadClass( className );  
    }  
    catch ( ClassNotFoundException e )  
    {throw new RuntimeException( "Unable to resolve class '" + className + "'" ); }  
}
```

Drools revisions with a fix for multi-threading and concurrency are not defined explicitly [128]. Experienced Drools practitioners modify source code to suit their needs. Drools 5.6.0 with a partial fix was highlighted as a possible solution. The research recompiled the PMA and evaluated the agent with a remote container (created in Figure 35) running Drools 5.6.0. The agent migration wasn't successful as well in this scenario.



```
mssc-lab@msclab-Aspire-E1-572:~/Desktop$ java -cp http.jar:iiop.jar:jade.jar:jade  
Tools.jar:drools-decisiontables-5.6.0.Final.jar:antlr-2.7.7.jar:drools-compiler-  
5.6.0.Final.jar:knowledge-api-5.6.0.Final.jar:mvel2-2.1.8.Final.jar:antlr-3.3.jar:  
antlr-runtime-3.3.jar:drools-core-5.6.0.Final.jar:commons-codec-1.5.jar:drools-  
jsr94-5.6.0.Final.jar jade.Boot -container -host 172.20.56.47 -port 1099  
Mar 20, 2015 8:14:26 AM jade.core.Runtime beginContainer
```

Figure 35: Drools 5.6.0 in JADE

Considering the time devoted to a Drools enabled PMA; the research explored Jess for rule based reasoning. The work on JADE and Drools was documented for future work.

7.1.5.2 JADE and Jess: Exceptions

Jess was originally considered for implementing expert systems. In MAS, the engine is used to build software which reason using knowledge supplied in the form of declarative rules. In principle a JADE agent is single-threaded (from Section 4.7.4). The Jess Rete.run method enables the engine to successively fire rules and return only when all rules are fired (when the engine stops); therefore, meanwhile the calling thread will be blocked. If the calling thread is blocked; then the entire single-threaded JADE agent will block [129]. The research incorporated techniques that allow separate behaviours to be executed from within a PMA at different

instances based on the behaviour definitions to avoid thread blocking. A JADE container was initialised at the destination node as follows:

```
root@msclab-Aspire-E1-572:/home/msc-lab/Desktop# java -cp http.jar:iop.jar:jade.jar:jadeTools.jar:jadex_jadeadapter.jar:jess.jar:jsr94.jar jade.Boot -container -host 172.20.56.49 -port 1099
```

Figure 36: Starting a JADE Runtime

It was anticipated that Jess can relocate its declarative rule file to a remote node to implement rule reasoning. An attempt to relocate the component issued an exception (shown in Figure 38) at the destination node.

```
107 //##Reasoning component##
108 Rete engine = new Rete();
109 engine.batch("/root/NetBeansProjects/mobilecomponent/src/mobilecomponent/rulesfile.clp");
110 engine.watchAll();
111 Fact fact = new Fact("VN_Reasoning", engine);
112 fact.setSlotValue("cpuinfo", new Value(new ValueVector().add(cpu_info), RU.LIST));
113 engine.assertFact(fact);
114 // Run the rule and report the result
115 int count = engine.run();
116 //returns decision
117 if (count > 0) {
118     action = engine.getGlobalContext().getVariable("**var*").toString();
119     System.out.println("action = " + action);
120 } else {
121     System.out.println("ignore inference");
122 }
```

Figure 37: Jess Instance

```
Jess reported an error in routine batch.
Message: Cannot open file.
    at jess.Batch.findDocument(Unknown Source)
    at jess.Batch.batch(Unknown Source)
    at jess.Batch.batch(Unknown Source)
    at jess.Batch.batch(Unknown Source)
    at jess.Batch.batch(Unknown Source)
    at jess.Rete.batch(Unknown Source)
    at jess.Rete.long(Unknown Source)
    at jess.Rete.<init>(Unknown Source)
    at jess.Rete.<init>(Unknown Source)
    at mobilecomponent.MobileAgent$TickerBehaviourImpl.onTick(MobileAgent.java:108)
    at jade.core.behaviours.TickerBehaviour.action(TickerBehaviour.java:70)
    at jade.core.behaviours.Behaviour.actionWrapper(Behaviour.java:340)
    at jade.core.Agent$ActiveLifeCycle.execute(Agent.java:1530)
    at jade.core.Agent.run(Agent.java:1468)
    at java.lang.Thread.run(Thread.java:745)
Caused by: java.io.FileNotFoundException: scriptlib.clp (No such file or directory)
```

Figure 38: FileNotFoundException

To capture the declarative rule file across to the destination runtime environment, a `fileInputStream` was defined (as in Figure 39).

```
//convert rulefile to bytes
FileInputStream fileInputStream;
file = new File("/root/NetBeansProjects/mobilecomponent/src/mobilecomponent/rulesfile.clp");
bFile = new byte[(int) file.length()];
try {
    //convert file into array of bytes
    fileInputStream = new FileInputStream(file);
    fileInputStream.read(bFile);
    fileInputStream.close();
    System.out.println("Done");
} catch (IOException e) {
    e.printStackTrace();
}
```

Figure 39: Rule File to Bytes

Selecting a specific PMA method to create the file stream from the source was crucial. The most sensible approach was to include `fileInputStream` in the `beforeMove` method. However an exception in Figure 40 was observed on migration. Defining the `fileInputStream` in the PMA void setup method resolved the exception.

```
MobileAgent Migrating to Volunteer node: Container-1
java.io.FileNotFoundException: /root/NetBeansProjects/mobilecomponent/src/mobilecomponent/r
ulesfile.clp (No such file or directory)
    at java.io.FileInputStream.open(Native Method)
    at java.io.FileInputStream.<init>(FileInputStream.java:146)
    at mobilecomponent.MobileAgent.beforeMove(MobileAgent.java:192)
    at jade.core.Agent$1.beforeMove(Agent.java:1002)
    at jade.core.mobility.AgentMobilityService$TransitLifeCycle.execute(AgentMobilitySe
rvice.java:1418)
    at jade.core.Agent.run(Agent.java:1468)
    at java.lang.Thread.run(Thread.java:745)
```

Figure 40: IOException

The PMA was deployed to a remote node where processing and reasoning initiated properly.

```
Mar 18, 2015 8:34:28 AM jade.core.mobility.AgentMobilityService$CommandSourceSink handleInf
ormMoved
SEVERE: Error in agent serialization. Abort transfer. java.io.NotSerializableException: jav
a.io.FileOutputStream
```

Figure 41: Serialisation Exception

However, the agent didn't migrate back to report on processing results due to an exception in Figure 41. The PMA serialized a fileoutputstream directly or the stream was reachable from objects being serialised. Declaring the stream as transient resolved the exception.

```
//converting byte array into a file
fileOutputStream = new FileOutputStream("/home/finalfile.clp");
fileOutputStream.write(bFile);
fileOutputStream.close();
```

Figure 42: Output Stream

7.2 Storage Component

The software logic that implements the DM and C-AP file-system services are discussed in the following sections.

7.2.1 DM File-System: Upload Service

The profile shown in Figure 43 specifies the agents and integral APIs for the DM file-system upload service.

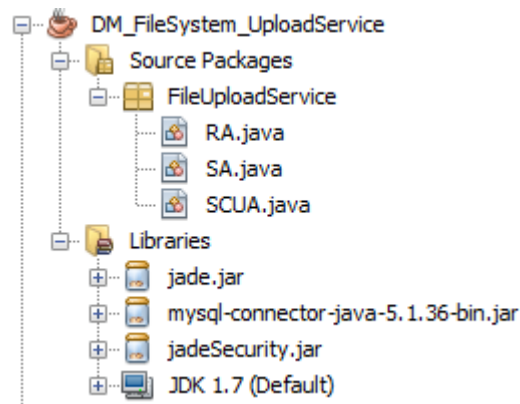


Figure 43: DM project

7.2.1.1 SCUA

The SCUA displays a file chooser (in Figure 44) when executed to locate a file and its name. The SCUA then converts the file content into a JADE message data type supported (see Appendix B.1.1 for code snippet).

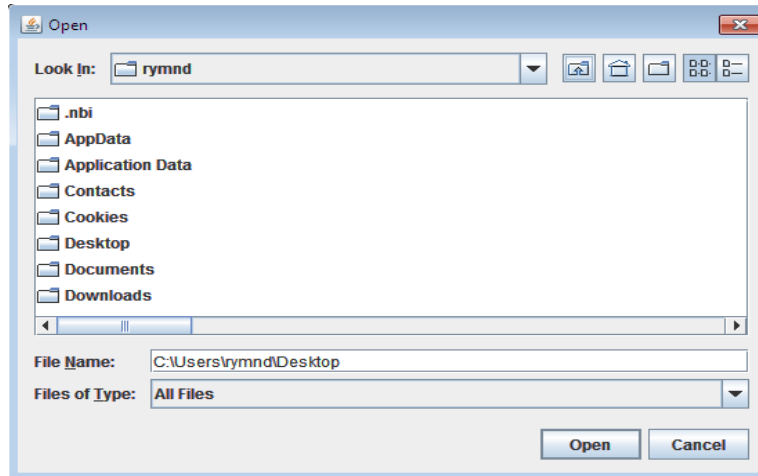


Figure 44: File Chooser

The SCUA searches and stores DF returned broker services as RAs_AIDs (in Appendix B.1.2). The SCUA then forwards its SCUA AID, the source filename and the file payload as forwards_file_metadata to identified RAs (returned in RAs_AIDs). The forwards_file_metadata parameters are important for the following reasons (see Appendix B.1.3 for parameters):

1. The SCUA AID uniquely identifies the parameters to stored files when an upload is successful;
2. Source filename allow the platform to recreate the same file container on file download. Java FileUtils converts file contents to byte arrays; hence the metadata on filename has to be preserved separately.

If an upload is successful; Figure 45 shows an upload status returned by a RA (see Appendix B.1.4 for code snippet).

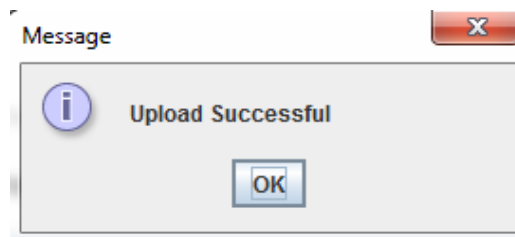


Figure 45: Upload Status

7.2.1.2 RA

The RA registers (in Appendix B.1.5) with the DM file-system and responds to SCUA upload requests. The agent receives and defines the SCUA content as file_metadata. From Figure 46,

the RA extracts the content and computes the size of the file uploaded to determine the nodes appropriate for storage through the OPTIMISESTORAGE function.

```

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
    try {
        //Receive Forwards_file_metadata here
        File_metadata = (ArrayList) msg.getContentObject();
    } catch (UnreadableException ex) {
        Logger.getLogger(RA.class.getName()).log(Level.SEVERE, null, ex);
    }

    //extract filepayload from received File_metadata
    filepayload = (byte[]) File_metadata.get(0);
    file_size_kb = filepayload.length/1024;
    //create sender SCUA reply object
    reply = msg.createReply();
    try {
        //call function to determine suitable SA/SS
        OPTIMIZESTORAGE(file_size_kb);
        //add Behaviour to send files to identified SA/SS
        addBehaviour(new send_upload());
    } catch (SQLException ex) {
        Logger.getLogger(RA.class.getName()).log(Level.SEVERE, null, ex);
    }
} else {
    block();
}
}

```

Figure 46: Handling SCUA Requests

The SA AID and its services are defined as “storageagent”+mac_address as introduced in Section 6.2.5.1. Apart from uniquely identifying SAs; the variable enables SA to SS relationship mapping. Information stored in MySQL is extracted as strings. Storing SA AID objects provided by a SA in a database table presented challenges as the string AID identifiers couldn’t be type casted to standard JADE AID variables. SAs were hence configured to provide SS MAC addresses to a MySQL database rather than their AIDs.

With reference to the OPTIMISESTORAGE function logic, if an upload is less than 60MB, a RA identifies the MAC addresses of SSs with 100 GB disk space. The MAC addresses of all shared nodes in a platform are identified by mac_address. MAC addresses in mac_address are appended with “storageagent” which allow RA to identify their SA AIDs (storageagent [count++]) from the DF. Three nodes are selected from the file-system for redundancy in file storage. The RA then forwards an upload request which consists of SCUA AID, source filename and file payload to chosen SAs (shown in Appendix B.1.7). A CyclicBehavior which listens on SA feedbacks is simultaneously initialised on sending SA bound requests. An upload status is then returned to the SCUA provided all SA feedbacks are returned.

7.2.1.3 SA

An upload service is dependent on message passing between RAs and SAs. A SA extracts the SS MAC address and registers its service as "*storageagent*" + *mac_address* on start-up (code snippet in Appendix B.1.10). The agent then extracts the disk utilisation parameters (in Appendix B.1.11) and stores the parameters for the first time in a database. The registered SS information is updated after a defined interval (in Appendix B.1.13). A directory that stores uploaded files is also created in the SS home directory.

```
try {
    File_metadata_received = (ArrayList) RA_Request.getContentObject();
} catch (UnreadableException ex) {System.out.println("Exception--"+ex);}
reply = RA_Request.createReply();
if (File_metadata_received != null) {
    count++;
    //Create SS Filename--> file+count++;
    newname = "file"+count;
    //Define file Container name in HOME's Uploads_Folder for the byte []payload
    CreateContainer = new File(Uploads_Folder+newname);
    try {
        //create file here
        CreateContainer.createNewFile();
        //start the output stream
        oFile = new FileOutputStream(CreateContainer);
        //Write byte[]--> CreateContainer
        oFile.write((byte[]) File_metadata_received.get(0));
        oFile.close();
        //intergrateParameters to the SS stored file
        Feedback_params = new ArrayList();
        Feedback_params.add(newname); //add SS fileName
        Feedback_params.add(mac_address.toString()); //add SS mac Address
        Feedback_params.add(File_metadata_received.get(1)); //add source file name
        Feedback_params.add(File_metadata_received.get(2)); //add senders SCUA AID
        reply.setPerformative(ACLMMessage.INFORM);
        reply.setContentObject(Feedback_params);
        myAgent.send(reply);
    } catch (IOException ex) {System.out.printf("exception"+ex);}
} else {
    reply.setPerformative(ACLMMessage.FAILURE);
    myAgent.send(reply);
}
```

Figure 47: Upload Service Implements

From the RA received parameters in Figure 47; the SA retrieves the file payload and writes the file in a SS under a new assigned alias (newname). The newname, SS MAC address, source file name and SCUA AID are then returned back to the RA as an upload feedback.

7.2.2 DM File-System: Download Service

The service discussed in this section downloads a file uploaded in Section 7.2.1. The platform assumes the same agent names but with different roles and negotiation patterns.

7.2.2.1 SCUA

A SCUA requests for broker services which identify locations to a stored file (see Appendix B.2.1 for code snippet). From a selected broker service (RA_AIDs [0]) (in Appendix B.2.2); the SCUA requests for an uploaded file by sending its AID.

```
if (feedback_params!= null) {
try {
download_params = (ArrayList) feedback_params.getContentObject();
} catch (UnreadableException ex){ System.out.println("Error--"+ex);}
if(download_params!=null){
//Define Path to stre file
Home_PATH = System.getenv("HOME");
File Download_path = new File(Home_PATH+"/");
//Extract && define Original File Container name
file_download = new File(Download_path+download_params.get(1).toString());
try {
//Create the source file
file_download.createNewFile();
returned = new FileOutputStream(file_download);
//Write byte[] contents of the file
returned.write((byte[])download_params.get(0));
returned.close();
//Show upload status to sender
SwingUtilities.invokeLater(new Runnable() {
public void run() {
JOptionPane.showMessageDialog(null, "Upload_Successful--"+
"check_File_in_directory--"+Home_PATH);
}
});
} catch (IOException ex) {System.out.println("error"+ex);}
} else {System.out.println("no File returned");}
} else{
block();
}
}
```

Figure 48: Storing Downloaded File

In responds to a download request, RA returns the source filename and byte payload to the SCUA. The SCUA's CyclicBehaviour logic (in Figure 48) then accesses the user's home directory, creates a source file container (file name and extension) and writes the file byte content. A download status is then provided to the user including the file path.

7.2.2.2 RA

The RA handles SCUA requests and identifies locations to an uploaded file. The RA service is registered (in Appendix B.2.3) to receive SCUA download requests. To identify a requested file, the RA receives the SCUA AID (in Appendix B.2.4) and queries a LocationsDatabase to identify a record associated with the SCUA AID. The affiliated SS MAC addresses, SS filenames and the source filename are then linked to the provided SCUA AID. The source filename to SS file name matching parameters are then added systematically to an array list with the SS MAC addresses bundled up in the locations parameter (see Appendix B.2.5 for code snippet).

The RA then appends the SS MAC addresses in locations to “*storageagent*” to identify the associated SA AIDs from the DF (in Appendix B.2.6). If SA AIDs are identified, requests containing SCUA AID, SS filename and source file name parameters are forwarded to identified SAs (in Appendix B.2.7).

The CyclicBehaviour logic (in Appendix B.2.8) initiates and listens on download feedback messages satisfying template definitions and assigns the responses to specific content objects (e.g. downloadfeedback1). Provided the SCUA AID and source filename parameters match for SA feedbacks; the file payload and source filename parameters are returned to a requestor SCUA (see Appendix B.2.9 for code extract).

7.2.2.3 SA

The SA accepts download requests from the RA and retrieves the file specified. The registration and updating of storage oriented node information is maintained as in Section 7.2.1.3. From Figure 49, the RA receives download parameters and retrieves the SS filename including its directory path. The file is then returned as extract_file. Since it is important for the file to assume the same name and extension; an array list (Payload) that encapsulate the required content (extract_file, source filename and SCUA AID) is defined. The Payload object is then enveloped in a reply and returned to RA as a download feedback.

```

if (msg != null) {
    try {
        download_params = (ArrayList) msg.getContentObject();
        ACLMessage reply = msg.createReply();
        if (download_params != null) {
            //defines the SS file Path
            File_PATH = System.getenv("HOME");
            //extracts&&defines the SS specific filename
            filelocation = new File(File_PATH+"/StorageFolder/"+download_params.get(0));
            //converts the file to byte[]
            extract_file = new byte[(int) filelocation.length()];
            //creates A list for byte[],SCUA AID and Source filename
            Payload = new ArrayList();
            Payload.add(extract_file);//byte[] file
            Payload.add(download_params.get(1));//source-filename
            Payload.add(download_params.get(2));//requestor SCUA AID
            reply.setPerformative(ACLMessage.INFORM);

            try {
                reply.setContentObject(Payload);
                //Send feedback to RA here
                myAgent.send(reply);
            }catch (IOException ex) {
                System.out.println("Error--"+ex);
            }
            else {
                reply.setPerformative(ACLMessage.FAILURE);
            }
        }catch (UnreadableException ex) {
            System.out.println("Error"+ex);
        }
    }
}
}

```

Figure 49: File Retrieval

7.2.3 C-AP File-System: Upload Service

A storage platform implementing the C-AP upload technique is introduced in this section. The C-AP file-system varies from the DM file-system approach in the way file uploads are handled through RA and SA agent types. The SCUA code remains unchanged for both DM and C-AP file-systems. The code snippets for registering agents and detecting disk utilisation patterns are also constant.

7.2.3.1 RA

The RA service is registered with the DF as in the DM file-system (Section 7.2.1.2). The RA retrieves the file payload and computes its size when an upload request is forwarded by SCUA (see Appendix B.3.1 for code snippet). The shared SS records registered in the nodeinfo

database are then selected by the RA (in Appendix B.3.2). For MAC addresses identified; the associated SA AIDs are returned on querying the DF as a sharednodes variable.

```
//Function for chunking files
void chunkserver_func(byte[] file_upload,int file_size){
    int CHUNK_SIZE=10*1024*1024;//split file into 10MB chunks
    final byte[][]chunked_files = new byte[(int)Math.ceil(
        file_upload.length/(double)CHUNK_SIZE)][];
    int start = 0;
    for(int i = 0; i < chunked_files.length; i++) {
        if(start + CHUNK_SIZE > file_upload.length) {
            chunked_files[i] = new byte[file_upload.length-start];
            System.arraycopy(file_upload, start, chunked_files[i],
                0, file_upload.length - start);
        }
        else{
            chunked_files[i] = new byte[(int)CHUNK_SIZE];
            System.arraycopy(file_upload, start, chunked_files[i],
                0, (int) CHUNK_SIZE);
        }
        start += CHUNK_SIZE ;
    }
}
```

Figure 50: Chunking a file

To initiate file chunking; the file payload and size calculated are parsed into the chunkserver_func function (in Figure 50). The function reads in the file payload and splits the payload into 10 MB chunks. On splitting the file, logic which maps the file size to the number of SS/SA required for the file is selected. If a file is 10MB or less, the chunk replica is forwarded to a single node/SA (in Appendix B.3.3). At most two SA AIDs and parameters are selected for a file payload in the 11MB to 20MB range. The pattern is maintained up to the largest upload size supported by the platform.

A CyclicBehaviour is initialised on forwarding the chunk replicas to identified SAs to listen on append feedbacks. Templates and receive () methods in the behaviour are defined to extract these responses. If all chunk append feedbacks for a file upload are returned, the Savechunklocations function is called to save the parameters in a database (in Appendix B.3.4). The MAC addresses, byte offsets and appended file names for each file chunk are stored as a single object reducing on the number of fileappendchunklocations columns created (see Appendix B.3.5 for code snippets).

7.2.3.2 SA

```
public void action() {
    MessageTemplate mt = MessageTemplate.MatchPerformative
        (ACLMessage.ACCEPT_PROPOSAL);
    final ACLMessage msg = myAgent.receive(mt);
    if(msg != null) {
        file_length_b4_append = new byte[(int)file.getAbsoluteFile().length()];
        //defines the chunk append starting offset
        first_offset_value = file_length_b4_append.length+1;
        getchunks = msg.getByteSequenceContent();
        reply = msg.createReply();
        if (getchunks != null) {
            try {
                //Chunk append here
                oFile = new FileOutputStream(file,true);
                oFile.write(getchunks);
                oFile.close();
                //defines the last chunk append offset
                last_offset_aftr_append = (first_offset_value+getchunks.length);
                String append_offset = (first_offset_value)+"-"+last_offset_aftr_append;
                append_paragms = new ArrayList();
                append_paragms.add(mac_address);
                append_paragms.add(file.getName());
                append_paragms.add(append_offset);
                //sets the performative for response..
                reply.setPerformative(ACLMessage.INFORM);
                reply.setContentObject(append_paragms);
                myAgent.send(reply);
            } catch (IOException ex) {
                System.out.println("Error -- " + ex);
            }
        } else {
            reply.setPerformative(ACLMessage.FAILURE);
        }
    } else {
        block();
    }
}
```

Figure 51: Atomic Append Approach

To validate the C-AP technique, the code in Figure 51 is executed as follows:

1. The agent defines the byte length of the local file before an append as file_length_b4 _append;
2. If a RA request is non-zero; the SA extracts the chunk and appends it to the end of a SS file;
3. The byte length of the new file is then computed to determine the last byte offset of an atomic append. The resultant byte offset is then formulated and captured as a string value append_offset;
4. The append parameter (append_params) which include the SS MAC address, chunk byte offset and SS appended file name is encapsulated and returned as feedback to the RA.

7.2.4 C-AP File-System: Download Service

The C-AP file-system's download service which recovers chunk appends and reconstructs an uploaded file is described in this section.

7.2.4.1 RA

If RA is subject to a SCUA download request; the agent extracts all chunk parameters and selects a record that matches the sender's SCUA AID (in Appendix B.4.1). The returned string chunk objects include a SS MAC address, SS file name and chunk byte offset parameters. To uniquely identify the string chunk objects returned, the RA generates an individual array comprised of a MAC address, SS file name and byte offset for each chunk location stored (code extract in Appendix B.4.2).

For a chunk append associated with an array (i.e. `Chunk1A_Tokens`); the RA identifies the SS MAC address where the file is hosted and resolves the related SA AID from the DF (in Appendix B.4.3). If the SA AID is available the code envelops the parameters (SS file name and chunk byte offset) and forwards a chunk retrieval request to the SA

A behaviour that listens on feedbacks is initialised simultaneously. To identify and extract chunk responses the behaviour defines templates that match set conversation identities. If all chunk units are returned, the chunk payloads are appended and the generated file is returned to SCUA (in Appendix B.4.4). On reaching the SCUA, the file is written to the requesting user node.

7.2.4.2 SA

To handle chunk retrieval requests from RA, the code in Figure 52 is executed by a SA as follows:

1. The agent extracts the string byte offset variable and defines start and end offsets;
2. The SS file name encapsulating the chunk is extracted; and the file is located in the SS directory;
3. The byte length of the SS file is computed. The chunk is then copied into chunkfile by specifying the start and end points of the byte file;
4. The chunk retrieved is then returned to the RA as feedback.


```

ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
    try{
        //Receive params
        getParams = (String[]) msg.getContentObject();
    }catch(UnreadableException ex){
        System.out.println("Exception -- "+ex);
    }
    ACLMessage reply = msg.createReply();
    //if parameters != zero
    if (getParams != null) {
        ///###getParams[2] = "chunkfile" name
        //getParams[3] = the byterange written e.g "24-8419848"
        //remove the "-" in start_byte_index and the end_byte_index
        delimiters = "-";
        split_params_offset = getParams[3].split(delimiters);
        System.out.println("Startbytepoint"+split_params_offset[0] );
        System.out.println("EndbytePoint"+split_params_offset[1]);
        //chunk file name here
        System.out.println("params2 -> "+getParams[2]);
        //Direct where the chunk is stored is stored
        file = new File(home+"/StorageFolder/"+getParams[2]);
        //read the bytes in the stored file <-> determine range to read from
        byte[] bFile = new byte[(int) file.length()];
        //convert string points to integers
        //startbyterange-> split_params_offset[0]
        //Endbyte range -> split_params_offset[1]
        int startbytepoint = Integer.valueOf(split_params_offset[0]);
        int endbytepoint = Integer.valueOf(split_params_offset[1]);
        //extract the chunk here
        chunkfile = Arrays.copyOfRange(bFile, startbytepoint, endbytepoint);
        reply.setPerformative(ACLMessage.INFORM);
        reply.setByteSequenceContent(chunkfile);
    }else{
        reply.setPerformative(ACLMessage.FAILURE);
    }
    myAgent.send(reply);
}else{
    block();
}

```

Figure 52: Chunk Retrieval

7.3 Conclusion

This chapter presented the MAOG compute and storage codes implemented to demonstrate the DM and C-AP designed services. The agent components in the file-systems were developed on JADE MAS. Base classes, methods and code fragments used to implement specific systems including the CNP and rule based reasoning modules were discussed. The next chapter present the experimentation conducted on the developed prototypes.

Chapter 8

8 Results and Analysis

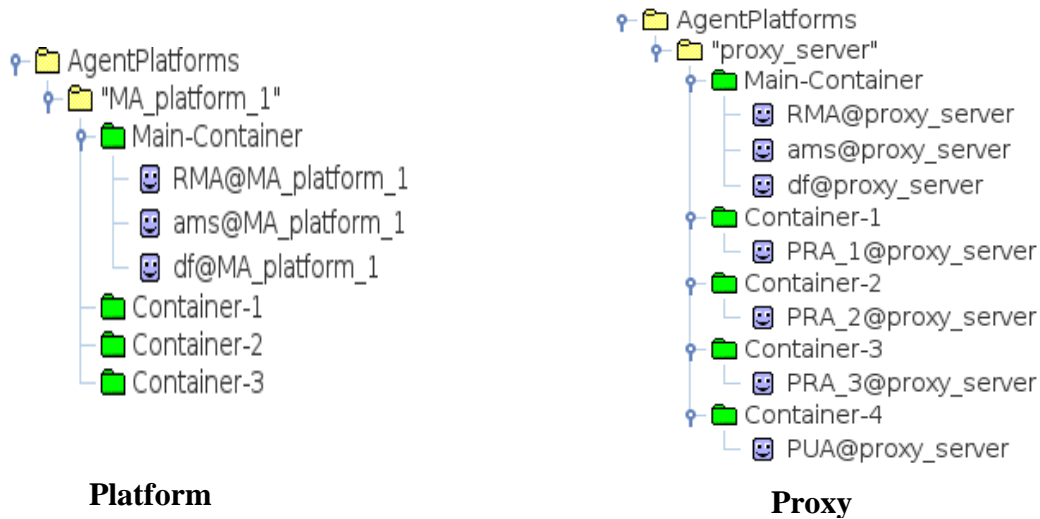
The proof of concept prototypes designed and implemented are evaluated in this chapter. The results are based on a number of experiments conducted in validating the multi-agent approach to public resource computing. The MAOG services conceptualize a virtual low-cost commodity grid founded on the VC paradigm. A discussion on the findings is presented in the following sections.

8.1 Compute Component

The MA platform results are discussed in the following sub-sections.

8.1.1 Evaluation

The component is focused on the functional and technical adequacy of MAS in utilizing shared processor capabilities. To assess the CNP in resource allocation, three platforms identified by PRAs were set-up. The platforms were joined through a proxy-server. The MA platform and proxy-server profiles are shown below.



Nodes running Linux with different processor specification were shared to platforms by running resource provider NPAs. For a PMA to initialize successfully on migration, it was important to

initialize JADE containers on nodes running the same Java version. The underlying environment was setup as in Figure 53.

```
msc-lab@msclab-Aspire-E1-572:~$ update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority  Status
  ----
0            /usr/lib/jvm/java-6-openjdk-1386/jre/bin/java  1061     auto mode
1            /usr/lib/jvm/java-6-openjdk-1386/jre/bin/java  1061     manual mode
* 2          /usr/lib/jvm/java-7-openjdk-1386/jre/bin/java  1051     manual mode

Press enter to keep the current choice[*], or type selection number: 2
```

Figure 53: Configuring Java

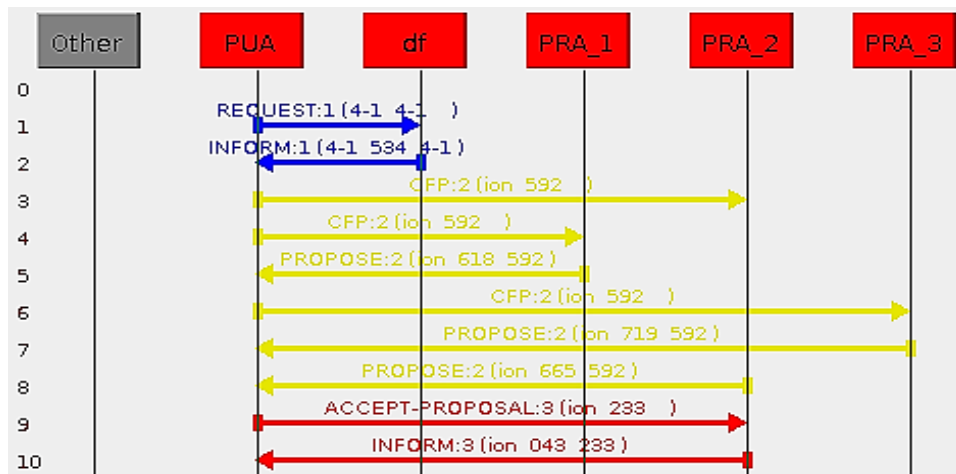


Figure 54: CNP Mechanism

Figure 54 shows the steps taken to validate the CNP mechanism when a PUA was loaded:

1. **Lines 1-2:** PUA request PRA services from the DF;
2. **Lines 3-8:** The PUA submits call for proposals to identified PRAs and obtains shared node mean load averages as bids;
3. **Lines 9-10:** PUA offers a contract to PRA_2 and receives shared node information in return.

The proxy profile in Figure 55 was displayed on loading a PMA and selecting a compute resource. Turnaround Time (TAT) in processing was defined as the interval between PMA deployment, application processing and return of results to the source node.

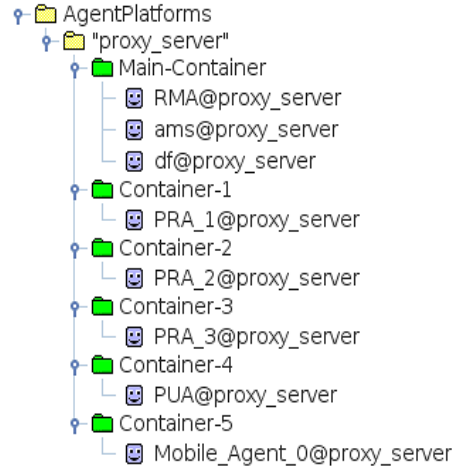


Figure 55: Loading a PMA

Random operations were induced during PMA processing on shared nodes to simulate resource providers recalling nodes. The patterns were important to confirm the rule reasoning functionality in automated remigrations to prioritize resource provider activity. Table 11 lists the TATs recorded for a sample platform selected for compute tasks. Variations in node processor capabilities, bandwidth and PMA migrations in processing were observed as the main factors influencing the trend in TATs.

Deployment (n)	1	2	3	4	5	6
TAT (milliseconds)	71043	141875	141952	431137	70765	141821

Table 11: PMA TAT

Support for strong mobility which can be offered through Java thread capturing techniques (detailed in Section 4.10.1) wasn't integrated in the MA compute component. Although the methods aren't integral to this research, they have the potential to reduce the processing TATs since computations reinitialize from previous captured state. The ability of PMAs to successfully process embedded applications verified the functional and technical adequacy of the MA paradigm. Additionally, the compute platform was adaptive to computing in heterogeneous environments by: (1) integrating the CNP in resource identification and allocation; and (2) adding a rule reasoning component to reason about changes in the execution environment.

8.2 MAOG File-Systems

The DM and C-AP file-systems were experimented to benchmark the storage services using defined metrics. The processes aimed at recommending an optimum storage approach for an

ICT4D context based on quantifiable measures. Random files (i.e. 5,10,15...n MB) were simulated to the platforms in an experimental environment. The files were generated as follows:

```
dd if=/dev/urandom of=filename.log bs=1M count=file size
```

The interaction intervals between SCUA, RA and SA were recorded. The times (milliseconds) start when a SCUA request storage oriented services until feedback is returned. The overall upload and download times are defined as Upload Round Trip Time (URTT) and Download Round Trip Time (DRTT) based on a service requested. The Round Trip Times (RTTs) are further broken down to account for elementary transitional times.

8.2.1 Measurement Criteria

The experimental environment was setup as in Section 6.2.3. Java's *System.currentTimeMillis()* was utilised to calculate transition times in handling identified services. The transition times that make up a RTT are shown in Figure 56.

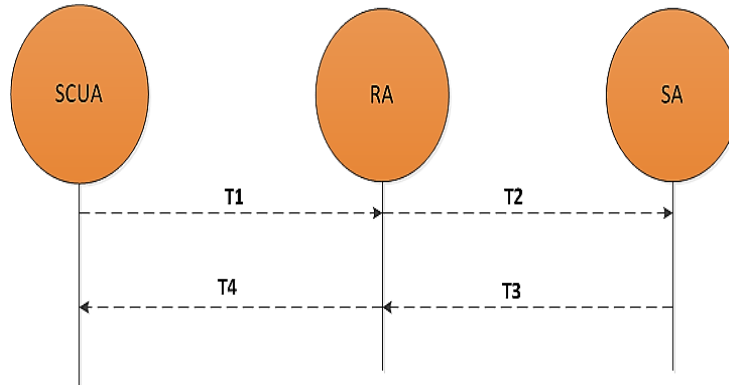


Figure 56: Elementary Intervals

An upload interaction includes the following:

- **U_T1:** This is the time taken to forward a file payload to RA.
- **U_T2:** Include the time taken for RA to:
 - identify suitable nodes (i.e. identified by MAC address from database);
 - resolve the identities of the SAs affiliated with the nodes;
 - send received file payload to identified SAs.
- **U_T3:** The time start when SAs receive RA requests until feedback is returned to RA.

- **U_T4:** This is the time taken to return feedback to a sender from RA.
- **U_ReponseTime:** Is the overall response time. The value sums up U_T1,U_T2,U_T3 and U_T4 times.

A download request includes the following times:

- **D_T1:** the time taken for SCUA to send its AID to RA;
- **D_T2:** Interval start when RA receives the SCUA AID, identifies parameters to stored file and sends a download request to SAs;
- **D_T3:** In the time range, a SA receives RA request, retrieves SS file/append and returns the payload to RA;
- **D_T4:** Transition returns the file payload to SCUA;
- **D_ReponseTime:** Is the response time for a download service.

The response times in handling upload and download requests are generalised as follows:

- **U_ReponseTime=URTT = U_T1 + U_T2 + U_T3 + U_T4**
- **D_ReponseTime =DRTT = D_T1 + D_T2+ D_T3+ D_T4**

The Turn Around Time (TAT) was introduced to compare on file read and write speeds associated with developed storage methods implementing DM and C-AP. In addition the parameter identifies: (1) the effects of erratic node connectivity on RTT; and (2) the filesystem method that enhance performance in heterogeneous connectivity settings. For the upload and download operations, the TAT is generalised as follows:

$$\mathbf{TAT = T2+T3}$$

8.2.2 Experimentation

The DM and C-AP file-systems were evaluated based on metrics defined in the previous section. The DRTT, URTT and UTAT were collected from simulations conducted on the two storage systems. The main motivations for the parameters as reference points for evaluation include:

1. **URTT:** Allow deduction on the response time in requesting an upload service and getting an upload feedback.

2. **UTAT:** Enable performance measurement between the mirroring and chunking approaches proposed earlier which influence the URTT. The DTAT values weren't collected as the research assumed UTAT as adequate to validate and contrast the identified storage approaches. The following assumption was defined in this regard: $UTAT \approx DTAT$.
3. **DRTT:** Defines the projected response times in requesting a download service.

A linear regression analysis was utilised to model the relationship between the file sizes (in MB) and measurement metrics defined. The main interest was in how the explanatory variable (file size) explain or change the response variables (measurement metrics). If plots of the variables resemble a straight line, a linear relationship may be assumed. The following hypothesis was stated for linear regression models tested:

H₀: There is no linear relationship between explanatory and response variables

H₁: Linear relationships exists between explanatory and response variables

The measures of the strength and direction of a linear model were then determined by the covariance and the Pearson correlation coefficient. The Pearson product-moment coefficient measures the degree of linear dependence between two variables. The coefficient gives values between +1 and -1 inclusive; where 1 is total positive correlation, -1 is total negative correlation and 0 is no correlation. The covariance, correlations and regression lines (lines of best fit) were computed using the R project for statistical computing.

8.2.3 DM Service Evaluation

The message passing sequences to achieve specific objectives are identical in the DM and C-AP file-systems. The DM file-system upload functionality is validated in Figure 57. The DM file-system writes an identical file uploaded redundantly on three shared SSs selected from the infrastructure. Lines 1-2 request RA identities from the DF. Line 3 sends an upload request encapsulating the file payload. The SS MAC addresses and affiliated SA AIDs are resolved in lines 4-9. In lines 10-15 the file is written in respective SSs. The upload feedback is returned to a requestor SCUA in line 16.

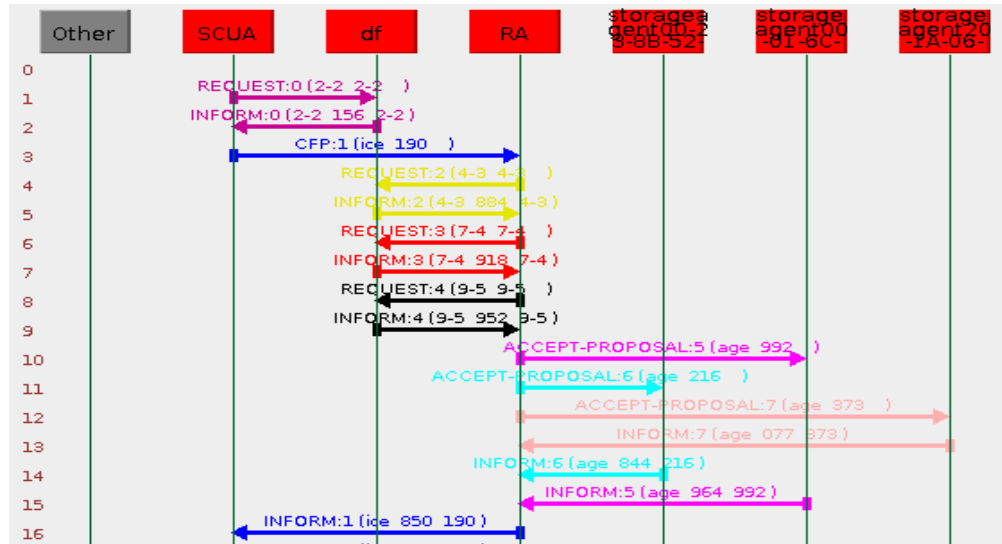


Figure 57: Upload Interactions

The download service functional adequacy is certified in Figure 58. In lines 1-2 a SCUA requests for RA identities. Line 3 forwards the SCUA AID to an identified RA; in lines 4-9, MAC addresses are resolved into SA AIDs; in lines 10-15, download requests are forwarded to SAs and the different copies of the same file are returned. A file is then returned to a user in line 16. In addition to URTT, DRTT and UTAT; the $U_T1 \approx D_T1$ value was recorded to test the transfer rates of uploads and downloads between SCUA and RA.

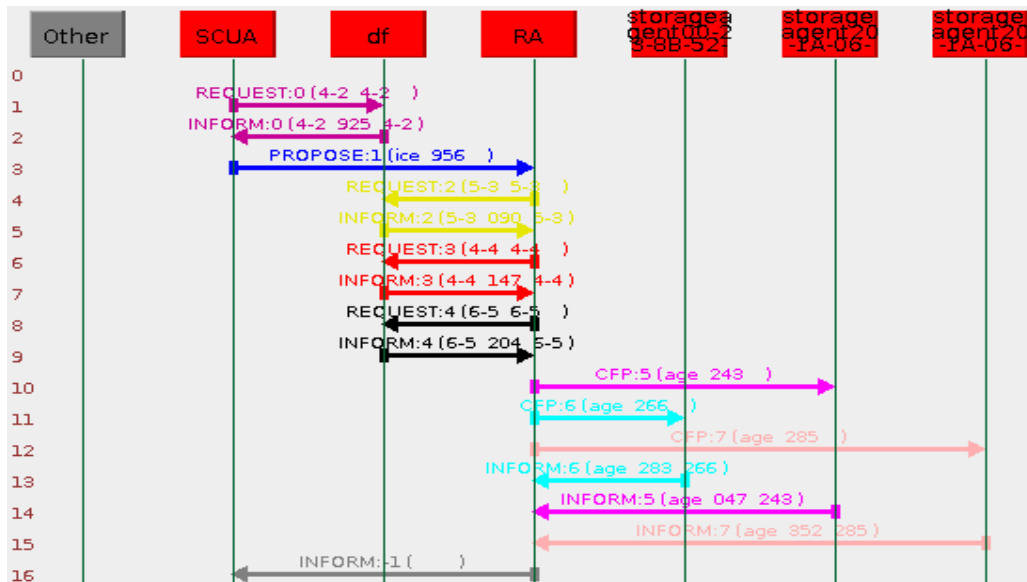


Figure 58: Download Interactions

8.2.3.1 DM U_T1 Analysis

The mean transfer times (U_T1) against file size plot is shown in Figure 59. The 55 MB file size was determined as the largest file category supported in the DM file-system before RAs broke at approximately 60 MB. A sample RA error log is shown in Figure 60. The research hence defined 55 MB as the maximum file size supported on all storage services implemented.

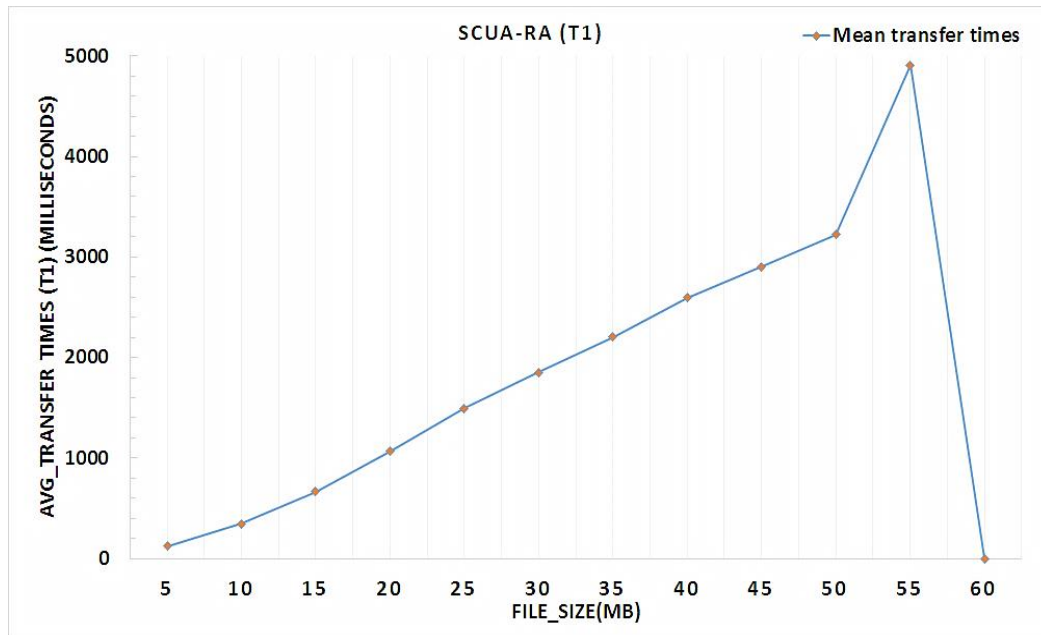


Figure 59: DM_U_T1

```
INFO: Adding node <Container-3> to the platform
Jun 11, 2015 2:11:28 AM jade.core.messaging.MessagingService clearCachedSlice
INFO: Clearing cache
Jun 11, 2015 2:11:28 AM jade.core.PlatformManagerImpl$1 nodeAdded
INFO: --- Node <Container-3> ALIVE ---
the file size : 61440
*** Uncaught Exception for agent RA ***
ignoreERROR: Agent RA died without being properly terminated !!!
State was 2
```

Figure 60: RA Error Log

A simple linear regression analysis was modelled to determine the relationship between U_T1 and file size (in MB). The `lm()` function in R performed the initial step in the regression as in Figure 61.


```

> results = lm(testdata$DMUT1~testdata$file)
> results

Call:
lm(formula = testdata$DMUT1 ~ testdata$file)

Coefficients:
(Intercept)  testdata$file
      180.88         49.26

```

Figure 61: Regression Analysis

From the output, the least squares regression takes the form of: $U_T1 = 180.88 + 49.26$ (filesize). The model states that for a 1MB increment in file size; the U_T1 increase by a factor of 49.46 milliseconds. A regression line superimposed on the data's scatter plot using the `abline()` function observed the graph shown below (Figure 62):

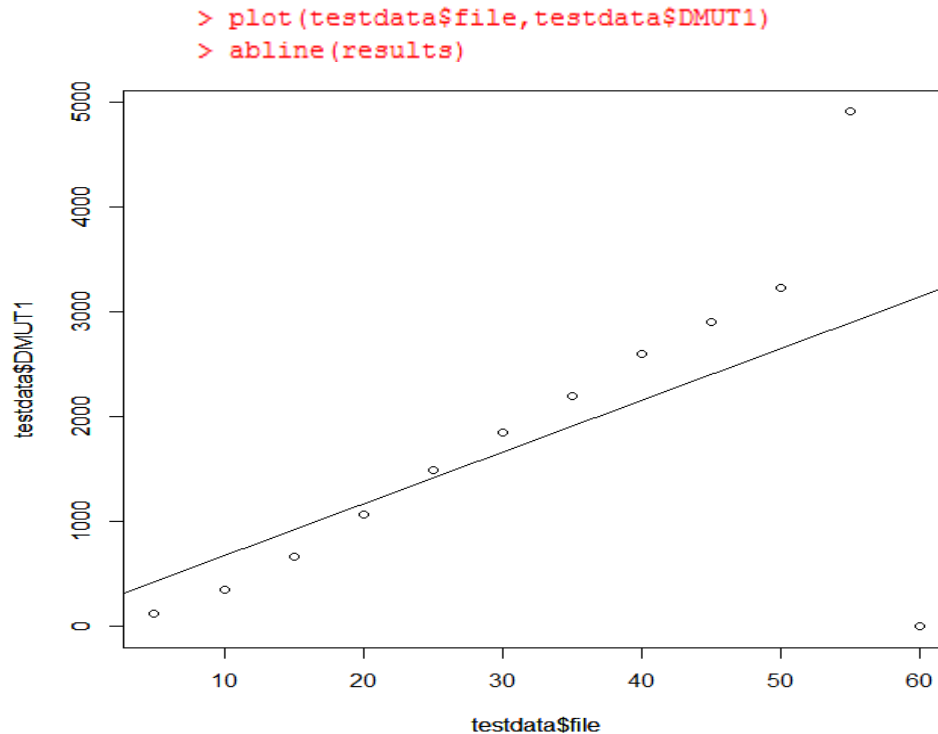


Figure 62: DM_U_T1 vs. File size Plot

There appeared to be a linear relationship between the variables from the scatter plot above. The relationship between the variables was quantified by computing the covariance (in Figure 63):

```

> cov(testdata)
      file  DMUT1
file  325.00 16008.52
DMUT1 16008.52 2161235.21

```

Figure 63: Covariance

It was observed that the file size and U_T1 have variances of 325.00 and 2161235.21 respectively. The covariance of 16008.52 between the variables indicated a positive linear relationship.

		filesizeMB	DM_UT1
filesizeMB	Pearson Correlation	1	.604*
	Sig. (2-tailed)		.038
	N	12	12
DM_UT1	Pearson Correlation	.604*	1
	Sig. (2-tailed)	.038	
	N	12	12

Figure 64: Pearson Coefficient

The Pearson coefficient (in Figure 64) concluded that there is an above average positive correlation between file size and U_T1 equal to 0.604. The coefficient of determination (r^2) which indicates how well the data fit the statistical model hence highlighted a 36.5 % ($r^2 \times 100$) variability in U_T1 explained by file size increments. The higher the r^2 value, the better the model fits the data.

```
> summary(results)

Call:
lm(formula = testdata$DMUT1 ~ testdata$file)

Residuals:
    Min       1Q   Median       3Q      Max
-3136.3  -268.5   138.7   460.7  2016.2

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   180.88     756.28   0.239   0.8158
testdata$file    49.26      20.55   2.397   0.0375 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1229 on 10 degrees of freedom
Multiple R-squared:  0.3649,    Adjusted R-squared:  0.3013
F-statistic: 5.744 on 1 and 10 DF,  p-value: 0.03752
```

Figure 65: Significance Test

The summary () function was executed to test whether the slope of the regression is zero. A zero value deems the model useless. From Figure 65, since the p-value (0.03752) is much less than 0.05, the model rejected the null hypothesis of no linear relationship between U_T1 and file size. At 0.05 significance level there was sufficient evidence to conclude that the U_T1 have a positive contribution to the upload service response time.

8.2.3.2 DM UTAT Analysis

The UTATs for different file uploads are plotted together with TAT pings in Figure 66. The graph estimates how heterogeneous network connectivity affects the performance of distributed file-systems. The UTATs for 40 MB and 45 MB file categories indicate a marked deviation from expected values when TAT pings recorded minimum values. The plot illustrated that shared node network connectivity fluctuations can affect the UTAT hence the overall file-system performance.

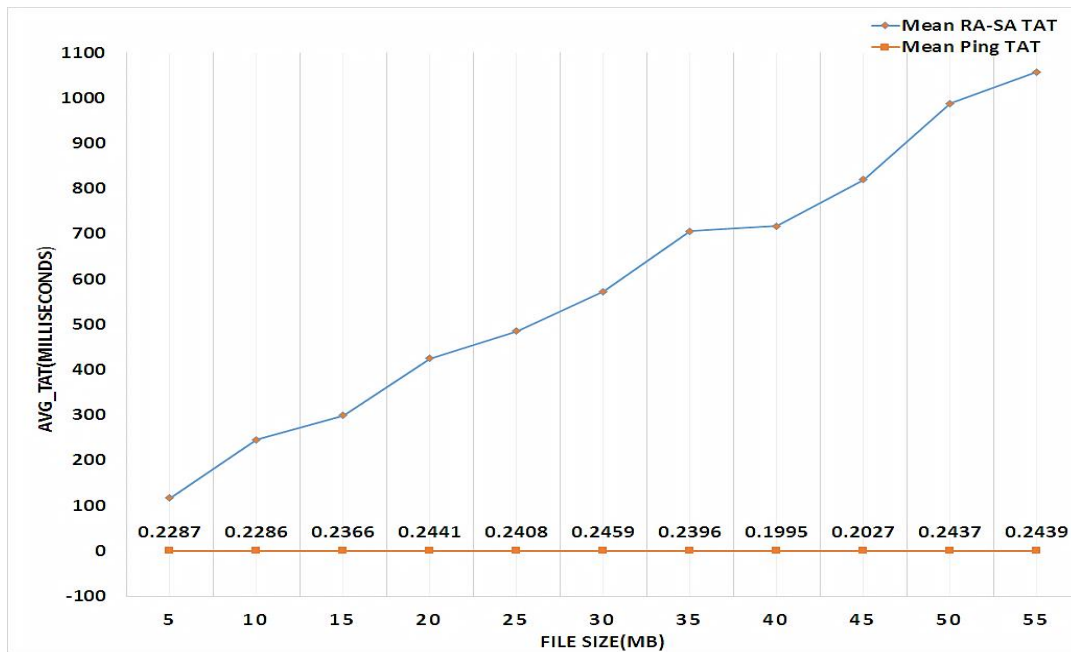


Figure 66: DM_U_TAT

```
> cov(testdata)
      filesize  DMUTAT
filesize 275.0 5021.50
DMUTAT   5021.5 92520.74
```

Figure 67: Covariance

A closer look at the covariance coefficient (in Figure 67) and scatter plot (in Figure 68) with fitted regression line revealed a positive linear scaling in performance for the DM technique.

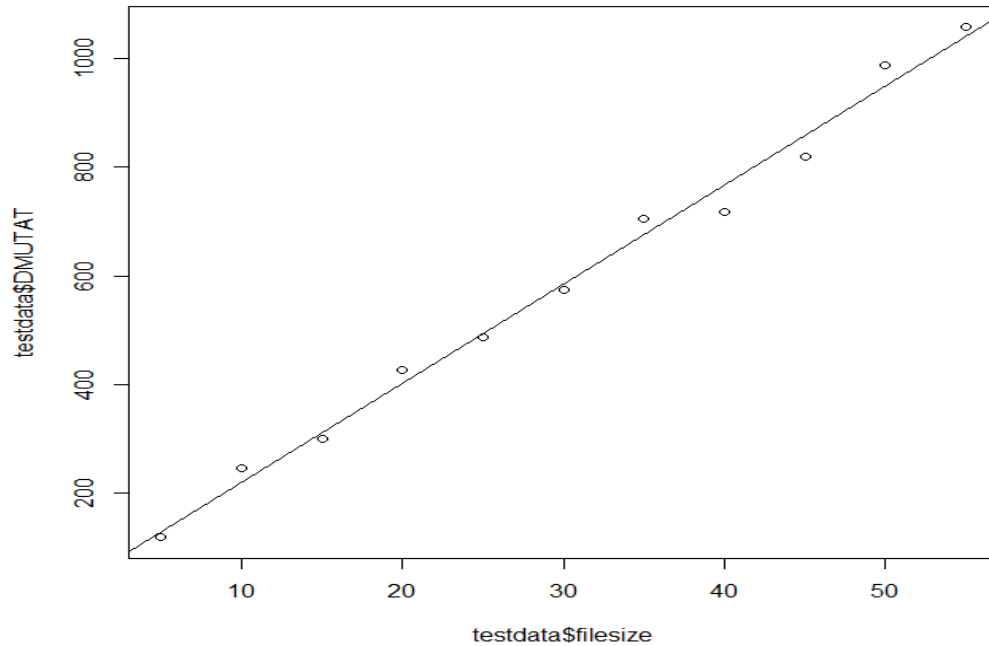


Figure 68: DM_U_TAT vs. File-size Plot

A strong linear correlation (0.996) between the file size and UTAT in Table 12 clarified a 99.2 % variability in UTAT explained by file size increments.

		filesizeMB	DM_U_TAT
filesizeMB	Pearson Correlation	1	.996**
	Sig. (2-tailed)		.000
	N	11	11
DM_U_TAT	Pearson Correlation	.996**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 12: DM_U_TAT Correlations

Based on the UTAT intercept and file size estimates in Figure 69; the linear regression equation indicates that for every additional 1MB in file size the UTAT increase by 18.26 milliseconds. At $\alpha = 0.05$ level, there is sufficient evidence to accept the hypothesis of linear relationship between UTAT and file size for p-value= $(1.577e-10) < 0.05$.

```

Call:
lm(formula = testdata$DMUTAT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-50.295 -12.138  -8.181   24.455   37.605

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)    37.3234    19.6163   1.903   0.0895 .
testdata$filesize 18.2600     0.5785  31.567 1.58e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 30.33 on 9 degrees of freedom
Multiple R-squared:  0.991,    Adjusted R-squared:  0.9901
F-statistic: 996.5 on 1 and 9 DF,  p-value: 1.577e-10

```

Figure 69: Test for Significance

8.2.3.3 DM U_RTT Analysis

A clear picture of the URTT and file size correlation is shown in Figure 70. As expected, the larger the file uploaded, the higher the URTT. A fitted regression model visually confirmed a linear relationship in the data points.

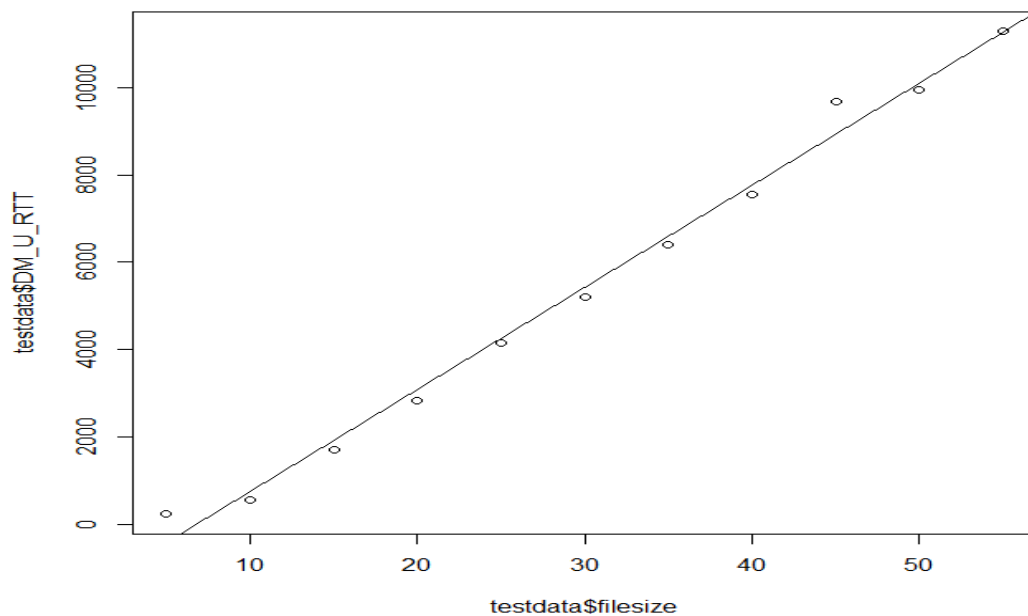


Figure 70: DM U_RTT vs. File size Plot

The covariance (in Figure 71) on the fitted data described a positive linear relationship of 64223.83 between the URTT and file-size.

```
> cov(testdata)
      filesize  DM_U_RTT
filesize  275.00  64223.83
DM U RTT  64223.83 15127967.33
```

Figure 71: Covariance test

A strong linear correlation of 0.996 estimated a 99.2 % variability in the URTT explained by simulated files.

		filesizeMB	DM_U_RTT
filesizeMB	Pearson Correlation	1	.996**
	Sig. (2-tailed)		.000
	N	11	11
DM_U_RTT	Pearson Correlation	.996**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 13: Pearson Coefficient

The following were concluded from the model significance test (in Figure 72): (1) A 1 MB file size increase results in a 233.541 milliseconds increase in URTT and (2) The model rejects the null hypothesis of no linear relationship based on a non-zero p-value ($1.27e-10 < 0.05$). The statistical model hence predicted an upload service with linear scaling in performance.

```
Call:
lm(formula = testdata$DM_U_RTT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-247.51 -204.52 -172.72  -29.41   765.46

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   -1589.752    244.878   -6.492 0.000112 ***
testdata$filesize  233.541      7.221  32.342 1.27e-10 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 378.7 on 9 degrees of freedom
Multiple R-squared:  0.9915,    Adjusted R-squared:  0.9905
F-statistic: 1046 on 1 and 9 DF,  p-value: 1.27e-10
```

Figure 72: Test for Significance

8.2.3.4 DM DRTT Analysis

The data points for the download service in Figure 73 shows a linear relationship. The fitted points are closer to the regression line up to the 30 MB file size and tend to be a bit spaced from the line of best fit for file sizes greater.

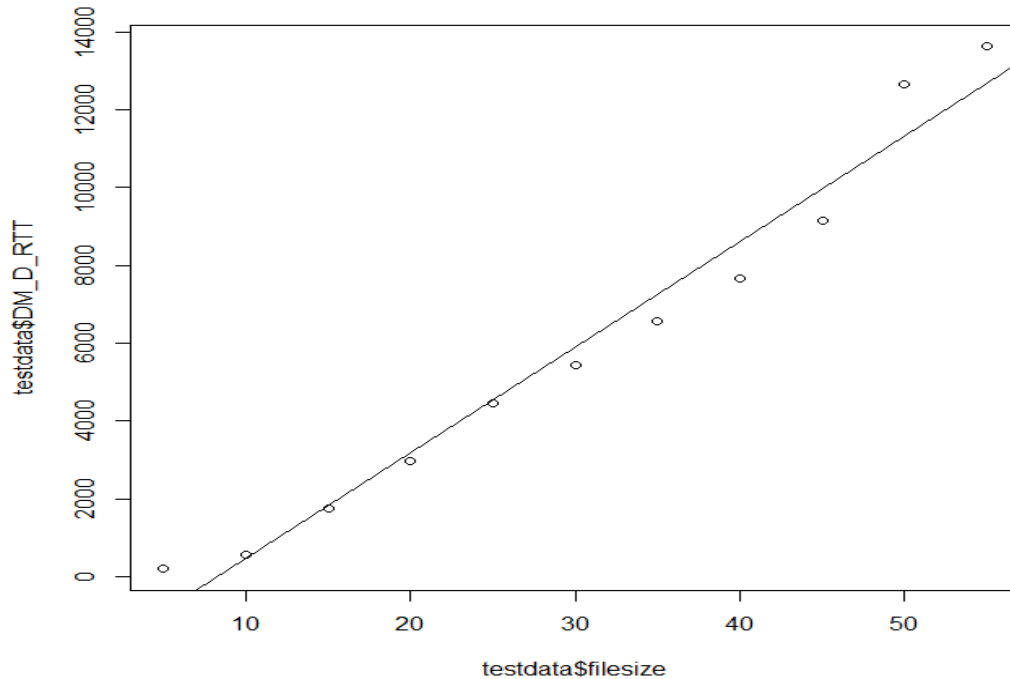


Figure 73: DM_D_RTT vs. File-size Plot

An analysis on the covariance shows a larger positive linear relationship (74599.66) compared to other associated covariance from related sample populations (U_T1, UTAT and URTT).

```
> cov(testdata)
      filesize  DM_D_RTT
filesize  275.00  74599.66
DM_D_RTT 74599.66 20856456.15
```

Figure 74: Covariance

The coefficient (in Table 14); close to 1 (one) illustrates a strong linear relationship. The file size explain 97 % change in the download response time. However, since further inferences cannot be made on the relationships and the linear model using Pearson's coefficient; a significance test was conducted as in Figure 75.

		filesizeMB	DM_D_RTT
filesizeMB	Pearson Correlation	1	.985**
	Sig. (2-tailed)		.000
	N	11	11
DM_D_RTT	Pearson Correlation	.985**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 14: Pearson Coefficient

From the output, the download response time increase by 271.27 milliseconds for a single unit increase in file size. The model accept the alternative hypothesis of linear relationship in DRTT and file size for $p = 3.518e-08$. Therefore the download service's performance scales to increase in file size.

```
Call:
lm(formula = testdata$DM_D_RTT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-960.9 -597.4 -102.4   515.6 1321.2

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2229.05     536.60  -4.154  0.00247 **
testdata$filesize   271.27       15.82  17.144 3.52e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 829.8 on 9 degrees of freedom
Multiple R-squared:  0.9703,    Adjusted R-squared:  0.967
F-statistic: 293.9 on 1 and 9 DF,  p-value: 3.518e-08
```

Figure 75: Significant Test

8.2.4 C-AP File-System Evaluation

The evaluations described in this step detail the storage system's performance in offering specific services. The section evaluates the C-AP technique performance benefits in contrast to the DM approach using linear regression analysis on collected data.

8.2.4.1 C-AP U_TAT Analysis

The fitted regression model (in Figure 76) shows a linear relationship in the population sample. This correlation hence assumes a linear scaling in the C-AP technique's performance.

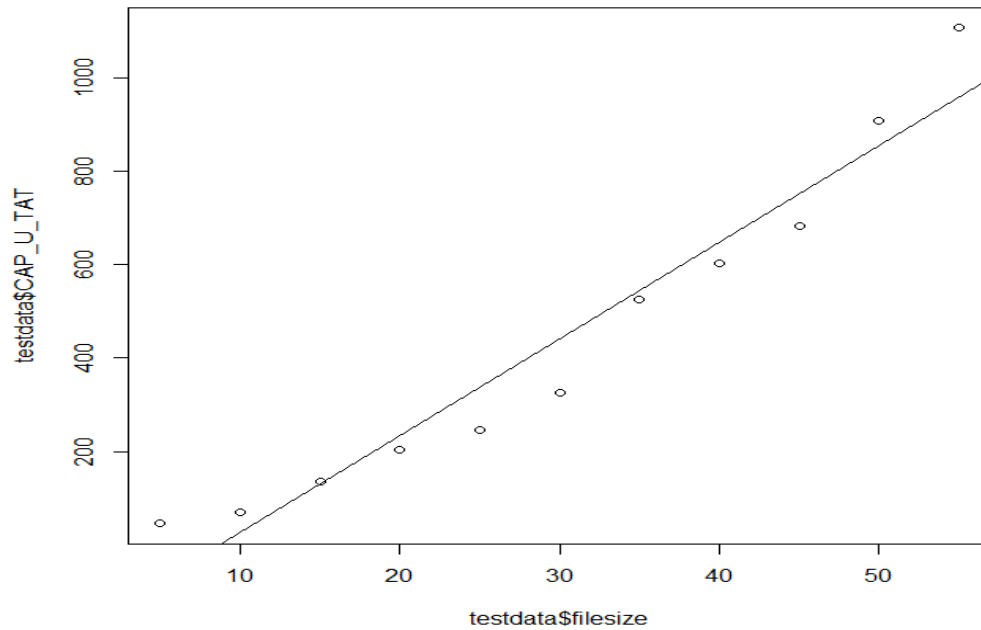


Figure 76: C-AP U_TAT vs. File-size Plot

Observing the covariance output (in Figure 77); it was concluded that the UTAT and file size have a positive linear correlations of 5695.6.

```
> cov(testdata)
      filesize CAP_U_TAT
filesize    275.0    5695.6
CAP_U_TAT   5695.6   125094.8
```

Figure 77: Covariance

A further analysis on the correlation coefficient (in Table 15) specified a strong linear correlation in the UTAT and file size with 94.2 % of the variability in the UTAT explained by file size.

		filesizeMB	CAP_U_TAT
filesizeMB	Pearson Correlation	1	.971**
	Sig. (2-tailed)		.000
	N	11	11
CAP_U_TAT	Pearson Correlation	.971**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 15: Pearson Coefficient

The test shown in Figure 78 describes a regression equation of the form: $UTAT = -179.932 + 20.711 (\text{filesize})$. For a 1MB increase in file size, the UTAT change is negligible (-159.221). The change is significantly small compared to the UTAT coefficient (18.26 milliseconds) for the DM

file-system inferred in Section 8.2.3.2. This implies that C-AP considerably increase performance in the C-AP file-system compared to the DM file-system's mirroring approach.

```
Call:
lm(formula = testdata$CAP_U_TAT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-115.54  -56.90  -19.03   47.93  148.34

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -179.932     57.565  -3.126   0.0122 *
testdata$filesize   20.711       1.697  12.201 6.68e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 89.02 on 9 degrees of freedom
Multiple R-squared:  0.943,    Adjusted R-squared:  0.9367
F-statistic: 148.9 on 1 and 9 DF,  p-value: 6.683e-07
```

Figure 78: Significance Test

At $\alpha = 0.05$ level, the null hypothesis is rejected for the sample population at p-value = 6.683e-07. Hence the TAT transition in the upload service scales linearly to an increase in file size.

8.2.4.2 C-AP URTT Analysis

The UTAT from the previous section projected a significant reduction in the URTT for the C-AP file-system upload service. The URTT and file size graph was computed as in Figure 80. From a high level, a linear relationship is confirmed from the plot and the covariance (in Figure 79).

```
> cov(testdata)
           filesize CAP_U_RTT
filesize    275.00  13422.65
CAP_U_RTT 13422.65 698603.64
```

Figure 79: Covariance

From the correlation test (in Table 16); a strong linear correlation coefficient close to one (0.968) was observed, explaining a 93.7 % variability in the URTT influenced by the file size.

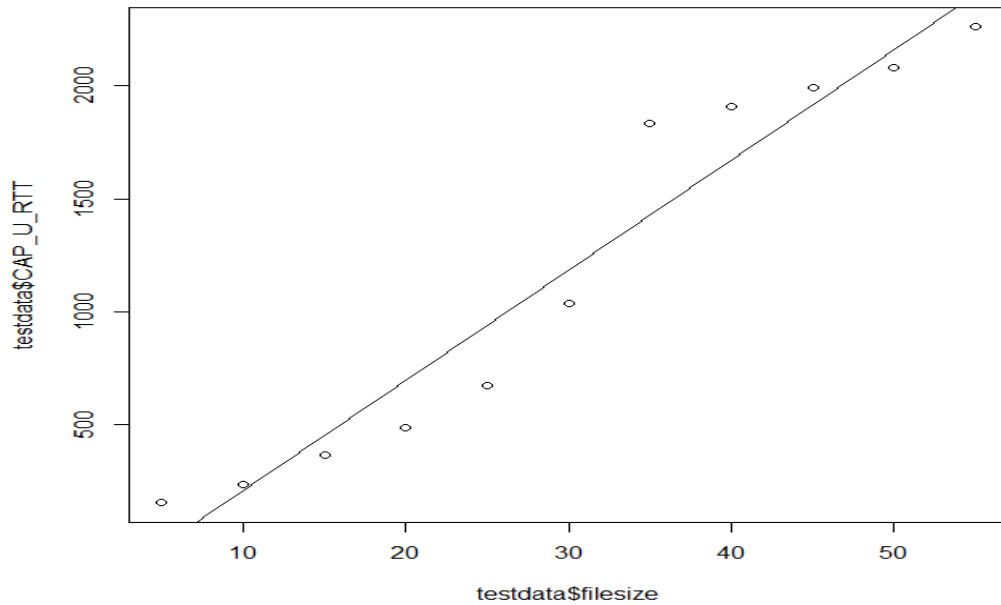


Figure 80: C-AP_U_RTT vs. File size Plot

		filesizeMB	CAP_U_RTT
filesizeMB	Pearson Correlation	1	.968**
	Sig. (2-tailed)		.000
	N	11	11
CAP_U_RTT	Pearson Correlation	.968**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 16: Pearson Coefficient

A summary on the regression model in Figure 81 shows a small -232.13 milliseconds increase in the URTT caused by file size increase. The increment in URTT is considerably small compared to a similar metric (233.541 milliseconds in Section 8.2.3.3) computed for the DM file-system. From the analysis it is concluded that the C-AP file-system upload service response time is optimal in offering upload services.

At the 0.05 significance level the linear regression model concluded a linear relationship in upload service's scaling to increase in file size uploaded and performance of the C-AP file system at p-value = 9.908e-07.

```

Call:
lm(formula = testdata$CAP_U_RTT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-269.40 -144.57  -78.99  133.68  405.26

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    -280.94     142.09  -1.977   0.0794 .
testdata$filesize    48.81       4.19  11.649 9.91e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 219.7 on 9 degrees of freedom
Multiple R-squared:  0.9378,    Adjusted R-squared:  0.9309
F-statistic: 135.7 on 1 and 9 DF,  p-value: 9.908e-07

```

Figure 81: Significance Test

8.2.4.3 C-AP DRTT Analysis

The download service's plotted points (in Figure 82) are scattered about the regression line which resemble a linear relationship. A positive linear relationship is confirmed in Figure 83.

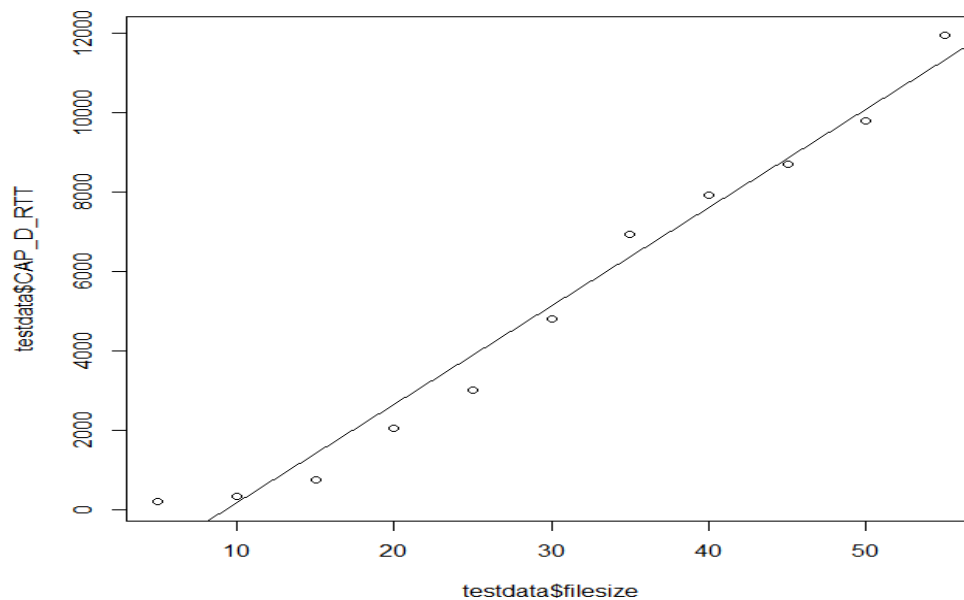


Figure 82: C-AP DRTT vs. File Size Plot

```

> cov(testdata)
              filesize CAP_D_RTT
filesize      275.0    68029.5
CAP_D_RTT    68029.5 17245794.5

```

Figure 83: Covariance

An inference on the correlation approved a linear correlation in the test data. 97.6 % of the trend in DRTT is accounted for by the explanatory variable with the remainder explained by other factors.

		filesizeMB	CAP_D_RTT
filesizeMB	Pearson Correlation	1	.988**
	Sig. (2-tailed)		.000
	N	11	11
CAP_D_RTT	Pearson Correlation	.988**	1
	Sig. (2-tailed)	.000	
	N	11	11

Table 17: Pearson Coefficient

Based on the p-value=1.384e-08 (in Figure 84); the statistical analysis rejected the null hypothesis of no linear relationship between the DRTT and file sizes.

```
Call:
lm(formula = testdata$CAP_D_RTT ~ testdata$filesize)

Residuals:
    Min       1Q   Median       3Q      Max
-881.0 -469.5 -142.0  441.1 1249.9

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  -2289.95     440.00  -5.204 0.000561 ***
testdata$filesize   247.38       12.97  19.066 1.38e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 680.4 on 9 degrees of freedom
Multiple R-squared:  0.9758,    Adjusted R-squared:  0.9732
F-statistic: 363.5 on 1 and 9 DF,  p-value: 1.384e-08
```

Figure 84: Significance Test

From the regression equation (DRTT = -2289.95 + 247.89 (filesize)) formulated; it is established that a single unit increase in file size effects an insignificant (-2041.62) increase in the response time of the download service. This implies a close to constant response time in all file download categories for file chunks appended. Compared to the DM file-system download service coefficient rise (271.27 milliseconds in Section 8.2.3.4); the linear regression model accepts the C-AP file-system as optimal in offering download services.

8.2.5 C-AP vs. DM File-System

The linear regression analysis in Section 8.2.4 proved the C-AP file-system ideal for offering storage services compared to the DM file-system. The analysis in this section complements the linear regression deductions by comparing the C-AP and DM file-systems side by side. The C-AP file-system outperforms the DM file-system upload service in the same experimental environment configurations (from Figure 85). The contributing factor to the performance gap in addition to the storage methods was attributed to the JADE internal environment (detailed in Section 9.4).

For DM file-system experiments; the argument (*-jade_core_messaging_MessageManager_max_queue_size<bytesize>*) was defined on initialising the platforms to support file simulations greater than the default 10 MB internal queue threshold [130]. From the graph, it's likely that the measure impacted negatively on the DM file-system performance.

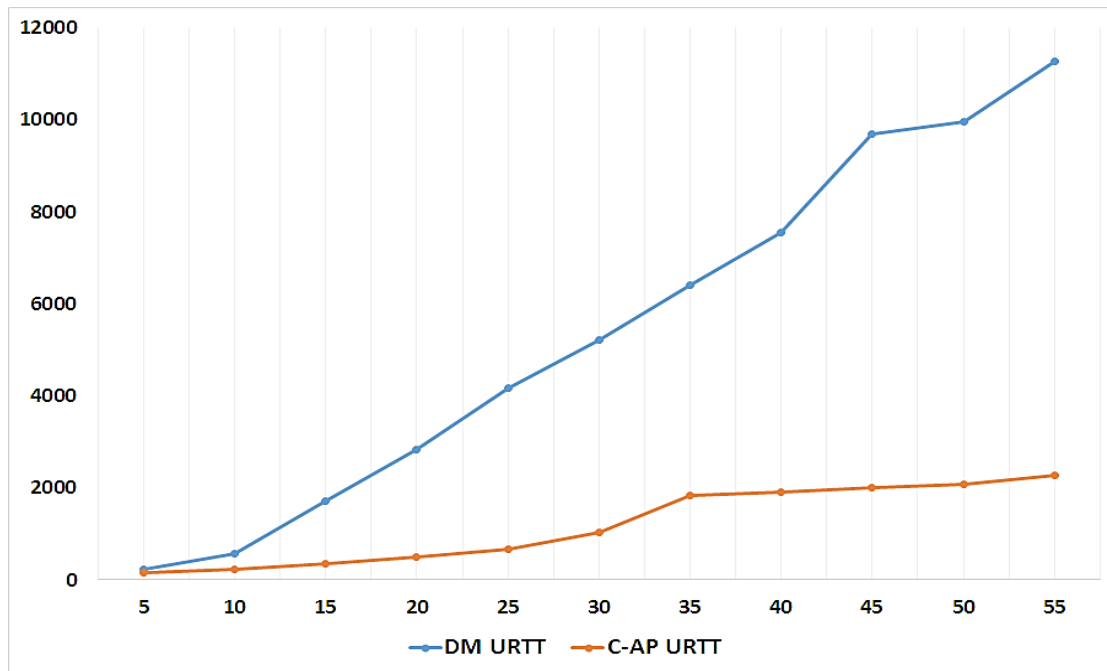


Figure 85: DM & C-AP Mean URTT

The C-AP performs well than the DM method as evidenced in the curve lag shown in Figure 86. In reference to JADE MAS, chunking optimised the default internal queue size (10MB) thereby incorporating high scalability and performance in message passing.

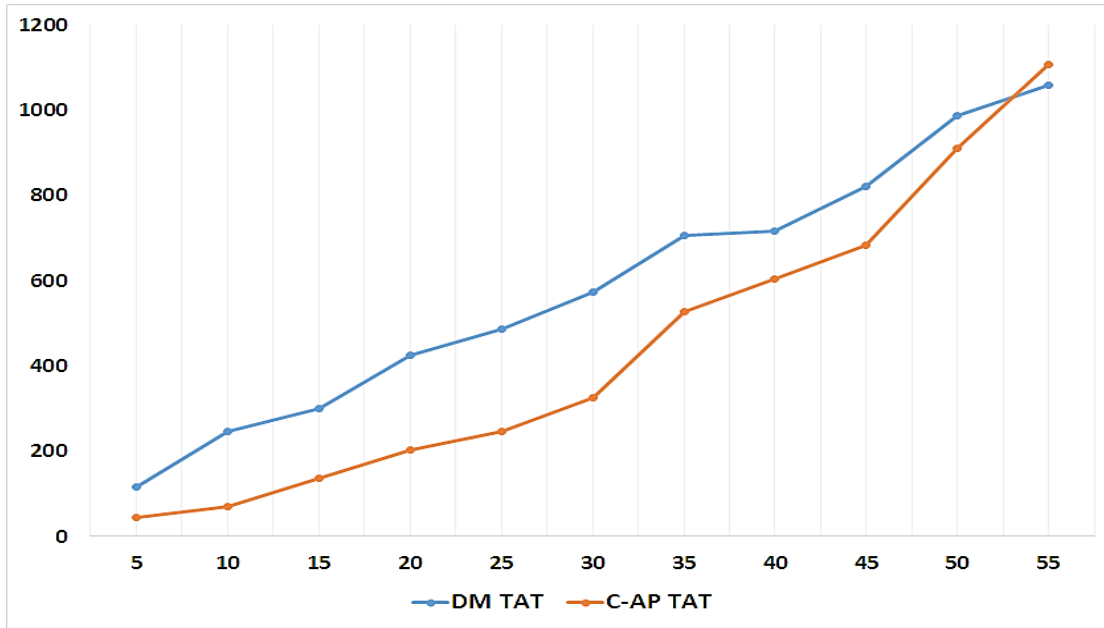


Figure 86: DM & C-AP Mean U_TATs

From Figure 87, it is observed that the C-AP file-system's DRTT curve delays behind the DM file-system's DRTT up to the 30 MB file size and continues with the trend from the 45 MB file-size. The pattern broke for the 35 MB and 40 MB file size range which requires 4 nodes for chunk appends. Node delays in returning chunk feedbacks is the likely cause since C-AP DRTT depends on all chunk responses being returned for a download to be successful. The remark is however trivial to disprove the overall optimality demonstrated by the C-AP download service.

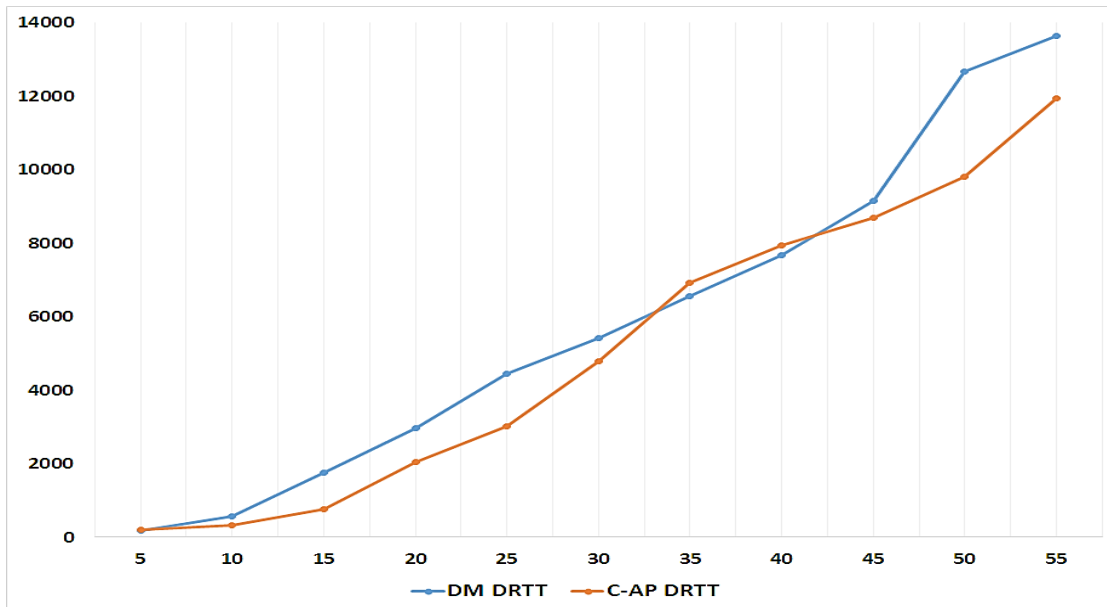


Figure 87: DM & C-AP Mean DRTT

From the tests (in Section 8.2.4), the C-AP file-system operations were desirable based on the linear regression analysis model. In contexts where shared nodes have reliable connectivity; the C-AP service platforms are effective since the networking guarantees chunk feedback return. This is endorsed by GFS setting in which the C-AP technique is originally proposed [54]. In settings with erratic connectivity frameworks the DM file-system may be optimal as redundantly stored file replicas are identical and can be accessed from any available host. Based on the proof-of-concepts evaluations discussed, the C-AP file-system was concluded as technically sufficient independent of JADE environment performance concerns.

8.3 Security

This research introduced security as crucial in open distributed MAS. The JADE-S add-on [131] provided a means for integrating message integrity and confidentiality through signature and encryption guarantee. Selectively, signatures ensure message integrity and identity of the message originator. Encryption on the other hand enables for confidentiality by protecting message data from eavesdropping. Participant users in the system simply request a message to be signed or check whether a received message has been signed to extend the functionalities. Concerning JADE-S, support in JADE is implemented as a set of services [131]:

- **jade.core.security.SecurityService:** The service provide crypto engines and agent key pairs management functionality in addition to authentication;
- **jade.core.security.permission.PermissionService:** The service checks if agent actions performed (e.g. sending messages, migration to other containers) are actually authorized;
- **jade.core.security.signature.SignatureService:** The service signs and validates incoming signed messages when requested by the sender and receiver respectively;
- **jade.core.security.encryption.EncryptionService:** The service encrypts messages when defined by the sender and decrypt incoming encrypted messages.

In order to activate JADE-S services that match the security requirements proposed for the MAOG system; the SecurityService, SignatureService and EncyprtionService arguments where parsed in agent runtime initializations.

8.4 Conclusion

The evaluation of compute and storage components investigated on the functional and technical adequacy of the MAOG services. The MA component's CNP identified an optimum platform to assign a computation resulting in reasonable turnaround times in application processing. The fluctuation in these times observed MA code relocations which validated the rule reasoning functionality encapsulated. The C-AP file-system was concluded as optimum in offering storage services due to its optimisation of JADE MAS default configurations. However, the DM file-system scalability was affected greatly by adopting the mirroring approach to file storage with reference to the defined measurement criteria.

Chapter 9

9 Conclusion and Future Work

This chapter reports on the research goal as presented in chapter 1. The research summary, findings, contributions, limitations and future work prospects are discussed. A research paper published in this work is underlined at the end.

9.1 Summary

OGs join compute and storage resources matching the capabilities of high performance and file hosting systems. The related projects solve problems in compute intensive and data storage setups utilising volunteered and idle shared machines. Considering the nature of resource providers and connectivity, it's often complex to join nodes and prioritise resource providers in modern distributed applications. It was hence valuable to explore context aware strategies in a specific MAOG service.

Agent technologies provide means of natural understanding, designing and implementation of autonomous distributed applications. Scholarly views consider proactivity and responsiveness as key for agent technologies to achieve their goals and objectives. A review justified MAS as autonomous codes that communicate using unique communication languages. To confirm the MAS approach, commitment in understanding the pragmatic areas of agent development such as analysis and design, micro (agent) level and macro (society) level were considered. Integrating agent negotiation and reasoning reduce transaction costs in resource identification and allocation. To this end automation enabled for proactivity and reactivity in the MAOG services analysed.

9.2 Findings

As stated in the introductory chapter, the main research aim was to investigate the use of MAS technology in Opportunistic Grid Computing. To address the research questions highlighted, the onion metaphor transformed research questions into this research project. The high level processes explained by the onion layers in addressing the questions are summarized further on:

1. Can low-cost commodity computers in ICT contexts be exploited for an OG setup?

An ICT infrastructure in Dwesa founded on low-cost machine resources was recommended as theoretically feasible from the SLL connectivity framework reviewed in Section 5.1. The research activities detailed in Section 2.1.3.1 evaluated the nature of resources, resource providers and domain specific concerns (e.g. unreliable power, need for increased redundancy) that may affect an OG platform solution.

2. What is the most appropriate distributed computing design and implementation for an ICT4D OG?

The research question was addressed in Chapter 3. Section 3.1.1 accepted VC as a significant approach based on the nature of resource providers and resources identified. The Cloud and Grid computing literature foundations were then reviewed in Sections 3.2 and 3.3 to justify the research's computing perspective in distributed computing. Section 3.2 established OGs as different from grid computing in view of resource types, connectivity, dedication and trust characteristics. Section 3.3.2 discussions on the business, programming and virtualisation technologies distinguished the cloud model from all distributed computing approaches considered. Research questions 1 and 2 confirmed the distributed computing perspective and the deployment context of the MAOG system.

3. Is a MAS solution a feasible technology for implementation of the platform and does it provide the necessary functional adequacy?

Chapter 4 presented the agent technology concepts in terms of standard compliance; communication strategies and protocols; coordination and development pitfalls. A comparative analysis of MAS development platforms in Section 4.7 motivated the JADE MAS platform's adoption for the MAOG development using the methodology assumed in Section 2.1.4.4. By incorporating the mobile code paradigm (in Section 4.10); the MA compute, DM file-system and the C-AP file-system designed and implemented in Chapters 6 and 7 formalised the functional adequacy of the services.

4. What is the most suitable MAS analysis and design methodology to be utilized in the implementation of such a MAS system?

Section 2.1.4 compared various MAS methodologies and adopted a methodology by Nikraz et al for the system development stage. The methodology includes people and legacy systems through the support for the top-down and bottom-up approaches.

5. How can agents reason about their execution environment to adapt the system to dynamic environment changes? And what is the most suitable rule engine to implement rule reasoning in MAS?

In addition to proactive, responsive and social characteristics recognised in MAS (detailed in Section 4.1), Section 4.11 introduced knowledge presentation and reasoning concepts that enable agents to reason on supplied knowledge using symbolic rules. Jess inference engine was selected for rule based reasoning due to integration issues encountered with Drools (Issues in Section 7.1.5.1).

In this dissertation, a multi-agent grid solution was realised. The platform services were tested in experimental set-ups. To adapt the platforms to unreliable settings, the MA component validated the effectiveness of CNP and rule reasoning. The storage component made up of C-AP and DM file-systems was introduced for storage oriented services.

The tests performed on the MA compute show the effectiveness of CNP in intelligent resource identification and allocation to offer lower response times in computations. Rule reasoning expressed in Jess presented a means of interpreting execution environment changes to prioritise resource provider activity. The functionality solves possible conflicts in which deployed workflows would have to be resubmitted on node recalls or failures.

For an increase in file size in related services; the DM file-system's response and turnaround times demonstrated poor scalability and performance. File storage optimisations through GFS's C-AP recommended the C-AP file-system services.

9.3 Contributions

The main contributions of this research are presented here:

1. A model for harvesting networked resources founded on public resource computing
Section 3.1 motivated the research's Opportunistic grid model considering the nature of resource providers. An account of available projects harnessing compute and storage resources for scientific research was highlighted in Section 3.1.2 to motivate the feasibility of the model in the research context.

2. Validation of the CNP for resource identification and allocation in an open market structure based on bids and contract allocation

The CNP negotiation mechanism in Section 4.5.1 was effective in identifying readily available compute resources that can process a compute application using donated processor resources. The results from Section 8.1.1 validated the effectiveness of the market structure in enabling reduced turnaround times in application processing.

3. Demonstration of rule based reasoning in MAS to incorporate context awareness in distributed applications using Jess

The Jess rule engine was utilised for reasoning in the MA compute system. The most crucial functionality identified in the design phase (discussed in Section 6.1) prioritised resource provider activity on donated nodes. In this scenario the inference engine instance encapsulated in the processing code was effective in detecting user activity on a processing node and relocating computations. The detailed technical details in handling the proposed services are documented in Section 7.1.4.1.

4. Confirming the efficiency of MAs in handling compute services

The MA paradigm's ability to encapsulate applications and processing using idle CPU was confirmed in Section 6.1. The developed MA processing component processed a test application in Appendix A.1.1.

5. Implementation of DM and C-AP file-systems and their evaluation to identify the best approach for an ICT4D setting

The DM and C-AP file-systems were designed and implemented in Sections 6.2 and 7.2 respectively. From the evaluations in Sections 8.2.3, 8.2.4 and 8.2.5 the C-AP file-systems was established as a best approach for the ICT4D background. The C-AP file-system's optimisation of the 10 MB JADE MAS queue size derived from the GFS chunking approach (discussed in 3.4.1) was stated as a crucial design method.

9.4 Limitations

In JADE, ACL messages are inserted into an OutBox (internal queue) before forwarding. Dedicated threads extract these messages from the OutBox and deliver them to defined receivers [130]. The OutBox size hence depends the number of messages to be forwarded and the content encapsulated. The default queue threshold in JADE is defined as 10MB [130]. When an OutBox memory exceeds 10MB generally a multi-agent system becomes slow. The observation was particularly important in simulating different file sizes to developed file-systems in an experimental environment. In the tests, *jade_core_messaging_MessageManager_maxqueue size* *<byte size>* argument was defined to increase the threshold on running agents in the JADE containers. Considering the messaging feature discussed, JADE performance might not scale to storage requests involving large files.

9.5 Future Work

The research described in this dissertation has formalised the potential of ICTs in socio-economic development through sharing of expertise. Developments in distributed computing are creating new insights for extending the MAOG platform functionality. By separating the deployment context, confirmed ideas can support many distributed computing models such as Clouds. The described research also identified promising guidelines for future exploration:

6. **File-system operations:** Criticism might be levelled against the lack of completeness in actions expected for a file hosting service. The MAS based file-systems designed, implemented and evaluated included the download and upload as key actions. Further integration will include pending operations (i.e. file deletion and platform refresh).
7. **JADE's internal queue size:** The evaluation of DM file-system services modified the Outbox queue sizes to accommodate file uploads and downloads greater than the JADE default. The modification could have presented some anomalies in the disk mirroring technique to storage. It would be interesting to determine the coefficient of error combined in measurements and their effect on the overall results.
8. **Evaluation of MA and REV approaches:** Future research will design and implement a MAS based REV compute component. The component will be evaluated with the MA compute developed.

9. **Permissions and authentication:** Future development will consider authentication and permissions in the developed services.

9.6 Publications

The following journal was published in this research:

10. R. T. Muranganwa and M. Thinyane, “Design of a multi-agent opportunistic grid computing platform,” *Multiagent Grid Syst.*, no. 10, pp. 199–212, 2014.

Bibliography

- [1] M. Rahimpour, "Computer Assisted Language Learning(CALL)," *International Journal of Instructional Technology and Distance Learning*, vol. 8, no. 1, pp. 3–9, 2011.
- [2] S. Marshall and T. Wal, "Collaboration as a critical success factor in using ICT for capacity building and community development," *International Journal of Education and Development using Information and Communication Technology*, vol. 1, no. I, pp. 2–4, 2005.
- [3] S. Bailur, "The Complexities of Community Participation in ICT for Development Projects:The Case of 'Our Voices,'" in *International Conference on Social Implications of Computers in Developing Countries*, 2007, pp. 1–17.
- [4] D. Gichoya, "Factors Affecting the Successful Implementation of ICT Projects in Government," *The Electronic Journal of e-Government*, vol. 3, no. 4, pp. 175–184, 2005.
- [5] M. Siphiosami, N. Mamba, and N. Isabirye, "Information Technology for Development A Framework to Guide Development Through ICTs in Rural Areas in South Africa," *Information Technology for Development*, pp. 1–16, 2014.
- [6] N. K. Roy, "ICT –Enabled Rural Education in India," *International Journal of Information and Education Technology*, vol. 2, no. 5, pp. 525–529, 2012.
- [7] J. C. Sipior and B. T. Ward, "Bridging the Digital Divide for e-Government inclusion : A United States Case Study," *The Electronic Journal of e-Government*, vol. 3, no. 3, pp. 137–146, 2005.
- [8] S. Yi, D. Kondo, and A. Andrzejak, "Reducing Costs of Spot Instances via Checkpointing in the Amazon Elastic Compute Cloud," in *International Conference on Cloud Computing*, 2010, pp. 236 – 243.
- [9] L. Ponciano and F. Brasileiro, "Assessing Green Strategies in Peer-to-Peer Opportunistic Grids," *Journal of Grid Computing*, vol. 11, no. 1, pp. 129–148, 2013.
- [10] "Siyakhula Living Lab:Project Overview." [Online]. Available: <http://siyakhulall.org/>. [Accessed: 29-Mar-2015].

- [11] L. Dalvit, I. Siebörger, and H. Thinyane, "The expansion of the Siyakhula Living Lab: A holistic perspective," in *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering*, 2012, pp. 228–238.
- [12] S. Marshall and W. Taylor, "Facilitating the use of ICT for community development through collaborative partnerships between universities, governments and communities," *International Journal of Education and Development using ICT*, vol. 1, no. 1, 2005.
- [13] Mark Baker, Ed., "Cluster Computing White Paper," in *IEEE Computer Society Task Force on Cluster Computing (TFCC)*, 2000.
- [14] N. Hritonenko and Y. Yatsenko, "Creative Destruction of Computing Systems :Analysis and Modeling," *Journal of Supercomputing*, vol. 38, no. 2, pp. 143–154, 2006.
- [15] M. Saunders, P. Lewis, and A. Thornhill, "Research Onion," in *Research Methods for Business Students*, Pearson Education, Ed. 2009, pp. 136–162.
- [16] A. Håkansson, "Portal of Research Methods and Methodologies for Research Projects and Degree Projects," in *International Conference on Frontiers in Education: Computer Science and Computer Engineering*, 2013.
- [17] N. J. Salkind, *Exploring research*, 6th ed. Pearson International Edition, 2006.
- [18] M. Myers, *Qualitative research in Business and Management*. SAGE Publication Inc. London, UK., 2009.
- [19] G. Caire, J. Stark, W. Coulier, F. Garijo, J. Gomez, F. Leal, R. Evans, F. Garijo, J. Pavon, E. Vargas, P. Kearney, and P. Massonet, "Agent Oriented Analysis using MESSAGE / UML," in *Lecture Notes in Computer Science*, G. Weiss and P. Ciancarini, Eds. Springer-Verlag, 2001, pp. 119–135.
- [20] Y. Shohan, "Agent Oriented Programming," Stanford University Technical Report STAN-CS-90-1335, Stanford, 1990.
- [21] M. Wooldridge, *An Introduction to Multiagent Systems*. 2002.
- [22] M. Nikraz, G. Caireb, and B. Parisa A., "A Methodology for the Analysis and Design of Multi-Agent Systems using JADE," *International Journal of Computer Systems Science and Engineering*, no. 2, p. 21, 2006.

- [23] M. Wooldridge, N. R. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 285–312, 2000.
- [24] "Tropos Methodology." [Online]. Available: <http://www.troposproject.org>. [Accessed: 28-Feb-2015].
- [25] P. Giorgini, M. Kolp, J. Mylopoulos, and M. Pistore, "The Tropos Methodology: An Overview," in *Methodologies and Software Engineering for Agent Systems*, Kluwer Academic Press, 2003, p. 505.
- [26] D. P. Anderson and G. Fedak, "The Computational and Storage Potential of Volunteer Computing," in *IEEE International Symposium on Cluster Computing and the Grid*, 2006, pp. 73–80.
- [27] I. C. Wu, C. Chen, P. H. Lin, G. C. Huang, L. P. Chen, D. J. Sun, Y. C. Chan, and H. Y. Tsou, "A volunteer-computing-based grid environment for connect6 applications," in *International Conference on Computational Science and Engineering*, 2009, vol. 1, pp. 110–117.
- [28] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, "Characterizing and Classifying Desktop Grid," in *IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 743–748.
- [29] Z. Constantinescu, "A Desktop Grid Computing Approach for Scientific Computing and visualisation," Norwegian University of Science and Technology, 2008.
- [30] L. F. G. Sarmenta and S. Hirano, "Bayanihan: building and studying web-based volunteer computing systems using Java," *Future Generation Computer Systems*, vol. 15, pp. 675–686, 1999.
- [31] D. P. Anderson, "BOINC: A system for public-resource computing and storage," in *International Workshop on Grid Computing*, 2004, pp. 4–10.
- [32] "SETI@home." [Online]. Available: <http://setiathome.ssl.berkeley.edu/>. [Accessed: 21-Feb-2015].
- [33] "Distributed.net." [Online]. Available: www.distributed.net/. [Accessed: 23-Feb-2015].

- [34] A. Chien, B. Calder, S. Elbert, and K. Bhatia, "Entropia : architecture and performance of an enterprise desktop grid system," *Journal of Parallel and Distributed Computing*, vol. 63, pp. 597–610, 2003.
- [35] M. Vladioiu and Z. Constantinescu, "Development Journey of QADPZ - A Desktop Grid Computing Platform," *International journal of Computers, Communicatuons & Control*, vol. 4, no. 1, pp. 82–91, 2009.
- [36] P. Kacsuk, J. Kovacs, Z. Farkas, A. C. Marosi, G. Gombas, and Z. Balaton, "SZTAKI Desktop Grid (SZDG): A flexible and scalable desktop grid system," *Journal of Grid Computing*, vol. 7, no. 4, pp. 439–461, Sep. 2009.
- [37] B. O. Christiansen, P. Cappello, M. F. Ionescu, M. O. Neary, K. Schauser, and D. Wu, "Javelin: Internet-Based Parallel Computing Using Java," *Concurrency and Computation: Practice and Experience*, vol. 9, no. 11, pp. 1139–1160, 1997.
- [38] A. L. Beberg, D. L. Ensign, G. Jayachandran, S. Khaliq, and V. S. Pande, "Folding@home: Lessons from eight years of volunteer distributed computing," in *IEEE International Parallel and Distributed Processing Symposium*, 2009, pp. 1–8.
- [39] O. Nov, D. Anderson, and O. Arazy, "Volunteer computing: a model of the factors determining contribution to community-based scientific research," in *International conference on World wide web*, 2010, pp. 741–750.
- [40] D. P. Anderson, "ACM Crossroads," *Volunteer Computing: the ultimate cloud*, pp. 7–10, Mar-2010.
- [41] D. P. Anderson, C. Christensen, and B. Allen, "Designing a Runtime System for Volunteer Computing," in *IEEE Computer*, 2006.
- [42] D. Anderson, J. Cobb, E. Korpela, and M. Lebofsky, "SETI @ home," *Communications of the ACH*, vol. 45, no. 11, pp. 56–61, 2002.
- [43] "Folding@home." [Online]. Available: <http://folding.stanford.edu/>. [Accessed: 17-Feb-2015].
- [44] A. L. Beberg and V. S. Pande, "Storage@home: Petascale Distributed Storage," in *IEEE International Parallel & Distributed Processing Symposium*, 2007, pp. 1–6.

- [45] “RSA.” [Online]. Available: <http://www.emc.com/domains/rsa/>. [Accessed: 23-Feb-2015].
- [46] “Distributed.net:History.” [Online]. Available: <http://en.wikipedia.org/wiki/Distributed.net>. [Accessed: 22-Feb-2015].
- [47] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang, “Characterizing and classifying desktop grid,” in *IEEE International Symposium on Cluster Computing and the Grid*, 2007, pp. 743–748.
- [48] R. T. Muranganwa and M. Thinyane, “Design of a multi-agent opportunistic grid computing platform,” *Multiagent and Grid Systems*, no. 10, pp. 199–212, 2014.
- [49] P. Mell and T. Grance, “The NIST Definition of Cloud Computing Recommendations of the National Institute of Standards and Technology,” *Nist Special Publication*, vol. 145, p. 7, 2011.
- [50] E. Kourpas, “Grid Computing: Past, Present and Future - An Innovation Perspective,” 2006.
- [51] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” in *Grid Computing Environments Workshop*, 2008, pp. 1–10.
- [52] A. Dimakis, Y. Wu, M. . Wainwright, and K. Ramchandran, “Network Coding for Distributed Storage Systems,” *IEEE Transactions on Information Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.
- [53] Y. Wang and A. Merchant, “Proportional-share scheduling for distributed storage systems,” in *5th USENIX Conference on File and Storage Technologies*, 2007, p. 4.
- [54] S. Ghemawat, H. Gobioff, and S.-T. Leung, “The Google file system,” *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. p. 29, 2003.
- [55] “BigTable.” [Online]. Available: <https://en.wikipedia.org/wiki/BigTable>. [Accessed: 03-Apr-2015].

- [56] F. Chang, J. Dean, S. Ghemawat, W. Hsieh, D. Wallach, M. Burrows, T. Chandra, A. Fike, and R. Gruber, "BigTable: A Distributed Storage System for Structured Data," *ACM Transactions on Computer Systems*, vol. 26, no. 2, 2008.
- [57] M. R. Genesereth, "Software Agents," *Communications of the ACH*, vol. 37, no. 7, 1994.
- [58] N. R. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems*, vol. 1, no. 1, pp. 275–306, 1998.
- [59] Shoham and Y, "Agent-oriented programming," *Artificial Intelligence*, vol. 60, no. 1, pp. 51–92, 1993.
- [60] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. MIT Press Cambridge, Massachusetts, 2000.
- [61] G. Weiß, "Adaptation and Learning in Multi-Agent Systems: Some Remarks and a Bibliography," in *IJCAI Workshop on Adaption and Learning in Multi-Agent Systems*, 1995, pp. 1–21.
- [62] "FIPA." [Online]. Available: <http://www.fipa.org/>. [Accessed: 31-Mar-2015].
- [63] F. Bellifemine, A. Poggi, and G. Rimassa, "JADE—A FIPA-compliant agent framework," in *Conference on the Practical Application of Intelligent Agents and Multi-agent Technology*, 1999, pp. 97–108.
- [64] M. T. Kone, A. Shimazu, and T. Nakajima, "The State of the Art in Agent Communication," *Knowledge and Information Systems*, vol. 2, no. 3, pp. 259–284, 2000.
- [65] B. Chaib-draa and F. Dignum, "Trends in Agent Communication Language," *Computational Intelligence*, vol. 18, no. 2, pp. 89–101, May 2002.
- [66] T. Finin, R. Fritzson, D. McKay, and R. McEntire, "KQML as an Agent Communication Language," in *International Conference on Information and knowledge Management*, 1994, pp. 456–463.
- [67] J. R. Searle, *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: The University Press, 1969.

- [68] B. Fabio, C. Giovanni, and G. Dominic, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.
- [69] A. Liekna, E. Lavendelis, and A. Grabovskis, “Experimental Analysis of Contract NET Protocol in Multi-Robot Task Allocation,” *Applied Computer Systems*, vol. 13, no. 1, pp. 6–14, 2012.
- [70] R. G. Smith, “The Contract Net Protocol: High Level Communication and Control in a Distributed Problem Solver,” in *IEEE Transactions on Computers*, 1980, pp. 1104–1113.
- [71] R. Davis and R. G. Smith, “Negotiation as a Metaphor for Distributed Problem Solving,” *Artificial Intelligence*, vol. 20, no. 1, pp. 63–109, 1983.
- [72] C. Yu and T. N. Wong, “A multi-agent architecture for multi-product supplier selection in consideration of the synergy between products,” *International Journal of Production Research*, 2015.
- [73] A. More, S. Vij, and D. Mukhopadhyay, “Agent Based Negotiation using Cloud - an Approach in E-Commerce,” in *ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India- Vol I*, Springer International Publishing, 2014, pp. 489–496.
- [74] J. Zhang, F. Ren, and M. Zhang, “Bayesian-based preference prediction in bilateral multi-issue negotiation between intelligent agents,” *Knowledge-Based Systems*, pp. 108–120, 2015.
- [75] B. An, V. Lesser, D. Irwin, and M. Zink, “Automated Negotiation with Decommitment for Dynamic Resource Allocation in Cloud Computing,” in *International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 981–988.
- [76] S. K. Garg, R. Buyya, and S. Versteeg, “Automated SLA Negotiation Framework for Cloud Computing,” in *IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013, pp. 235–244.
- [77] D. Ouelhadj, J. Garibaldi, J. Maclaren, R. Sakellariou, and K. Krishnakumar, “A Multi-agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing,” in *Advances in Grid Computing - EGC 2005 Lecture Notes in Computer Science*, Springer Verlag, 2005, pp. 651–660.

- [78] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Modern Approach to Artificial Intelligence*. MIT Press Cambridge, Massachusetts, 2000.
- [79] B. Moulin and B. Chaib-Draa, "An overview of distributed artificial intelligence," in *Foundations of Distributed Artificial Intelligence*, G. M. . O'Hare and N. . Jennings, Eds. New York: John Wiley & Sons Inc, 1996, pp. 3–55.
- [80] A. Singh, D. Juneja, and A. K. Sharma, "Agent Development Toolkits," *International Journal of Advancements in Technology*, vol. 2, no. 1, pp. 158–164, 2011.
- [81] G. Nguyen, T. Dang, L. Hluchy, M. Laclavik, Z. Balogh, and I. Budinska, "Agent Platform Evaluation and Comparison." Institute of informatics, Slovak Academy of Sciences, Pellucid 5FP IST -2001-34519, pp. 1–11, 2002.
- [82] P. E. Clements, T. Papaioannou, and J. Edwards, "Aglets: Enabling the virtual enterprise," in *International Conference on Managing Enterprises-Stakeholders, Engineering, Logistics and Achievement*, 1997.
- [83] S. Fischmeister, G. Vigna, and R. A. Kemmerer, "Evaluating the Security Of Three Java-Based Mobile Agent Systems," in *IEEE Mobile Agents Lecture*, Springer, 2001.
- [84] S. S. Mudumbai, W. Johnston, and A. Essiari, "Anchor Toolkit- A Secure Mobile Agent System," in *eScholarship*.
- [85] M. R. Thompson, A. Essiari, and S. Mudumbai, "Certificate-Based Authorization Policy in a PKI Environment," *ACM Transactions on Information and System Security*, vol. 6, no. 4, pp. 566–588, 2003.
- [86] H. S. Nwana, L. C. Lee, D. T. Ndumu, J. C. Collis, and I. R. Ipswich, "Zeus : A Toolkit and Approach for Building Distributed Multi-Agent Systems," *Applied Artificial Intelligence Journal*, vol. 13, pp. 129–186, 1999.
- [87] D. Camacho, R. Aler, C. Castro, and J. M. Molina, "Performance Evaluation of Zeus , JADE and SkeletonAgent Frameworks," in *IEEE International Conference on Systems, Man and Cybernetics*, 2002.
- [88] "JADE." [Online]. Available: [Http://jade.tilab.com/](http://jade.tilab.com/). [Accessed: 15-Mar-2015].

- [89] R. C. Nicol and P. D. O'Brien, "FIPA — Towards a Standard for Software Agents," *BT Technology Journal*, vol. 16, no. 3, pp. 51–59, 1998.
- [90] B. Fabio, C. Giovanni, and G. Dominic, "JADE and the Agents Paradigm," in *Developing Multi-Agent Systems with JADE*, 2007, pp. 29–34.
- [91] B. Fabio, C. Giovanni, and G. Dominic, "Agent Tasks," in *Developing Multi-Agent Systems with JADE*, 2007, pp. 57–62.
- [92] "JADE book-trading Example." [Online]. Available: <https://github.com/bluezio/jade-booktrading>. [Accessed: 04-Apr-2015].
- [93] "FIPA ACL Message Structure Specification," 2002. [Online]. Available: <http://www.fipa.org/specs/fipa00061/SC00061G.pdf>. [Accessed: 04-Apr-2015].
- [94] B. Fabio, C. Giovanni, and G. Dominic, "Admin and Debugging Tools," in *Developing Multi-Agent Systems with JADE*, 2007, pp. 42–50.
- [95] A. Bieszczad, "Mobile Agents for Network Management," in *IEEE Communications Survey*, 1998, vol. 1.
- [96] J. Huang and B. H. Far, "Information Collection and Survey: Infrastructure, APIs, and Software Tools for Agent-based Systems (An Overview of JADE)," 2003.
- [97] N. R. Jennings, "Agent-Based Computing : Promise and Perils," in *International Joint Conference on Artificial Intelligence*, 1997, pp. 1429–1436.
- [98] M. Wooldridge and N. R. Jennings, "Pitfalls of Agent-Oriented Development," in *International Conference on Autonomous Agents*, 1998, pp. 385–391.
- [99] F. Kon, A. Goldchleger, A. Goldman, M. Finger, and C. Bezerra, "Grid middleware leveraging idle computing power of desktop machines," in *Concurrency and Computation: Practice and Experience*, 2002, pp. 1–12.
- [100] A. Karmouch, L. Korba, and E. R. M. Madeira, "Mobigrid*: Framework for Mobile Agents on Computer Grid environments," in *Mobility Aware Technologies and Applications*, 2004, pp. 147–157.

- [101] J. M. Solanki, S. Khushalani, and N. N. Schulz, "A multi-agent solution to distribution systems restoration," *IEEE Transactions on Power Systems*, vol. 22, no. 3, pp. 1026–1034, 2007.
- [102] J. Wu, X. Xu, P. Zhang, and C. Liu, "A novel multi-agent reinforcement learning approach for job scheduling in Grid computing," *Future Generation Computer Systems*, vol. 27, no. 5, pp. 430–439, 2011.
- [103] A. Galstyan, K. Czajkowski, and K. Lerman, "Resource allocation in the Grid using reinforcement learning," in *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004, vol. 3, pp. 1314–1315.
- [104] A. Carzaniga, G. Pietro Picco, G. Vigna, and U. C. S. Barbara, "Is Code Still Moving Around? Looking Back at a Decade of Code Mobility," in *International Conference on Software Engineering*, 2007, pp. 9–20.
- [105] H. Kumar and A. K. Verma, "Comparative Study of Distributed Computing Paradigms," *International Journal of Information Technology*, vol. 1, no. 2, pp. 97–100, 2009.
- [106] R. F. Lopes and F. Da Silva, "Migration Transparency in a Mobile Agent Based Computational Grid," in *International Conference on Simulation, Modeling and Optimisation*, 2005, pp. 31–36.
- [107] D. B. Lange and M. Oshima, "Seven good reasons for mobile agents," *Communications of the ACM*. 1999.
- [108] W. J. Buchanan, M. Naylor, and A. V. Scott, "Enhancing network management using mobile agents," in *International Conference and Workshop on the Engineering of Computer Based Systems*, 2000, pp. 218–226.
- [109] J. Stamos and G. Gifford, "Remote Evaluation," in *ACM Transactions on Programming Languages and Systems*, 1990, pp. 537–565.
- [110] M. R. Lee, "An Exception Handling of Rule-Based Reasoning Using Case-Based Reasoning," *Journal of Intelligent and Robotic Systems*, vol. 35, no. 3, pp. 327–338, 2002.
- [111] J. Prentzas and L. Hatzilygeroudis, "Categorizing Approaches Combining Rule-Based and Case- Based Reasoning," *Expert Systems*, vol. 24, no. 2, pp. 97–122, 2007.

- [112] K. Walzer, T. Breddin, and M. Groch, "Relative Temporal Constraints in the Rete Algorithm for Complex Event Detection," in *International Conference on Distributed Event-Based Systems*, 2008, pp. 147–155.
- [113] S. Lin and X. Huang, "Rule-Based Systems," in *International Conference of Computer Science, Environment, Ecoinformatics, and Education*, 2011, p. 58.
- [114] S. Singh and R. Karwayun, "A Comparative Study of Inference Engines," in *International Conference on Information Technology: New Generations*, 2010, pp. 53–57.
- [115] C. L. Forgy, "Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *Artificial Intelligences*, vol. 19, no. 1, pp. 17–37, 1982.
- [116] K. Walzer, M. Groch, and T. Breddin, "Time to the Rescue - Supporting temporal reasoning in the rete algorithm for complex event processing," in *International conference on Database and Expert Systems Applications*, 2008, pp. 635–642.
- [117] Bhansali and B. N. Grosz, "Extending the SweetDeal Approach for e-Procurement Using SweetRules and RuleML," in *Rules and Rule Markup Languages for the Semantic Web Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2005, pp. 113–129 .
- [118] J. J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson, "Jena: Implementing the Semantic Web Recommendations," in *International World Wide Web Conference on Alternate Track Papers & Posters*, 2004, pp. 74–83.
- [119] Y. Zou, T. Finin, and H. Chen, "F-OWL: An Inference Engine for Semantic Web," in *Formal Approaches to Agent-Based Systems Lecture Notes in Computer Science*, Springer Berlin Heidelberg, 2005, pp. 238–248.
- [120] "Drools(BRMS)." [Online]. Available: <http://www.drools.org/>. [Accessed: 20-Feb-2015].
- [121] S. Conger, "Knowledge Management for Information and Communications Technologies for Development Programs in South Africa," *Information Technology for Development*, no. August 2014, pp. 1–22, May 2014.
- [122] M. Thinyane, A. Terzoli, H. Thinyane, S. Hansen, and S. Gumbo, "Living Lab Methodology as an Approach to Innovation in ICT4D: The Siyakhula Living Lab Experience," in *IST-Africa 2012*, 2012, pp. 1–9.

- [123] “Disk Mirroring.” [Online]. Available: https://en.wikipedia.org/wiki/Disk_mirroring. [Accessed: 03-Mar-2015].
- [124] “Non-functional requirement.” [Online]. Available: http://en.wikipedia.org/wiki/Non-functional_requirement. [Accessed: 06-Jun-2015].
- [125] D. Gross and E. Yu, “Evolving System Architecture to Meet Changing Business Goals: an Agent and Goal-Oriented Approach,” *ICSE-2301 Workshop: From Software Requirements to Architectures*, pp. 16–21.
- [126] “Exploring mobility.” [Online]. Available: <http://www.iro.umontreal.ca/~vaucher/Agents/Jade/>. [Accessed: 23-Jun-2015].
- [127] “Rule Engine Intergration.” [Online]. Available: http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html.
- [128] “Drools Development List.” [Online]. Available: <https://issues.jboss.org/browse/DROOLS-538>.
- [129] “Jade and Jess.” [Online]. Available: http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html.
- [130] “JADE limitation.” [Online]. Available: <http://jade.tilab.com/support/faq/what-does-the-warning-messagemanager-queue-size-10000000-mean/>. [Accessed: 02-Mar-2015].
- [131] “JADE Security.” [Online]. Available: http://jade.tilab.com/doc/tutorials/JADE_Security.pdf.

Appendix A - Compute Implementation Details

A. MA Platform

The following are code snippets of the different agents implemented in the MA compute Platform. The detailed functionality of the agent components are discussed in Section 7.1.

A.1 Processing User Agent-PUA

```
addBehaviour(new OneShotBehaviour() {
    public void action() {
        // Search for PRA_AIDss
        DFAgentDescription template = new DFAgentDescription();
        ServiceDescription sd = new ServiceDescription();
        sd.setType("PRA");
        template.addServices(sd);
        try {
            DFAgentDescription[] result = DFService.search(myAgent, template);
            PRA_AIDss = new AID[result.length];
            for (int i = 0; i < result.length; ++i) {
                PRA_AIDss[i] = result[i].getName();
                System.out.println(PRA_AIDss[i].getName());
            }
        } catch (FIPAException ex) {
            System.out.println("Exception-- "+ex);
        }
        //Initialise CFP and Bid Evaluation Behaviour
        myAgent.addBehaviour(new RequestPerformer());
    }
});
```

Appendix A.1.2: PRA DF Search

```
//Send CFP to PRA agents
ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
for (int i = 0; i < PRA_AIDss.length; ++i) {
    cfp.addReceiver(PRA_AIDss[i]);
}
cfp.setContent(service_requested);
cfp.setConversationId("CNP-negotiation");
cfp.setReplyWith("cfp"+System.currentTimeMillis());
myAgent.send(cfp);
// Prepare the template to get bids
mt = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("CNP-negotiation"), MessageTemplate.MatchInReplyTo(cfp.getReplyWith()));
```

Appendix A.1.3: CFP Forwarding

```

// Receive all proposals/refusals PRA_AIDss
ACLMessage reply = myAgent.receive(mt);
if(reply != null) {
    // Reply received
    if(reply.getPerformative() == ACLMessage.PROPOSE) {
        // This is an offer
        double cpuload = Double.parseDouble(reply.getContent());
        if(OptimumPlatformPRA == null || cpuload < least_cpu_load_identifier) {
            //This is the bid with least NCL here
            least_cpu_load_identifier = cpuload;
            OptimumPlatformPRA = reply.getSender();
        }
    }
    Bidcount++;
    if (Bidcount >= PRA_AIDss.length) {
        //All CFP replies Received from PRAs
        step = 2;
    }
}

```

Appendix A.1.4: PUA Bid Evaluation

```

// Send offer>>>>OptimumPlatformPRA with least NCL
ACLMessage offer = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
offer.addReceiver(OptimumPlatformPRA);
offer.setContent("request WorkerAgents info");
offer.setConversationId("Contract-NP-negotiation");
offer.setReplyWith("offer"+System.currentTimeMillis());
myAgent.send(offer);
// Prepare the Template to get shared nodes information
mt = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("Contract-NP-negotiation"),MessageTemplate.MatchInReplyTo
    (offer.getReplyWith()));

```

Appendix A.1.5: Contract Allocation

```

//Start PMA GUI
public void call() {
    myGui = new PMA_GUI(this,locations.keySet());
    myGui.setVisible(true);
}

```

Appendix A.1.6: Initialising PMA GUI

A.2 Node Processing Agent-NPA

```
mysql> describe CPU_util_info;
```

Field	Type	Null	Key	Default	Extra
ss_pcname	varchar(25)	NO	PRI	NULL	
os_version	float	NO		NULL	
ss_architecture	varchar(15)	NO		NULL	
ss_platform	varchar(15)	NO		NULL	
available_processors	int(11)	NO		NULL	
comm_virtual_memory_size	double	NO		NULL	
free_physical_memory_size	double	NO		NULL	
process_cpu_load	double	NO		NULL	
process_cpu_time	double	NO		NULL	
system_cpu_load	double	NO		NULL	
system_load_average	double	NO		NULL	

Appendix A.1.7: CPU Utilisation Database

```
operatingSystemMXBean = (com.sun.management.OperatingSystemMXBean)
    ManagementFactory.getOperatingSystemMXBean();
ss_pcname = System.getProperty("user.name");
os_version = operatingSystemMXBean.getCommittedVirtualMemorySize();
ss_architecture = operatingSystemMXBean.getArch();
ss_platform = operatingSystemMXBean.getName();
available_processors = operatingSystemMXBean.getAvailableProcessors();
comm_virtual_memory_size = operatingSystemMXBean.getCommittedVirtualMemorySize();
free_physical_memory_size = operatingSystemMXBean.getFreePhysicalMemorySize();
process_cpu_load = operatingSystemMXBean.getProcessCpuLoad();
process_cpu_time = operatingSystemMXBean.getProcessCpuTime();
system_cpu_load = operatingSystemMXBean.getSystemCpuLoad();
system_load_average = operatingSystemMXBean.getSystemLoadAverage();
```

Appendix A.1.8: CPU utilisation parameters

```
try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    insert="INSERT INTO CPU_util_info VALUES('"+ss_pcname+"','"+os_version+"'"
        + "','"+ss_architecture+"','"+ss_platform+"','"+available_processors+"','"+comm_virtual_memory_size+"'"
        + "','"+free_physical_memory_size+"','"+process_cpu_load+"','"+process_cpu_time+"'"
        + "','"+system_cpu_load+"','"+system_load_average+"')";
    stmt.executeUpdate(insert);
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
}
```

Appendix A.1.9: CPU info Registration

```

try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    update="UPDATE CPU_util_info SET os_version = '"+os_version+"'
        + ", ss_architecture='"+ss_architecture+"',ss_platform = '"+ss_platform + "'"
        + ", available_processors='"+available_processors+"',comm_virtual_memory_size='"+
        comm_virtual_memory_size + "',free_physical_memory_size='"+ free_physical_memory_size+"',"
        + "process_cpu_load='"+process_cpu_load+"',process_cpu_time='"+process_cpu_time+"',"
        + "system_cpu_load='"+usable_disk_space+"',system_load_average = '"+system_load_average+"'"
        + "where ss_pcname = '"+ss_pcname+"'";
    stmt.executeUpdate(update);
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
}

```

Appendix A.1.10: Updating CPU info

A.3 Processing Resolver Agent-PRA

```

// Register PRA service
DFAgentDescription description_rs = new DFAgentDescription();
description_rs.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("PRA");
sd.setName("Resolver");
description_rs.addServices(sd);
try{
    DFService.register(this, description_rs);
}catch (FIPAException ex){
    System.out.println("Exception --"+ex);
}

```

Appendix A.1.11: PRA Registration

```

//Section returns the shared nodes(JADE runtimenvironments)
//AMS container handling service
AMSSubscriber subscriber = new AMSSubscriber() {
    public void installHandlers(Map map) {
        AMSSubscriber.EventHandler addedHandler = new AMSSubscriber.EventHandler() {
            public void handle(Event event) {
                AddedContainer addedContainer = (AddedContainer) event;
                availableContainers.add(addedContainer.getContainer());
            }
        };
        map.put(IntrospectionVocabulary.ADDED_CONTAINER, addedHandler);
        AMSSubscriber.EventHandler removedHandler = new AMSSubscriber.EventHandler() {
            public void handle(Event event) {
                RemovedContainer removedContainer = (RemovedContainer) event;
                ArrayList<ContainerID> temp = new ArrayList<>(availableContainers);
                for(ContainerID container : temp){
                    if(container.getID().equalsIgnoreCase(removedContainer.getContainer().
                        .getID())) availableContainers.remove(container);
                }
            }
        };
        map.put(IntrospectionVocabulary.REMOVED_CONTAINER, removedHandler);
    }
};

```

Appendix A.1.12: AMSSubscriber


```

try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.16.12.150:3306/MAOGP"
        , "root", "toor");
    stmt=(Statement)con.createStatement();
    select = "SELECT AVG(system_load_average) FROM CPU_util_info";
    ResultSet rs = stmt.executeQuery(select);
    while(rs.next()){
        mean_load_average=Double.parseDouble(rs.getString(1));
    }
    rs.close();
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
}

```

Appendix A.1.13: System load Average Calculation

```

if (mean_load_average != 0){
    //return bid here
    reply.setPerformative(ACLMessage.PROPOSE);
    reply.setContent(String.valueOf(mean_load_average));
    myAgent.send(reply);
}

```

Appendix A.1.14: Returns a bid

```

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.ACCEPT_PROPOSAL);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
    String service_type = msg.getContent();
    ACLMessage reply = msg.createReply();
    //if shared nodes are available: return the container runtime identities
    if (availableContainers != null&&service_type.equals("request_WorkerAgents_info")){
        reply.setPerformative(ACLMessage.INFORM);
        try {
            reply.setContentObject(availableContainers);
        }catch (IOException ex) {
            Logger.getLogger(PRA.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    else{
        reply.setPerformative(ACLMessage.FAILURE);
        reply.setContent("No shared resources in the platform");
    }
    myAgent.send(reply);
}

```

Appendix A.1.15: Accepting Contract

A.4 MobileAgent.java

```
// Retrieve arguments passed during agent creation
Object[] args = getArguments();
controller = (AID) args[0];
destination = here();//identifies the underlying JADE runtime environment
init(); //registers the ontology and language
report_results_sourcenode = here().getName();//extracts the user's source JADE runtime
FileInputStream fileInputStream=null;
//retrieves the jess rule file with reasoning logic
file = new File("/root/NetBeansProjects/mobilecomponent/src/mobilecomponent/rulesfile.clp");
bFile = new byte[(int) file.length()];
try {
    //convert file into array of bytes
    fileInputStream = new FileInputStream(file);
    fileInputStream.read(bFile);
    fileInputStream.close();
}catch(Exception ex){
    System.out.println("Exception-- "+ ex);
}
```

Appendix A.1.16: MA void setup

```
//Behaviour implements the PMA GUI spawn and Kill tabs
class ReceiveCommands extends CyclicBehaviour{
    ReceiveCommands(Agent a) { super(a); }
    public void action() {
        ACLMessage msg = receive(MessageTemplate.MatchSender(controller));
        if (msg == null) {
            block();
            return;
        }
        if(msg.getPerformative() == ACLMessage.REQUEST) {
            try{
                ContentElement content = getContentManager().extractContent(msg);
                Concept concept = ((Action)content).getAction();
                if (concept instanceof MoveAction){
                    MoveAction ma = (MoveAction)concept;
                    Location l = ma.getMobileAgentDescription().getDestination();
                    if (l != null) doMove(destination = l);
                }
                else if (concept instanceof KillAgent){
                    myGui.setVisible(false);
                    myGui.dispose();
                    doDelete();
                }
            }catch (Exception ex) {
                System.out.println("Exception-- "+ex);
            }
        }
        else{ System.out.println("Unexpected msg from controller agent"); }
    }
}
```

Appendix A.1.17: doMove () and doDelete ()

```

void init(){
    // Register language and ontology
    getContentManager().registerLanguage(new SLCodec());
    getContentManager().registerOntology(MobilityOntology.getInstance());
}

```

Appendix A.1.18: init () method

```

protected void afterMove(){
    //initialises the ontology and language
    init();
    try {
        //Writes the jess rule file in the shared node's home directory
        fileOutputStream = new FileOutputStream(directory+"/"+finalfile.clp");
        fileOutputStream.write(bFile);
        fileOutputStream.close();
    }catch (IOException ex){
        System.out.println("Exception-- "+ex);
    }
    //Executes the SimpsonRule user applications
    SimpsonsRule compute_application = new SimpsonsRule();
    compute_application.main();
    //Behaviour initilises the reasoning component
    addBehaviour(new TickerBehaviourImpl(this, 10000));
    //Extract destination node JADE runtime
    localContainerName = here().getName();
    container_array.add(localContainerName);//keeps track of visited nodes
    //If destination node==user source JADE runtime&&
    //report results and terminate
    if(localContainerName.equals(container_array.get(0))){
        System.out.print("processing results "+ count );
        doDelete();
    }
}
}

```

Appendix A.1.19: afterMove ()

```

public class SimpsonsRule {
    public double f(double x){
        return Math.exp(- x * x / 2) / Math.sqrt(2 * Math.PI);
    }
    public double SimpsonsRule(double a, double b) {
        int N = 10000;
        double h = (b - a) / (N - 1);
        double sum = 1.0 / 3.0 * (f(a) + f(b));
        for (int i = 1; i < N - 1; i += 2) {
            double x = a + h * i;
            sum += 4.0 / 3.0 * f(x);
        }
        for (int i = 2; i < N - 1; i += 2) {
            double x = a + h * i;
            sum += 2.0 / 3.0 * f(x);
        }
        return sum * h;
    }
    public void main() {
        double count =0;
        for(b=2; b < 279; b++){
            for(a= 1; a<300;a++){
                SimpsonsRule(a,b);
                count++;
            }
        }
        ContainerID result = new ContainerID();
        result.setName(report_results_sourcenode);
        doMove(result);
    }
}

```

Appendix A.1.20: Simpson Rule Class

```

public void onTick(){
    OperatingSystemMXBean operatingSystemMXBean = ManagementFactory.getOperatingSystemMXBean();
    for (Method method : operatingSystemMXBean.getClass().getDeclaredMethods()) {
        method.setAccessible(true);
        if (method.getName().startsWith("getSystemCpuLoad") && Modifier.isPublic(method.getModifiers())) {
            try{
                cpuinfo = (Double) method.invoke(operatingSystemMXBean);
                //initilising Jess
                System.out.println(cpuinfo);
            }catch(Exception ex){System.out.println("exception here---"+ex);}
            try{
                System.out.println("parttern matching initiated");
                engine = new Rete();
                engine.batch(directory+"/"+finalfile.clp");
                engine.watchAll();
                // Plug in the "chosen date"
                Fact fact = new Fact("rulereasoning", engine);
                fact.setSlotValue("cpuinfo", new Value(new ValueVector().add(cpuinfo), RU.LIST));
                engine.assertFact(fact);
                // Run the rule and report the result
                count = engine.run();
                if (count > 0) {
                    action = engine.getGlobalContext().getVariable("**var*").toString();
                    System.out.println("action = " + action);
                } else {
                    System.out.println("ignore call.");
                }
            }
        }
    }
}

```

Appendix A.1.21: Initializing Jess

```

if(action.equals("migrate")){
    // Register language and ontology
    getContentManager().registerLanguage(new SLCodec());
    getContentManager().registerOntology(MobilityOntology.getInstance());
    try {
        //Request Container identities fromAMS
        sendRequest(new Action(getAMS(), new QueryPlatformLocationsAction()));
        //Receive response from AMS
        MessageTemplate mt = MessageTemplate.and(MessageTemplate.MatchSender(getAMS()),
            MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        ACLMessage resp = blockingReceive(mt);
        ContentElement ce = getContentManager().extractContent(resp);
        Result result = (Result) ce;
        //read all identities: (i.e. Container-1@kalibacktrack-Raymond)
        jade.util.leap.Iterator it = result.getItems().iterator();
        while (it.hasNext()){
            Location loc = (Location)it.next();
            if (loc.toString().length() > 12){
                containers = loc.toString().substring(0, 11);
                //remove platform container here
                if(!containers.startsWith("M")){
                    AMS_containers.add(containers);
                }
            }
        }
    }
}

```

Appendix A.1.22: Resolve AMS Containers

```

//Check traversed nodes and alternatives
for( int j=0; j< AMS_containers.size(); j++ ){
    traversed_containers = container_array.iterator().next();
    //check nodes not visited here
    if(!AMS_containers.get(j).equals(traversed_containers)){
        alternative_idle_node.add(AMS_containers.get(j));
        System.out.println("alternativenode: "+alternative_idle_node.get(j));
    }
}
//migrates to alternative node
ContainerID alternative_location = new ContainerID();
alternative_location.setName(alternative_idle_node.get(0));
doMove(alternative_location);

```

Appendix A.1.23: Resolve Alternative Shared Nodes

```

//If destination node==user source JADE runtime&&
//report results and terminate
if(localContainerName.equals(container_array.get(0))){
    System.out.print("processing results "+ count );
    doDelete();
}

```

Appendix A.1.24: Reporting on Results

Appendix B - Storage Implementation Details

B. Storage Component

B.1 DM File-System: Upload Service

```
public void setup() {
    String[] EXTENSION=new String[]{"all Files"};
    JFileChooser fc= new JFileChooser("Select File");
    fc.setFileSelectionMode(JFileChooser.FILES_AND_DIRECTORIES);
    FileNameExtensionFilter filter=new FileNameExtensionFilter
        ("MAOGP Supported video Formats",EXTENSION);
    fc.setFileFilter(filter);
    int ret = fc.showOpenDialog(null);
    if (ret == JFileChooser.APPROVE_OPTION) {
        try{
            selectedFile = fc.getSelectedFile();
            //extract sourcefile-name
            SourceFileName = selectedFile.getName();
            //convert selected file into byte[]
            bFile = new byte[(int) selectedFile.length()];
            fc.cancelSelection();
        }catch(Exception e){System.out.println("error:--"+e);}
    }
}
```

Appendix B.1.1: File chooser

```
//search for registered RA&& initiate upload services
addBehaviour(new OneShotBehaviour() {
    @Override
    public void action() {
        //Search for broker Service
        DFAgentDescription template = new DFAgentDescription();
        ServiceDescription Service_dcrp = new ServiceDescription();
        Service_dcrp.setType("BrokerService_uploads");
        template.addServices(Service_dcrp);
        try {
            DFAgentDescription[] result = DFService.search(myAgent, template);
            RAs_AIDS = new AID[result.length];
            for (int i = 0; i < result.length; ++i) {
                //Pop RA AIDs into -->Array
                RAs_AIDS[i] = result[i].getName();
            }
        }catch (FIPAException fe) {System.out.println("Exception:--"+fe);}
        //initialise Node_Request Behaviour
        myAgent.addBehaviour(new Request_nodes());
    }
});
```

Appendix B.1.2: Identify Broker Services

```

public class Request_nodes extends Behaviour {
//template receives replies
MessageTemplate temp;
int step = 0;
@Override
public void action() {
    switch (step) {
    case 0:
        // Request storage services from identified RAs_AIDS-->RA_AIDS
        ACLMessage cfp = new ACLMessage(ACLMessage.CFP);
        for (int i = 0; i < RAs_AIDS.length; ++i) {
            cfp.addReceiver(RAs_AIDS[i]); //define destination RAs_AIDS
        }
        //Filemetadata(SCUA_AID,SourceFiname,File payload)
        forwards_file_metadata = new ArrayList();
        forwards_file_metadata.add(bFile);
        forwards_file_metadata.add(SourceFileName);
        forwards_file_metadata.add(myAgent.getAID());
        {
        try {
            //define the Content Object
            cfp.setContentObject(forwards_file_metadata);
        } catch (IOException ex) {Logger.getLogger(SCUA.class.getName()).
            log(Level.SEVERE, null, ex);}
        }
        cfp.setConversationId("Request_Storage_Service");
        cfp.setReplyWith("cfp"+System.currentTimeMillis());
        myAgent.send(cfp);
        //Template to recieves Upload Status
        temp = MessageTemplate.and(MessageTemplate.MatchConversationId
            ("Request_Storage_Service"),MessageTemplate.MatchInReplyTo
            (cfp.getReplyWith()));
    }
}

```

Appendix B.1.3: Request_nodes Behaviour

```

// receive file upload status here
ACLMessage reply = myAgent.receive(temp);
if(reply != null && reply.getPerformative() == ACLMessage.INFORM) {
    //get String upload Status
    uploadstatus = reply.getContent();
    SwingUtilities.invokeLater(new Runnable() {
    @Override
    //Display upload status to grid user
    public void run() {
        JOptionPane.showMessageDialog(null, uploadstatus);
    }
    });
}

```

Appendix B.1.4: Feedback Dialogue

```

        DFAgentDescription dfd = new DFAgentDescription();
        dfd.setName(getAID());
        ServiceDescription sd = new ServiceDescription();
        sd.setType("BrokerService_uploads");
        sd.setName("RA");
        dfd.addServices(sd);

        try {
            DFService.register(this, dfd);
        } catch (FIPAException fe) {
            System.out.println("Exception:--"+fe);
        }
    }

```

Appendix B.1.5: PRA Registration

```

if(fileSize<60000){
    try{
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/"
            + "maogp", "root", "toor");
        stmt=(Statement)con.createStatement();
        //Select SS mac address with 100GB free usable disk space
        search="select mac_address from nodeinfo where usable_disk_space"
            + "<=100949000000";
        rs = stmt.executeQuery(search);
        mac_address = new ArrayList<>();
        //add string SS mac addresses into-->Arraylist
        while (rs.next()) {
            mac_address.add(rs.getString("mac_address"));
        }
        rs.close();
        //identify SA_AID associated with SS identified mac addresses in Arraylist
        template1 = new DFAgentDescription();
        sd = new ServiceDescription();
        storageagent = new AID[mac_address.size()];
        int count = 0;
        for (String mac_addres : mac_address) {
            sd.setType("storageagent" + mac_addres);
            template1.addServices(sd);
            try {
                DFAgentDescription[] result = DFService.search(this, template1);
                if(result.length > 0){
                    storageagent[count++] = result[0].getName();
                }
            } catch (FIPAException fe) {System.out.println("Error"+fe);}
        }
    } catch (ClassNotFoundException | SQLException e){
        JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
    }
}

```

Appendix B.1.6: OPTIMISESTORAGE logic


```

//SS 1 upload request
ACLMessage file1_upload = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file1_upload.addReceiver(storageagent[0]);
file1_upload.setContentObject(File_metadata);
file1_upload.setConversationId("node_1_storage");
file1_upload.setReplyWith("file2_upload"+System.currentTimeMillis());
myAgent.send(file1_upload);
start = System.currentTimeMillis();
//SS 2 upload request
ACLMessage file2_upload = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file2_upload.addReceiver(storageagent[1]);
file2_upload.setContentObject(File_metadata);
file2_upload.setConversationId("node_2_storage");
file2_upload.setReplyWith("file2_upload"+System.currentTimeMillis());
myAgent.send(file2_upload);
//SS 3 upload request
ACLMessage file3_upload = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file3_upload.addReceiver(storageagent[2]);
file3_upload.setContentObject(File_metadata);
file3_upload.setConversationId("node_3_storage");
file3_upload.setReplyWith("file3_upload"+System.currentTimeMillis());
myAgent.send(file3_upload);
// Prepare the Template to get SS storage feedback
file1_upload_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_1_storage"),MessageTemplate.MatchInReplyTo(file1_upload.getReplyWith()));
file2_upload_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_2_storage"),MessageTemplate.MatchInReplyTo(file2_upload.getReplyWith()));
file3_upload_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_3_storage"),MessageTemplate.MatchInReplyTo(file3_upload.getReplyWith()));

```

Appendix B.1.7: Sending Uploads to SAs

```

// Receive upload feedback
node1_response = myAgent.receive(file1_upload_response);
try {
    //SS upload parameters received here
    if (node1_response != null && !node1_response.getContentObject().equals("")
        && node1_response.getPerformative() == ACLMessage.INFORM) {
        //extract upload parameters
        SS_1 = (ArrayList) node1_response.getContentObject();
    }else{
        System.out.println("node1_response is null");
    }
} catch (UnreadableException Er) {
    Logger.getLogger(RA.class.getName()).log(Level.SEVERE, null, Er);
}

```

Appendix B.1.8: SS 1 feedback


```

//Send SCUA feedback && call Store_Upload_Params
if(SS_1 != null&&SS_2 != null&&SS_3!= null){
    Store_Upload_Params(SS_1,SS_2,SS_3);
    ACLMessage reply = new ACLMessage(ACLMessage.INFORM);
    reply.addReceiver((AID) SS_1.get(2));
    reply.setContent("Upload Successful");
    myAgent.send(reply);
}
comment
//Save parameterd in LocationsDatabase
public void Store_Upload_Params(ArrayList SS_1,ArrayList SS_2,ArrayList SS_3){
    String Node1_MAC,Node2_MAC,Node3_MAC,
    SA1_FileName,SA2_FileName,SA3_FileName;
    AID SCUA_AID = null;
    try{
        //Check if (SS_1&&SS_2&&SS_3).equal(SCUA AID)
        if (SS_1.get(3).equals(SS_2.get(3))&&SS_2.get(3).equals(SS_3.get(3))){
            SCUA_AID = (AID) SS_1.get(3);
        }else{System.out.println("System error");}
        Node1_MAC = (String) SS_1.get(1);
        SA1_FileName = (String) SS_1.get(0);
        Node2_MAC = (String) SS_2.get(1);
        SA2_FileName =(String) SS_2.get(0);
        Node3_MAC = (String) SS_3.get(1);
        SA3_FileName = (String) SS_3.get(0);
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/"
            + "maogp", "root", "toor");
        stmt=(Statement)con.createStatement();
        insert="INSERT INTO LocationsDatabase VALUES ('"+SCUA_AID+"','"+File_metadata.get(1)
            + "','"+Node1_MAC + "','"+SA1_FileName+"','"+Node2_MAC+"','"+SA2_FileName+"',"
            + "'"+Node3_MAC+"','"+SA3_FileName+"')";
        System.out.println("the sql statement "+ insert);
        stmt.executeUpdate(insert);
    }
}

```

Appendix B.1.9: Upload Feedback Return

```

try {
    ip_address = InetAddress.getLocalHost();
    aid = getAID().getName();
    NetworkInterface network = NetworkInterface.getByInetAddress(ip_address);
    byte[] mac = network.getHardwareAddress();
    //Extract SS mac address
    mac_address = new StringBuilder();
    for (int i = 0; i < mac.length; i++) {
        mac_address.append(String.format("%02X%s", mac[i],
            (i < mac.length - 1) ? "-" : ""));
    }
} catch (UnknownHostException | SocketException err) {
    System.out.println("Error"+err);
}
//Register SA service here
DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription service = new ServiceDescription();
//set service name-->"storageagent"+mac_address
service.setType("storageagent"+mac_address);
service.setName("SA");
dfd.addServices(service);
try {
    DFService.register(this, dfd);
} catch (FIPAException err) {
    System.out.println("Error--"+err);
}
}

```

Appendix B.1.10: SA Service Registration

```

operatingSystemMXBean = (com.sun.management.OperatingSystemMXBean)
    ManagementFactory.getOperatingSystemMXBean();
ss_pcname = System.getProperty("user.name");
total_disk_size = new File("/").getTotalSpace();
free_disk_space = new File("/").getFreeSpace();
usable_disk_space = new File("/").getUsableSpace();
free_swap_space_size = operatingSystemMXBean.getFreeSwapSpaceSize();
total_swap_space = operatingSystemMXBean.getTotalSwapSpaceSize();
ss_architecture = operatingSystemMXBean.getArch();
ss_platform = operatingSystemMXBean.getName();
os_version = operatingSystemMXBean.getCommittedVirtualMemorySize();

```

Appendix B.1.11: nodeinfo SS Parameters

```

try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    insert="INSERT INTO nodeinfo VALUES('"+ss_pcname+"','"+ip_address.getHostAddress()+"'
        + ", '"+mac_address+"','"+ss_platform+"','"+ss_architecture+"','"+os_version+"'"
        + ", '"+total_swap_space+"','"+free_swap_space_size+"','"+total_disk_size+"'"
        + ", '"+free_disk_space+"','"+usable_disk_space+"');";
    stmt.executeUpdate(insert);
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage(), "Error", 1);
}

```

Appendix B.1.12: SS Information Registration

```

try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    update="UPDATE nodeinfo SET ss_pcname='"+ss_pcname+"', ip_address='"+
        + ip_address.getHostAddress()+"', ss_platform='"+ss_platform+"'"
        + ", ss_architecture='"+ss_architecture+"', os_version='"+os_version+"'"
        + ", total_swap_space='"+total_swap_space+"', free_swap_space_size='"
        + free_swap_space_size+"', total_disk_size='"+total_disk_size+"', free_disk_space"
        + "'"+free_disk_space+"', usable_disk_space='"+usable_disk_space+"'"
        + "where mac_address = '"+mac_address+"';";
    stmt.executeUpdate(update);
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage(), "Error", 1);
}

```

Appendix B.1.13: System Information Updates

```

public void setup() {
    //Create a SS HOME directory folder to handle uploads
    Home_PATH = System.getenv("HOME");
    Uploads_Folder = new File(Home_PATH+"/"+ "StorageFolder");
    Uploads_Folder.mkdir();
    if(Uploads_Folder.exists()){
        System.out.print("Folder for Uploads Created");
    }
}

```

Appendix B.1.14: Creating an Upload Folder

B.2 DM File-System: Download Service

```

//Search for RAs specific for Downloads
DFAgentDescription template = new DFAgentDescription();
ServiceDescription sd = new ServiceDescription();
sd.setType("BrokerService_downloads");
template.addServices(sd);
try {
    DFAgentDescription[] result = DFService.search(myAgent, template);
    RA_AIDs = new AID[result.length];
    for (int i = 0; i < result.length; ++i) {
        RA_AIDs[i] = result[i].getName();
        addBehaviour(new retrieve_file());
    }
} catch (FIPAException fe) {
    System.out.println("Error--"+fe);
}

```

Appendix B.2.1: Broker Service search

```

//Initiate File Download Negotiation
public class retrieve_file extends OneShotBehaviour {
    @Override
    public void action() {
        try {
            ACLMessage request_message = new ACLMessage(ACLMessage.CFP);
            request_message.addReceiver(RA_AIDs[0]);
            request_message.setContentObject(myAgent.getAID());
            request_message.setConversationId("download-file");
            request_message.setReplyWith("request_message"+System.currentTimeMillis());
            myAgent.send(request_message);
            //Add behaviour to request File
            addBehaviour(new ReceiveFile());
        } catch (IOException ex) {
            System.out.println("Error--"+ ex );
        }
    }
}

```

Appendix B.2.2: Download Request

```

DFAgentDescription dfd = new DFAgentDescription();
dfd.setName(getAID());
ServiceDescription sd = new ServiceDescription();
sd.setType("BrokerService_downloads");
sd.setName("RA");
dfd.addServices(sd);

try {
    DFService.register(this, dfd);
}
catch (FIPAException fe) {
    System.out.println("Error--"+fe);
}

```

Appendix B.2.3: RA Service Registration

```

MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage receive_req = myAgent.receive(mt);
if (receive_req != null) {
    try {
        //Extract SCUA AID
        SCUA_AID = (AID) receive_req.getContentObject();
    } catch (UnreadableException ex) {
        System.out.println("Error-- " + ex);
    }
}

```

Appendix B.2.4: Extract SCUA AID

```

try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    //Select of Records in LocationsDatabase(LDA)
    select = "SELECT * FROM LocationsDatabase";
    try (ResultSet rs = stmt.executeQuery(select)) {
        locations = new ArrayList(); //A_list for SS mac addresses
        loc1filename = new ArrayList();//A_list for SourceFilename && SS_1 filename
        loc2filename = new ArrayList();//A_list for SourceFilename && SS_2 filename
        loc3filename = new ArrayList();//A_list for SourceFilename && SS_3 filename
        while (rs.next()) {
            //Check Field where "SCUA AID".equals(SCUA_AID.toString())
            if(rs.getString(1).equalsIgnoreCase(SCUA_AID.toString())){
                Sourcefilename = rs.getString(2);
                node1 = rs.getString(3);
                node1_filename= rs.getString(4);
                node2 = rs.getString(5);
                node2_filename = rs.getString(6);
                node3 = rs.getString(7);
                node3_filename = rs.getString(8);
                locations.add(node1);locations.add(node2);locations.add(node3);
                loc1filename.add(node1_filename);loc1filename.add(Sourcefilename);
                loc2filename.add(node2_filename);loc2filename.add(Sourcefilename);
                loc3filename.add(node3_filename);loc3filename.add(Sourcefilename);
            }
        }
    }
}

```

Appendix B.2.5: Parameters to Stored file

```

//A_list for SS mac addresses
//From identified SS mac_addresses in A_list(locations)-->derive SA AIDs
template1 = new DFAgentDescription();
sd = new ServiceDescription();
Location_SA = new AID[locations.size()];
int count = 0;
for (String location : locations) {
    sd.setType("storageagent" + location);
    template1.addServices(sd);
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template1);
        if(result.length > 0){
            Location_SA[count++] = result[0].getName();
        }
    }catch (FIPAException fe) {
        System.out.println("Error--"+fe);
    }
}

//Add Behavior to request for files from identified SS using SA AIDs
addBehaviour(new RequestPerformer1());
}catch(ClassNotFoundException | SQLException e){
    JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
}

```

Appendix B.2.6: Deriving SA AID from locations

```

//SA1 Request
file1_download = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file1_download.addReceiver(Location_SA[0]);
SS1_params = new ArrayList();
SS1_params.add(filename1.get(0));
SS1_params.add(filename1.get(1));
SS1_params.add(SCUA_AID);
file1_download.setContentObject(SS1_params);
file1_download.setConversationId("node_1_download");
file1_download.setReplyWith("file1_download"+System.currentTimeMillis());
myAgent.send(file1_download);

//SA2 Request
file2_download = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file2_download.addReceiver(Location_SA[1]);
SS2_params = new ArrayList();
SS2_params.add(filename2.get(0));
SS2_params.add(filename2.get(1));
SS2_params.add(SCUA_AID);
file2_download.setContentObject(SS2_params);
file2_download.setConversationId("node_2_download");
file2_download.setReplyWith("file2_download"+System.currentTimeMillis());
myAgent.send(file2_download);

//SA3 Request
file3_download = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
file3_download.addReceiver(Location_SA[2]);
SS3_params = new ArrayList();
SS3_params.add(filename3.get(0));
SS3_params.add(filename3.get(1));
SS3_params.add(SCUA_AID);
file3_download.setContentObject(SS3_params);
file3_download.setConversationId("node_3_storage");
file3_download.setReplyWith("file3_download"+System.currentTimeMillis());
myAgent.send(file3_download);

```

Appendix B.2.7: SA bound Requests

```

//Templates receive feedbacks(Source ACLMessage dependant)
file1_download_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_1_download"),MessageTemplate.MatchInReplyTo(file1_download.getReplyWith()))
file2_download_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_2_download"),MessageTemplate.MatchInReplyTo(file2_download.getReplyWith()))
file3_download_response = MessageTemplate.and(MessageTemplate.MatchConversationId
    ("node_3_storage"),MessageTemplate.MatchInReplyTo(file3_download.getReplyWith()));
node1_response = myAgent.receive(file1_download_response);
node2_response = myAgent.receive(file2_download_response);
node3_response = myAgent.receive(file3_download_response);
//Check if feedback satisfy performative and !=""
if (node1_response != null && !node1_response.getContent().equals("")
    && node1_response.getPerformative() == ACLMessage.INFORM) {
    try {
        downloadfeedback1 = (ArrayList) node1_response.getContentObject();
    } catch (UnreadableException ex) {System.out.println("Error--"+ex);}
} else{ System.out.println("Feedback is null");}
if (node2_response != null && !node2_response.getContent().equals("")
    && node2_response.getPerformative() == ACLMessage.INFORM) {
    try {
        downloadfeedback2 = (ArrayList) node2_response.getContentObject();
    } catch (UnreadableException ex) {System.out.println("Error--"+ex);}
} else{ System.out.println("Feedback is null");}
if (node3_response != null && !node3_response.getContent().equals("")
    && node3_response.getPerformative() == ACLMessage.INFORM) {
    try {
        downloadfeedback3 = (ArrayList) node3_response.getContentObject();
    } catch (UnreadableException ex) {System.out.println("Error--"+ex);}
} else{ System.out.println("Feedback is null");}

```

Appendix B.2.8: Receive Feedbacks

```

//Check SA feedbacks && if !=null
if( downloadfeedback1!=null && !downloadfeedback1.isEmpty()&&downloadfeedback2 !=null
    &&!downloadfeedback2.isEmpty()&&downloadfeedback3 !=null&&!downloadfeedback3.isEmpty()){
    //If the SCUA AID&&Sourcefilenames are equal;return file to requestor SCUA
    if(downloadfeedback1.get(1).equals(downloadfeedback2.get(1))&&downloadfeedback2.get(1)
        .equals(downloadfeedback3.get(1))&&downloadfeedback1.get(2).equals(downloadfeedback2.get(2))
        &&downloadfeedback2.get(2).equals(downloadfeedback3.get(2))){
        reply = new ACLMessage(ACLMessage.INFORM);
        reply.addReceiver((AID) downloadfeedback1.get(2));
        try {
            reply.setContentObject(downloadfeedback1);
            myAgent.send(reply);
            block();
        }catch (IOException ex){
            System.out.println("Error-- "+ex);
        }
    }
} else{
    block();
}

```

Appendix B.2.9: Return File to SCUA

B.3 C-AP File-System: Upload Service

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.CFP);
ACLMessage msg = myAgent.receive(mt);
if(msg != null) {
    try {
        recievedfile = (ArrayList) msg.getContentObject();
    } catch (UnreadableException ex) {
        System.out.println("Exception --"+ex);
    }
    file_upload = (byte[]) recievedfile.get(0); //retrive upload
    file_size = file_upload.length/(1024*1024); //calc uploadsize (MB)
    System.out.println("file size :"+file_size);
    SCUA_AID = (AID) recievedfile.get(1);
}
```

Appendix B.3.1: Handling SCUA Request

```
try {
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.37:3306"
        + "/maogp", "root", "toor");
    stmt=(Statement)con.createStatement();
    //select all shared resource information
    select="select * from nodeinfo";
    rs = stmt.executeQuery(select);
    mac_address = new ArrayList<>();
    while (rs.next()) {
        mac_address.add(rs.getString("mac_address"));
    }
    rs.close();
    template1 = new DFAgentDescription();
    sd = new ServiceDescription();
    sharednodes = new AID[mac_address.size()];
    int count = 0;
    for (String mac_address : mac_address) {
        sd.setType("storageagent" + mac_address);
        template1.addServices(sd);
    }
    try {
        DFAgentDescription[] result = DFService.search(myAgent, template1);
        if(result.length > 0){
            sharednodes[count++] = result[0].getName();
        }
    } catch (FIPAException fe) {System.out.println(fe);}
    }
    catch (ClassNotFoundException | SQLException e){
        JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
    }
    }
    reply = msg.createReply();
    chunkserver_func(file_upload,file_size);
}
```

Appendix B.3.2: Selecting Shared SS

```

if(file_size<=10){
    addBehaviour(new OneShotBehaviour(){
        public void action(){
            try{
                chunk_A1_P = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                chunk_A1_P.addReceiver(sharednodes[0]);
                chunk_A1_P.setByteSequenceContent(chunked_files[0]);
                chunk_A1_P.setConversationId("append_chunks1");
                chunk_A1_P.setReplyWith("chunk_index1"+System.currentTimeMillis());
                myAgent.send(chunk_A1_P);
                starttime =System.currentTimeMillis();
                System.out.println("-----7");
                addBehaviour(new receive_feedback());
            }catch(Exception ex){
                System.out.println("node<=10mb--- "+ex);
            }
        }
    });
}
else if(file_size>10&&file_size<=20){
    addBehaviour(new OneShotBehaviour(){
        public void action(){
            try{
                chunk_A1_P = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                chunk_A2_P = new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                chunk_A1_P.addReceiver(sharednodes[0]);
                chunk_A2_P.addReceiver(sharednodes[1]);
                chunk_A1_P.setByteSequenceContent(chunked_files[0]);
                chunk_A2_P.setByteSequenceContent(chunked_files[1]);
                chunk_A1_P.setConversationId("append_chunks1");
                chunk_A2_P.setConversationId("append_chunks2");
                chunk_A1_P.setReplyWith("chunk_index1"+System.currentTimeMillis());
                chunk_A2_P.setReplyWith("chunk_index2"+System.currentTimeMillis());
                myAgent.send(chunk_A1_P);
                myAgent.send(chunk_A2_P);
                starttime =System.currentTimeMillis();
                addBehaviour(new receive_feedback());
            }catch(Exception e){
                System.out.println("node<=20mb--- "+e);
            }
        }
    });
}

```

Appendix B.3.3: Selecting a node


```

//Executes when file<=10MB
if(file_size<=10){
    mt1 = MessageTemplate.and(MessageTemplate.MatchConversationId("append_chunks1"),
        MessageTemplate.MatchInReplyTo(chunk_A1_P.getReplyWith()));
    chunk_A1_P_feedback = myAgent.receive(mt1);
    if(chunk_A1_P_feedback.getPerformative() == ACLMessage.INFORM&&chunk_A1_P_feedback!=null){
        try{
            chunk1A= (ArrayList) chunk_A1_P_feedback.getContentObject();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent("upload successful");
            myAgent.send(reply);
            savechunklocations(SCUA_AID,chunk1A,chunk1B,chunk1C,chunk1D,chunk1E,chunk1F);
        }catch (UnreadableException | NullPointerException ex) {
            System.out.println("Exception " + ex);
        }
    }
}

//Executes when file>10&&file_size<=20
else if(file_size>10&&file_size<=20){
    mt1 = MessageTemplate.and(MessageTemplate.MatchConversationId("append_chunks1"),
        MessageTemplate.MatchInReplyTo(chunk_A1_P.getReplyWith()));
    mt2 = MessageTemplate.and(MessageTemplate.MatchConversationId("append_chunks2"),
        MessageTemplate.MatchInReplyTo(chunk_A2_P.getReplyWith()));
    chunk_A1_P_feedback = myAgent.blockingReceive(mt1);
    chunk_A2_P_feedback = myAgent.blockingReceive(mt2);
    if(chunk_A1_P_feedback!=null&&chunk_A1_P_feedback.getPerformative()==ACLMessage.INFORM
        &&chunk_A2_P_feedback!=null&&chunk_A2_P_feedback.getPerformative()==ACLMessage.INFORM) {
        try{
            chunk1A= (ArrayList) chunk_A1_P_feedback.getContentObject();
            chunk1B= (ArrayList) chunk_A2_P_feedback.getContentObject();
            reply.setPerformative(ACLMessage.INFORM);
            reply.setContent("upload successful");
            myAgent.send(reply);
            savechunklocations(SCUA_AID,chunk1A,chunk1B,chunk1C,chunk1D,chunk1E,chunk1F);
            block();
        }catch (UnreadableException |NullPointerException ex) {
            System.out.println("Exception " + ex);
        }
    }
}
}

```

Appendix B.3.4: Append Feedbacks

```

public void savechunklocations(AID SCUA_AID,ArrayList chunk1A,ArrayList chunk1B,ArrayList chunk1C,
    ArrayList chunk1D,ArrayList chunk1E,ArrayList chunk1F){
    try{
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp", "root", "toor");
        stmt=(Statement)con.createStatement();
        insert="INSERT INTO fileappendchunklocations VALUES ('"+SCUA_AID+"','"+chunk1A+"'"
            + ",'" +chunk1B+"','"+chunk1C+"','"+chunk1D+"','"+chunk1E+"'"
            + ",'" +chunk1F+"')";
        stmt.executeUpdate(insert);
    }catch (ClassNotFoundException | SQLException e){
        JOptionPane.showMessageDialog(null, e.getMessage() ,"Error", 1);
    }
}
}

```

Appendix B.3.5: Savechunklocations function

B.4 C-AP File-System: Download Service

```
MessageTemplate mt = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);
ACLMessage msg = myAgent.receive(mt);
if (msg != null) {
    try{
        //Receive requestor SCUA AID here
        SCUA_AID = (AID) msg.getContentObject();
    }catch (UnreadableException ex){
        System.out.println("Error-- "+ex);
    }
    reply = msg.createReply();
try{
    Class.forName("com.mysql.jdbc.Driver");
    con = DriverManager.getConnection("jdbc:mysql://172.20.56.41:3306/maogp"
        , "root", "toor");
    stmt=(Statement)con.createStatement();
    insert = "SELECT * FROM fileappendchunklocations";
    ResultSet rs = stmt.executeQuery(insert);
    while (rs.next()){
        if(rs.getString(1).equalsIgnoreCase(SCUA_AID.toString())){
            //return for each "[00-23-8B-52-6E-65, chunkfile, 24-8419848]"
            try{
                chunk1A = rs.getString(2);
                chunk1B = rs.getString(3);
                chunk1C = rs.getString(4);
                chunk1D = rs.getString(5);
                chunk1E = rs.getString(6);
                chunk1F = rs.getString(7);
            }catch(Exception e){
                System.out.println("Errors here :"+ e);
            }
        }
    }
}
```

Appendix B.4.1: Chunk Locations

```
try{
    //Split [00-23-8B-52-6E-65, chunkfile, 24-8419848] into 3 Arrays
    //token[0] =mac_address;token[1]=chunkfile_name;token[2]=byteRange
    //e.g. String Concatenate = "[00-23-8B-52-6E-65, chunkfile, 24-8419848]";
    //define the break points here // ","
    String delimiters = "[\\[, \\]]+";
    Chunk1A_Tokens = chunk1A.split(delimiters);
    Chunk1B_Tokens = chunk1B.split(delimiters);
    Chunk1C_Tokens = chunk1C.split(delimiters);
    Chunk1D_Tokens = chunk1D.split(delimiters);
    Chunk1E_Tokens = chunk1E.split(delimiters);
    Chunk1F_Tokens = chunk1F.split(delimiters);
}catch(NullPointerException ex){
    System.out.println("Error -- "+ex);
}
```

Appendix B.4.2: Chunk Arrays

```

try{
    if(!Chunk1A_Tokens[1].equals("")){
        template1 = new DFAgentDescription();
        chunk1_retrieve = new ServiceDescription();
        chunk1_retrieve.setType("storageagent"+Chunk1A_Tokens[1]);
        template1.addServices(chunk1_retrieve);
        try{
            DFAgentDescription[] result = DFService.search(myAgent, template1);
            chunk1_storage= new AID[result.length];
            for(int i = 0; i < result.length; ++i){
                chunk1_storage[i] = result[i].getName();
                addBehaviour(new OneShotBehaviour() {
                    public void action() {
                        try {
                            chunk1= new ACLMessage(ACLMessage.ACCEPT_PROPOSAL);
                            chunk1.addReceiver(chunk1_storage[i]);
                            chunk1.setContentObject(Chunk1A_Tokens);
                            chunk1.setConversationId("append_chunks1");
                            chunk1.setReplyWith("chunk_index1"+System.currentTimeMillis());
                            myAgent.send(chunk1);
                            //call Receive feedbacks behavior
                            addBehaviour(new Chunk1_Byte_Retrieve());
                        } catch (IOException ex) {
                            Logger.getLogger(RA.class.getName()).log(Level.SEVERE, null, ex);
                        }
                    }
                });
            }
        } catch (FIPAException ex){
            System.out.println("Error-- "+ex);
        }
    }
}

```

Appendix B.4.3: SA Request

```

mt1 = MessageTemplate.and(MessageTemplate.MatchConversationId("append_chunks1"),
    MessageTemplate.MatchInReplyTo(chunk1.getReplyWith()));
mt2 = MessageTemplate.and(MessageTemplate.MatchConversationId("append_chunks2"),
    MessageTemplate.MatchInReplyTo(chunk2.getReplyWith()));
chunk_A1_P_feedback = myAgent.blockingReceive(mt1);
chunk_A2_P_feedback = myAgent.blockingReceive(mt2);
if(chunk_A1_P_feedback!=null&&chunk_A1_P_feedback.getPerformative()==ACLMessage.INFORM
&&chunk_A2_P_feedback!=null&&chunk_A2_P_feedback.getPerformative()==ACLMessage.INFORM){
    try {
        chunk1A=(byte[]) chunk_A1_P_feedback.getContentObject();
        chunk1B=(byte[]) chunk_A2_P_feedback.getContentObject();
        byte[] combined_file = new byte[chunk1A.length + chunk1B.length];
        System.arraycopy(chunk1A,0,combined_file,0,chunk1A.length);
        System.arraycopy(chunk1B,0,combined_file,chunk1A.length,chunk1B.length);
        reply.setPerformative(ACLMessage.INFORM);
        reply.setByteSequenceContent(combined_file);
        myAgent.send(reply);
    } catch (UnreadableException | NullPointerException ex) {
        System.out.println("Exception-- " + ex);
    }
}
}

```

Appendix B.4.4: Reconstructing a File