An agent based approach towards autonomic services

Leong, Khai Weng

2009

Leong, K. W. (2009). An agent based approach towards autonomic services. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/14959>

<https://doi.org/10.32657/10356/14959>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**AN AGENT BASED APPROACH TOWARDS
AUTONOMIC SERVICES**

**LEONG KHAI WENG
SCHOOL OF COMPUTER ENGINEERING
2009**

AN AGENT BASED APPROACH TOWARDS AUTONOMIC SERVICES

LEONG KHAI WENG

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2009

An Agent Based Approach Towards Autonomic Services

Acknowledgements

I would like to acknowledge my supervisors for their invaluable advice and direction during the course of this research.

Supervisors:

- Dr Miao Chunyan (SCE,NTU)
- Dr Francis Lee Bu Sung (SCE, NTU)

I also wish to acknowledge the support from the senior management team, School of Media and Info-Communications Technology, Singapore Polytechnic

Finally, I would like to thank family members and friends for their understanding and encouragement during this period.

An Agent Based Approach Towards Autonomic Services

Table of Contents

CHAPTER 1	INTRODUCTION	1
1.1	Motivation.....	1
1.2	Research Challenges	5
1.3	Outline of Thesis.....	6
CHAPTER 2	LITERATURE SURVEY.....	7
2.1	Theoretical Background.....	7
2.1.1	Service-Oriented Computing (SOC).....	7
2.1.2	Service Oriented Architecture and Design	8
2.1.3	Service Engineering.....	12
2.1.4	SOC in Open Dynamic Environments.....	14
2.1.5	Agents and Multiagent Systems	15
2.1.6	Agent models	19
2.1.7	Goal-Oriented Agent Modelling.....	20
2.1.8	Agent architectures	23
2.1.9	Agent implementations	29
2.1.10	Autonomic Computing and Agents	31
2.2	Most Related	31
2.2.1	Autonomic Systems and Services.....	31
2.2.2	Agents and Autonomic Services.....	36
2.2.3	Agent Oriented Software Engineering for SOC	37
2.2.4	Software Engineering for Agents.....	38
2.2.5	Agent-mediated Services	39
2.3	Summary.....	40
CHAPTER 3	FCGN AGENT THEORY AND APPROACH.....	41
3.1	FCM, DCN and Behaviour Networks.....	41
3.1.1	Fuzzy cognitive map.....	41
3.1.2	Goal Net.....	43
3.2	Fuzzy and goal-oriented modelling	47
3.3	FCGN: Goal Net with FCM.....	49
3.3.1	FCGN MAS	50
3.3.2	FCGN MAS Behaviour	52
3.4	Fuzzy goal selection mechanism	53
3.5	FCM driven action selection mechanism.....	58
3.6	Mathematical Representation	60
3.6.1	Goal Net and Agent	60

An Agent Based Approach Towards Autonomic Services

3.6.2	Fuzzy Cognitive Goal Net and Agent	63
3.7	Expressiveness and Limitations of FCGN Model	66
3.8	Summary	67
CHAPTER 4 FCGN AGENT DESIGN AND IMPLEMENTATION		68
4.1	Overview	68
4.2	FCGN Agent Implementation	69
4.3	FCGN Agent Execution Framework	71
4.4	Simplified Goal net Markup Language (SGNML)	71
4.5	FCM Markup Language (FCMML)	73
4.6	Modules	74
4.7	Agent Implementation Platforms	74
4.8	FCGN Agents on JADE	79
4.9	Summary	88
CHAPTER 5 GOFAAS – GOAL-ORIENTED AND FCM FOR AGENT-MEDIATED AUTONOMIC SERVICES METHODOLOGY		89
5.1	Overview	89
5.2	Phases	90
5.3	Analysis	93
5.4	Specification	103
5.4.1	Agent Model	103
5.4.2	Interactions	105
5.5	Architecture	108
5.5.1	Mapping and Partitioning	108
5.5.2	Patterns	109
5.6	Design	112
5.7	Implementation	114
5.8	Testing and Verification	115
5.9	GOFASS Modelling Language (GOFASSML)	116
5.9.1	Actor	118
5.9.2	Goal	119
5.9.3	Plan	121
5.9.4	FCGN Goal	122
5.9.5	FCGN Transition	122
5.9.6	FCGN Arc	123
5.9.7	FCGN Event, Percepts and Route	124
5.10	Worked Example	125

An Agent Based Approach Towards Autonomic Services

5.10.1	Motivating Scenario.....	125
5.10.2	Detailed Worked Example.....	127
5.11	Summary.....	135
CHAPTER 6	EVALUATION	136
6.1	Evaluation of AOSE	136
6.1.1	Comparison of AOSE evaluation techniques	136
6.1.2	Proposed Evaluation of GOFASS.....	138
6.2	Metrics Based Evaluation	146
6.3	Case Studies and Survey Evaluation	148
6.3.1	Agent-Mediated Data Exchange (ADE)	148
6.3.2	Analysis, Specification and Architecture.....	150
6.3.3	Implementation and Evaluation	158
6.3.4	Mobile Digital Concierge (MDC).....	170
6.3.5	Analysis, Specification and Architecture.....	176
6.3.6	Implementation and Evaluation	186
6.3.7	Discussion.....	190
6.4	Comparison of Evaluations with Related Work	192
6.5	Summary.....	193
CHAPTER 7	CONCLUSIONS AND FUTURE WORK.....	194
7.1	Conclusions.....	194
7.1.1	GOFASS provides AOSE for services	196
7.1.2	GOFASS leads towards autonomic services	197
7.1.3	GOFASS leads towards practical implementations.....	197
7.2	Future work.....	199
7.2.1	U-Learning Agent Application	199
7.2.2	GOFASS for Digital Ecosystems	200
7.2.3	Emergent Behaviours.....	201
7.2.4	Other Evaluation Techniques.....	203
REFERENCES		205
APPENDIX A	AUTHOR'S PUBLICATIONS	223
APPENDIX B	SURVEY QUESTIONS & DATA SAMPLES	225
APPENDIX C	MATHEMATICAL BACKGROUND.....	233

An Agent Based Approach Towards Autonomic Services

List of Figures

Figure 2.1. Relationship between Services, Components and Objects	8
Figure 2.2. The layers of a SOA	10
Figure 2.3. Relationship between Business, Services and Components.....	11
Figure 2.4. Logical and Physical Service Development Life-Cycle.....	14
Figure 2.5. Three Level Model	28
Figure 2.6. Autonomic manager	32
Figure 2.7. Autonomic network	34
Figure 3.1. Goal net model elements.....	44
Figure 3.2. Agent knowledge model.....	51
Figure 3.3. Goal Net, Percepts and Events	56
Figure 3.4. FCM mapping percepts to goals.....	57
Figure 3.5. FCM for action selection	59
Figure 4.1. The FCGN agent architecture.....	70
Figure 4.2. SGNML	72
Figure 4.3. FCMML	73
Figure 4.4. FCGN Agent-JADE Integration.....	80
Figure 4.5. FCGN Execution Model in Adapter Agent	82
Figure 4.6. FCGN Agent Capability Meta-model	83
Figure 4.7. Web Service Integration Gateway.....	84
Figure 4.8. JADE and P2P Interface.....	87
Figure 5.1. Transformation of models in methodology.....	90
Figure 5.2. Example of Use Case.....	95
Figure 5.3. Agent Identification from Use Case	96
Figure 5.4. Example of GET card	97
Figure 5.5. Tropos Notations.....	99
Figure 5.6. Tropos Actor Notations	99
Figure 5.7. GOFASS Actor Diagram Notation.....	101
Figure 5.8. GOFASS Goal Diagram Notation.....	102
Figure 5.9. GOFASS FCGN Diagram.....	104
Figure 5.10. Higher Level FCGN	105
Figure 5.11. Interaction Diagram	107
Figure 5.12. Actor Concept.....	118
Figure 5.13. Goal Concept (Decomposition Relationship)	120

An Agent Based Approach Towards Autonomic Services

Figure 5.14. Goal Concept (Means-Ends Relationship)	120
Figure 5.15. Goal Concept (Contribution Relationship)	121
Figure 5.16. Plan Concept	122
Figure 5.17. Fuzzy Cognitive Goal Net (FCGN) Concept.....	124
Figure 5.18. Events, Percepts and Routes Concept	125
Figure 5.19. Application scenario for FCGN MAS	126
Figure 5.20. Agent knowledge model for scenario 1	127
Figure 5.21. Use Case based on scenario 1	128
Figure 5.22. Agent Identification based on scenario 1	129
Figure 5.23. Goal Diagram for scenario 1	130
Figure 5.24. FCGN Diagram based on scenario 1	131
Figure 5.25. Sub-goals of g1 and agents	132
Figure 5.26. Mapping FCGN Agent Design To Implementation	134
Figure 6.1. Closed group bidding for product	150
Figure 6.2. Use cases for mobile user	151
Figure 6.3. Use cases for user agent.....	152
Figure 6.4. ADE GET Card	154
Figure 6.5. ADE Goal Diagram	155
Figure 6.6. ADE Agent Identification	156
Figure 6.7. ADE Fuzzy Cognitive Goal Net.....	156
Figure 6.8. ADE Fuzzy Cognitive Goal Net Refinement.....	157
Figure 6.9. ADE FCM	158
Figure 6.10. Pure P2P Architecture	160
Figure 6.11. Hybrid P2P Architecture	160
Figure 6.12. Mobile proxy	161
Figure 6.13. SOA Layers	162
Figure 6.14. Mobile P2P Data Exchange Prototype.....	163
Figure 6.15. Hierarchy of Autonomic Managers.....	165
Figure 6.16. Distributed MAPE autonomic elements.....	166
Figure 6.17. P2P Network of Autonomic Agents.....	167
Figure 6.18. Structured P2P Network (Logical Ring)	168
Figure 6.19. ADE Simulation	169
Figure 6.20. ADE Simulation Model	170
Figure 6.21. Map navigation on the mobile device	173

An Agent Based Approach Towards Autonomic Services

Figure 6.22. Example of multiple hotspot types on the map	173
Figure 6.23. Example of single hotspot type after filtering	174
Figure 6.24. Use case diagram of agent architecture in a network	176
Figure 6.25. Activity diagram of agent architecture	177
Figure 6.26. Use case diagram of the location-based mapping service	179
Figure 6.27. Activity diagram of the overall Location-Based Mapping Service	180
Figure 6.28. Use case diagram of the Information Service(s)	181
Figure 6.29. Use case diagram of the Promotions and Events service	182
Figure 6.30. MDC GET Card.....	183
Figure 6.31. MDC Goal Diagram	184
Figure 6.32. MDC Fuzzy Cognitive Goal Net	184
Figure 6.33. MDC Agent Identification.....	185
Figure 6.34. Block Diagram for the Mobile Digital Concierge	187
Figure 6.35. System Diagram for the Mobile Digital Concierge.....	188
Figure 6.36. MDC Simulation	189
Figure 6.37. MDC Simulation Model	190
Figure 6.38. Graph comparison of ratings.....	191
Figure 6.39. Graph comparison of ratings (per question).....	192

An Agent Based Approach Towards Autonomic Services

List of Tables

Table 2.1. Common agent attributes.....	16
Table 2.2. Goal Driven Action Selection Mechanisms	23
Table 2.3. Agent architectures	25
Table 2.4. Agent architecture characterization.....	26
Table 2.5. Comparison of JADE, RETSINA, Bee-gent	29
Table 2.6. Autonomic Computing System Concepts.....	33
Table 2.7. Autonomic Computing Research Projects	34
Table 2.8. AOSE methodologies	38
Table 3.1. Arc Relationships	45
Table 3.2. Transition types	46
Table 3.3. Fuzzy Cognitive Goal Net UML Stereotypes	48
Table 4.1. FCGN Execution Model Implementation Elements	81
Table 5.1. GOFAAS Overview	91
Table 5.2. Comparison of Analysis Phase	93
Table 5.3. Analysis GOFAAS UML Stereotypes.....	100
Table 5.4. Design Phase Artifacts and Processes.....	113
Table 5.5. GOFASS Model Levels	116
Table 5.6. Capabilities Table.....	133
Table 6.1. Attributes Tree Model	142
Table 6.2. Metrics Evaluation Results	146
Table 6.3. ADE Autonomic Characteristics.....	153
Table 6.4. Comparison of P2P networks.....	167
Table 6.5. MDC Autonomic Characteristics.....	182
Table 6.6. Normalized Ratings.....	191
Table 7.1. Other Evaluation Techniques.....	203

An Agent Based Approach Towards Autonomic Services

Summary

Services play an increasingly important role in software applications today. There are increasing demands to build/compose software as a collection of services. A service in this context may be defined as a behaviour that is provided by a component for use by any other component based on a network-addressable interface contract. Such service oriented software raises some challenges when designed using current object-oriented methodologies in situations where there is a need to trade-off competing goals, where there are complex business workflows, or when service execution is highly dynamic. Multi-Agent Systems (MAS) fits well into complex services, but current Agent-Oriented Software Engineering (AOSE) methodologies do not address the aforementioned challenges of services. AOSE for services that operate in open, dynamic and complex environments such as Grids is a major problem that needs to be addressed.

This thesis describes an AOSE methodology that is particularly suited for building service oriented software. It has the advantages of being accessible and pragmatic. The thesis presents a novel Goal-Oriented and FCM for Agent-mediated Autonomic Services (GOFASS) methodology used to specify agents for the agent-mediated autonomic services architecture. The GOFASS methodology provides guidance from early requirements gathering through to agent design and implementation. GOFASS provides a complete set of guidelines for each phase of the development of complex services. GOFASS also provides sets of models, diagrams and notations that fit into the Unified Modelling Language (UML) 2.0 superstructure as a UML 2.0 Profile. The advantage of leveraging on UML is that it increases the accessibility of GOFASS to current software developers through the use of a familiar notation. To bridge the gap between abstract design and concrete implementation, we

An Agent Based Approach Towards Autonomic Services

implemented a framework for agent-mediated services with autonomic behaviour. In our framework, we proposed a new type of fuzzy cognitive goal net (FCGN) agent suited to autonomic services. This new type of agent shows fast fuzzy inference together with goal-directed behaviour. This thesis describes the design of the framework and how GOFASS utilizes FCGN agent capabilities to design-in and implement required autonomic behaviours.

Our case studies and prototypes have shown the proposed methodology is practical. Using GOFASS, two prototypes were created to evaluate the effectiveness of the FCGN agent framework in real world. The results show that our AOSE methodology is well suited for the practical implementation of complex services in open dynamic environments.

CHAPTER 1 INTRODUCTION

Services play an increasingly important role in software applications today. It has a role as a unifying concept across many different application domains; and promises better alignment between business processes and IT systems. Service-Oriented Computing (SOC) increases the speed of system development through loose coupling between system components, discovery of services and consolidating multiple heterogeneous services.

1.1 Motivation

Service-Oriented Computing (SOC) is a new paradigm in computing that uses services as the fundamental unit to support the development of cost-effective, distributed applications, and integrated services in heterogeneous environments. The grand vision of SOC is applications that can be easily assembled, dynamically from an available pool of cooperative services to enable agile business processes and on-demand computing services. SOC represents an evolution of system development techniques from function oriented, to data-oriented, to object-oriented, to interface-oriented, to component based, to aspect oriented; and finally to service-oriented.

The “service oriented” approach is independent of the programming language and operating system. Services are self-contained/autonomous functionalities that are exposed to service users through well-defined published interfaces; which can be discovered and bound by the service users. Services reflect an approach to programming that emphasizes concept loosely-coupled systems and flexible construction of a new application through assembly or composing of services, rather than invoking existing application component methods or linking to pre-compiled component libraries.

However, conventional SOC implementations are inflexible, vulnerable and inefficient in complex and dynamic processing. Conventional SOC systems are built using a client-server approach using relatively static pre-selected and contracted service providers. The client-server approach relies on major blocks of functionality being deployed on server machines. The server hardware represents potential single point of failures. The failure of a critical server can potentially stop the services provided completely.

However, service deployment in distributed infrastructure such as a cluster or grid also increases the complexity of the system. Within a grid or virtualized service infrastructure, the service execution environment allocating resources on-demand dynamically and service workflows would be correspondingly more complex. Services can be combined or consolidated into more complex services from simpler basic services. In conventional SOC systems, the services providers are pre-selected and contracted through a long and manual process.

In dynamic situations, for example where service providers may be added and removed during the lifetime of a service, the conventional approach would not suffice. WSBPEL and BPEL4WS allow the service workflows in the system to be configured via XML; which it easier to re-configure the service workflow. But, WSBPEL and BPEL4WS are relatively static in terms of their process execution logic. They would not be able to dynamically change their orchestration logic or business flow if there are new service providers or critical services fail.

Lastly, complex failure scenarios with multiple failure points and complex recovery paths require human intervention. Self-management or autonomic behaviour is not possible because the various service components lack the required autonomy. SOC in complex and dynamic environments needs to rely on agent-mediated functions to help them overcome the limitation of conventional techniques.

The limitation of conventional techniques for SOC in complex and dynamic environments can be overcome by using agent-mediated functions. Agent-mediated functions include trust negotiation, autonomic behaviours, semantics-driven service matching, dynamic composition, and goal-directed behaviours. By capitalizing on their individual technical strengths, a combination of both services-based and agent-based software engineering may lead towards better approaches in creating large-scale distributed service-oriented software systems.

Over the past few years, there have been several research efforts on the application of software agents to autonomic systems. IBM Research [Bigus et al., 2002] released ABLE [ABLE, 2007], a toolkit for building multiagent autonomic systems. Some of the limitations in prior work are: 1) the lack of fast and lightweight inference mechanisms for the agent framework, 2) the lack of fuzziness in goal representation and 3) gap between abstract models and practical implementations. Our research builds upon and extends previous research efforts in goal-autonomous agents [Shen et al., 2004]. Specifically, we will apply agency with goal-net and FCM models for creating autonomic services. The existing methodology for goal-autonomous agents [Shen et al., 2004a] was extended for the new problem domain.

With the rise of internet based systems and mushrooming wireless devices, our present day information infrastructure is suffering from burgeoning complexity that is fast out-stripping our human ability to manage the infrastructure effectively. Increasingly the only viable solution is seen as having the systems themselves exhibit self-management capabilities or autonomic characteristics [Patterson et al., 2002; Fox and Patterson, 2003]. Such self-managing computer systems are referred to as autonomic computing systems (ACS). Autonomic computing (AC) [Horn, 2001; Kephart, 2003] is still a nascent area of research with many open research issues. However, it is viewed as the only viable option in dealing with the mushrooming

complexity of information systems. Fundamentally, autonomic computing is about self-management; relieving the tasks of managing and regulating the information infrastructure from human administrators. Autonomic systems draw inspiration from natural biological regulatory systems such as the human autonomic nervous systems. The autonomic nervous system regulates low-level and vital functions such as the body temperature and heart-rate without burdening the brain. For the advancement and deployment of AC, there is a pressing need for the adoption of software engineering methodologies for improving autonomic systems. In this respect, agent-based software engineering (AOSE) holds the most promise because of the inherent flexibility and pro-active nature of software agents.

In the last few years, the agent research community has been advocating the use of agents as the next generation software development model. Agent techniques are suited for creating complex distributed software [Jennings, 2000]. Agent techniques have also been actively researched in the areas of grid computing [Agarwal et al., 2003] and autonomic computing systems [Poellabauer et al., 2003]. Yet despite this increasing interest, practical agent systems are still uncommon. Several researchers have identified what they termed as a “gap” [Sudeikat et al, 2004] between the conceptual agent models and practical implementation architectures. Agent modelling and agent methodology for practical use still need to be researched and refined.

To address the limitations we have identified with current SOC and meet the challenges posed by a dynamic, virtualized service infrastructure, we investigated methods for agent-mediated services to enable autonomic behaviour. The hybrid of the Goal-Net (GN) model and Fuzzy Cognitive Map (FCM) model results in the development of a new type of fuzzy cognitive goal net (FCGN) agent suited to autonomic services. FCGN agents rely on lightweight mathematical computations instead of symbolic manipulations for their inference mechanism and can be

implemented in resource-limited devices such as personal digital assistants (PDAs) and Smart-Phones. The combination of services with a lightweight FCGN agent framework allows for the creation of autonomic services for a variety of application domains. To achieve this, a number of research challenges need to be surmounted.

1.2 Research Challenges

Autonomic computing is a multi-faceted approach to ensuring the continued growth of computing systems beyond the present era of high management complexities and costs. To make this vision a reality, critical components needed include an autonomic AOSE methodology for designing autonomic services and practical implementation architecture for the autonomic agent-based system. In this research our challenges are follows:

- Investigate a new agent model which extends Goal Net [Shen, 2005] by including fuzzy goals and a FCM reasoning mechanism. Applying the new FCGN agent model to enable autonomic characteristics.
- Develop a new AOSE methodology for autonomic services. The AOSE must bridge the gap between modelling and implementation.
- Demonstrate the practicality of the new AOSE methodology to different problem domains. We intend to investigate several case studies to explore the advantages and shortcomings of the new methodology in real world applications.

In this research, we will characterize the agents that demonstrate autonomicity. We propose a new agent model that is a hybrid of the goal-net model and fuzzy cognitive networks. This new model should exhibit goal and behaviour autonomy

while retaining the fast action selection required for dynamic open situations. We will also develop a new AOSE methodology that addresses autonomic computing. This new methodology will bridge the gap between modelling and implementation.

1.3 Outline of Thesis

This thesis is divided into the following chapters. We start the first chapter with an introduction to the research topic. Chapter 2 deals with the theoretical background and related works. Chapter 3 defines the new FCGN agent model proposed. Chapter 4 is concerned with design and implementation. Chapter 5 explains the FCGN agent methodology. Chapter 6 describes the evaluation of the methodology. Chapter 7 concludes with future work and detailed plans for future research.

CHAPTER 2 LITERATURE SURVEY

This chapter gives the theoretical background for our research. This chapter also details some of the related work which our research builds upon or extends.

2.1 Theoretical Background

The following sub-sections give the theoretical background to our work.

2.1.1 Service-Oriented Computing (SOC)

Service orientation uses services as the building blocks for rapid development of distributed applications. A service oriented design and development methodology focuses on business processes, which it considers to be reusable elements that are platform and programming language independent. The entire enterprise is viewed as a network of inter-related services connected through well-specified interfaces that could be published and discovered.

Definition 2.1: *Service-Oriented Computing (SOC) is a model for distributed computing that utilizes services as fundamental elements to enable building networks of collaborating applications distributed within and across organizational boundaries. Services are self-contained, platform-independent computational elements that can be described, published, discovered, orchestrated and deployed.*

2.1.2 Service Oriented Architecture and Design

The system architecture required to support service orientation is the Service-Oriented Architecture (SOA). Older software development paradigms such as Object-Oriented Programming (OOP) and Component-Based Development (CBD) have limitations in supporting SOA because they do not address the key elements of SOA such as services, service assemblies (compositions), components realizing services, and business flows. Figure 1 shows the relationship between OOP, CBD and Services.

Figure 2.1. Relationship between Services, Components and Objects

Many early SOA systems were built through the addition of a service-oriented wrapper layer around existing pre-built software components and applications. Since component methodologies focused on the interface, developers assumed that these methodologies could also be applied equally well to service-oriented system. The use of a thin Simple Object Access Protocol (SOAP) [SOAP, 2007] / Web Services Description Language (WSDL) [WSDL, 2007] / Universal Description Discovery and Integration (UDDI) [UDDI, 2007] layer over existing software components and

applications has become a practice in industry. However, many existing components are not designed to be re-usable services. Thus, the current practice does not result in easily composed services or loose-coupling; which attenuates the benefits which can be derived from service orientation. Components and services present differences in terms of type of communication, type of invocation, type of interface, and type of request brokering. They are also fundamentally different in the way they approach reusability and flexibility. Service selection is often done in a policy-driven manner. Services may be composed or assembled from a collection of services. Service provider implementation are continuously maintained and improved. Installed software component do not support the same kind of reuse or dynamic behaviour. In a distributed environment, component based software involves additional complexity in locating and invoking the distributed components.

The Service Oriented Analysis and Design (SOAD) [Arsanjani, 2004; Zimmerman et al., 2004] focuses on using existing methodologies such as OOAD, BPM and complementing them with service-oriented modelling and patterns for middleware such as the Enterprise Service Bus (ESB). Existing modelling disciplines such as Object-Oriented Analysis and Design (OOAD) and Business Process Modelling (BPM) provide useful abstractions that are helpful in creating a SOA. However, because they are used in isolation from each other, they do not address SOA concerns such as service discovery, service compositions and service bus middleware patterns. OOAD focuses on micro level abstractions such as classes and object instances. Figure 2.2 shows the relationship between the business process, services, components and cross-cutting concerns in a SOA.

Presentation	Integration Architecture	QoS, Security, Management & Monitoring
Business Process Choreography		
Services		
Enterprise Components		
Operational Systems		

Figure 2.2. The layers of a SOA

The SOAD has to assist the software architect in defining a business-level view of the services, concentrating on defining the interfaces and service-level agreements (SLA) between the services. The SOAD has to extend BPM by providing design guidelines for deriving services and their interactions from process models. Rational Unified Process (RUP) is recognized as a leading OOAD process. When compared with SOAD, in RUP the architecture of the system consists of components interacting via interfaces. In contrast, SOAD focuses on business services that match the BPM models. The business services may be composed of other services which are orchestrated or choreographed to work together. SOAD implements additional layers of service-oriented abstraction over the RUP components. Figure 2.3 shows how a business function domain is mapped into business process, business services and underlying service implementation.

Figure 2.3. Relationship between Business, Services and Components

The Open Service Oriented Architecture [OSOA, 2007] collaboration web site has defined a model for SOA named Service Component Architecture (SCA) [SCA, 2007] and a uniform interface model for data named Service Data Objects (SDO). An open source implementation of the SCA and SDO [SDO, 2007] is available at the Apache Tuscany [Tuscany, 2007] web site. The SCA provides a simple model for creating systems based on the service oriented architecture. SCA organizes the application based on components that implement business logic, which provide their functionality through service-oriented interfaces and also able to consume the service provided by other components through service-oriented interfaces called service references. The creation of the service-oriented application is divided into 2 steps: (1) the implementation of the components which provide services and consume services, and (2) the assembly of the components through the wiring of service references to services.

SCA supports service components written in different programming languages. SCA

supports binding using a variety of access methods ranging from web services, to messaging, to CORBA IIOP. Bindings are configured declaratively and independent of implementation code. Security, transactions, reliable messaging, and other infrastructure capabilities are defined declaratively and separated from implementation code. Usage of the infrastructure capabilities is defined through the use of policies. The SCA is complemented by the SDO which provides a uniform interface to manipulate data from different data sources such as relational data bases, XML files, and web services. The SCA Assembly Model deals with the linking of components through wiring. The Assembly Model is independent of implementation language.

SCA provides a model both for the composition of services and for the creation of service components, including the reuse of existing application function within SCA compositions. An SCA composite is used to assemble SCA elements in logical groupings. It is the basic unit of composition within an SCA System. An SCA composite contains a set of components, services, references and the wires that interconnect them, plus a set of properties which can be used to configure components. Composites may form component implementations in higher-level composites – in other words the higher-level composites can have components that are implemented by composites. The content of a composite may be used within another composite through inclusion.

2.1.3 Service Engineering

Service engineering requires the identifying of suitable services that are aligned with the business goals. Some researchers have focused on goal-oriented requirements analysis and modelling for service identification. In these techniques, requirements analysis begins with the users' and stakeholders' goals. The goals are decomposed into lower-level goals. The satisfaction of the lower level goals contributes to the

satisfaction of the root-level goal. Services and their operations are identified to implement the goals.

The service development life-cycle for the service is divided into logical and physical parts (see Figure 2.4). During the logical part of the development life cycle, the focus is on the business domain, business processes and business services. The physical part of the service development life cycle comprises infrastructure services and component implementations that map logical services to existing resources. The infrastructure services are the “glue” that ties the logical business services to actual physical components. Infrastructure services comprise management and monitoring services. The infrastructure services enable the integration of services through intelligent routing, protocol mediation, and other transformation mechanisms (considered as part of the Enterprise Service Bus). The physical part of the development life-cycle also maps the business services to component implementations such as the financial and operational functions and data available from resources such as ERP, databases, CRM and other systems.

Figure 2.4. Logical and Physical Service Development Life-Cycle

2.1.4 SOC in Open Dynamic Environments

Both SOAD and SCA use declarative mechanisms to facilitate flexibility in reconfiguring the composite services and replacing service providers. However, in a dynamic and complex service environment, both of these have their limitations. Declarative mechanisms facilitate reconfiguration, but they require human intervention to select and contract new service providers or modifications to the business process flow. The declarative mechanism does not support autonomous re-negotiation of services by the system.

Secondly, complex failure scenarios cannot be recovered from without human intervention to re-route around failed components or to bring in additional replacement services. Hence, autonomic behaviour or self-management of the system would be difficult to support using conventional approaches to SOC.

Service composition requires human inputs for selection of service providers. Service orchestration is defined declaratively using WSBPPEL or BPPEL4WS which is

static and does not allow for changes to the process flow during run-time; even if there is sudden change in the environment. Conventional approaches do not support dynamic negotiation of service level agreements or semantic matching for service discovery.

One of the main challenges of SOC is the engineering of service compositions. SOA depends on the business process orchestration and choreography mechanism to discover services and execute the required services for the business process in a coordinated manner. Currently, WSBPEL is a leading XML based standard for orchestrating the business processes. With service orchestration, the business process is controlled centrally by one of the parties in the process. Service choreography is more distributed and collaborative in nature. The business process is executed through executing clearly defined rules of participation for collaboration. On the research front, some researchers have focused on dynamic compositions and modular compositions. Automatic retrieval and service composition is enabled through enhanced service descriptions. The service description is enhanced through techniques from semantic web services. There is some research into service composition mechanisms that are context-aware.

2.1.5 Agents and Multiagent Systems

Tracing the history of software modelling approaches, the first generation of structured, defined approaches to the creating software began with “Structure modelling” [Demarco, 78]. Object-oriented (OO) was proposed in the 1980s, and superseded structured modelling [Mellor and Johnson, 97; Rumbaugh, 1991]. OO decomposes the problem space into a hierarchy of classes and objects. These objects collaborate with each other through method invocations. Private state information and data is protected through encapsulation. Objects have control over their own state

information. However, objects do not have autonomous control over their behaviours since their behaviours are only activated upon invocation of the object's public methods by an external entity. Agents are a new approach to the software development complex distributed systems [Jennings, 2000].

Definition 2.5: *An agent is a **computer** system situated in some environment, and that is capable of **autonomous** action in this environment in order to meet its design objectives.* (Adapted from [Wooldridge and Jennings, 1995]).

A list of attributes (see Table 2.1) often used for agents are as follows: autonomous, goal-oriented, collaborative, flexible, self-starting, temporal continuity, emotional, communicative, adaptive, mobile.

Table 2.1. Common agent attributes

Attributes	Alternative Term	Meaning
reactive	sensing and acting; responsive	responds in a timely fashion to changes in the environment
autonomous		exercises control over its own actions
goal-oriented	pro-active; purposeful	does not simply act in response to the environment
temporally continuous		is a continuously running process
communicative	social	communicates with other agents, perhaps including people
learning	adaptive	changes its behaviour based on its previous experience
mobile		able to transport itself from one machine to another
flexible		actions are not scripted
emotional	believable	believable "personality" and emotional state.

Definition 2.6: *Mobile agents are agents that have, in addition to the other attributes, the capability to alter their execution location.*

Mobile agents are able to move from one host machine to another one. Mobile agents systems are inherently decentralized, concurrent and autonomous. These characteristics enable mobile agents to take advantage of a distributed processing environment such as the grid.

Definition 2.7: *A multi-agent system (MAS) is a computing system that comprises a number of individual agents working and interacting with each other [Sycara, 98; Wood and DeLoach, 2001].*

The individual agents within a MAS may employ different approaches or combinations of approaches in interacting with each other and distributed problem solving. The individual agents compete with other agents in the pursuit of their own self-interest goals. Individual agents may also co-operate with each other in order to fulfil common goals. Each individual agent may negotiate with other agents in order to co-ordinate their actions towards common goals.

A MAS can be designed in centralized or decentralized view [Xuan and Lesser, 2002]. By a centralized view, the whole status of a MAS system is changed along the time. Each agent is assumed to have global information of the system. The problem solving strategy is the jointed actions of the agents. In a decentralized view, each agent has its own local knowledge and makes decision based on the knowledge. The knowledge is updated by detecting local environment and communicating with other agents. The centralized view can simplify the system design and several methods are

proposed to obtain centralised policies. However, it must assume that each agent has global information and this is impossible for most real situations. In [Xuan and Lesser, 2002], Xuan and Lesser proposed a method to transform centralized policies into decentralized policies. TÆMS modelling language is used to build MAS [Vincent et al, 2000]. TÆMS is an extension of HTN (Hierarchical Task Network). In our realisation of MAS, decomposition of centralized policies to create decentralized policies is also adopted. We proposed an extension of Goal-net to build MAS. Goal-net is decomposed into subnets, which are implemented as agents.

Organization of MAS shows the relationship and connections among agents. As stated in a survey by Horling and Lesser [Horling and Lesser, 2005], from the view of organizational paradigm, MAS can be identified as hierarchies, holarchies, coalitions, teams, congregations, societies, federations, markets, and matrix. Our MAS examples shown in experiment chapter are organised as hierarchies. Agents are designed by decomposing Goal-net. Each agent has related up-level agents to connect with. An agent will pass its output to its up-level agents. A little difference from traditional hierarchies organization is that agents in same layer may communicate with each other.

One of the many practical applications for MAS is in Agent-based Grid computing [Foster et al., 2004] where the negotiation mechanisms and distributed problem solving approaches from MAS research help to solve some issues in Grid computing.

In an open distributed environment such as in grid computing, intelligent mobile agents have several advantages. Some of the strengths of mobile agents include the following:

- Conservation of bandwidth
- Reduction in total completion time
- Reduction in latency

- Disconnected operation and mobile computing
- Load balancing
- Dynamic deployment

None of these strengths are unique to mobile agents, but no competing technique shares all six. Mobile agents are a general-purpose framework for implementing distributed applications.

2.1.6 Agent models

A software model abstracts the basic characteristics of a computer system's programming in order to explain the behaviour of the system using a higher-level constructs. A software agent may be viewed from a variety of viewpoints which is each described by a different abstract model; for example, an agent model, a life-cycle model, a computational model, a security model, an interaction model and finally a mobility model.

Definition 2.9: *The agent model defines the internal structure of the intelligent agent part of an agent. In essence it defines the autonomy, learning and co-operative characteristics of an agent. Additionally, it specifies the reactive and proactive nature of agents.*

The following is a list of other important abstract models for a software agent:

- **Life-cycle model:** This model defines the different execution states of a mobile agent and the events that cause the movement from one state to another. It is closely related to the computational model which describes how the execution

occurs.

- **Computational model:** The computational model defines how a mobile agent executes. The computation takes place in an environment and is facilitated by some form of processor. A processor could be the CPU of a computer or a more abstract processor as can be found in the Java virtual machine.
- **Security model:** Mobile agent security can be split into two broad areas: 1) the protection of host nodes from destructive mobile agents, and 2) the protection of mobile agents from destructive hosts.
- **Interaction model:** Communication is used when accessing services outside of the mobile agent, during co-operation and co-ordination between mobile agents and other entities, and finally to facilitate competitive behaviour between self-interested agents. A protocol is an implementation of an interaction model.
- **Mobility model:** This model concerns itself with all aspects of agent mobility from the discovery and resolution, to the manner in which a mobile agent is transported.

2.1.7 Goal-Oriented Agent Modelling

In recent years, there has been an increased recognition that goal modelling is an important and effective mean of eliciting the requirements of agent based systems, as well as analyzing agent based systems [Anton, 1996; Anton, 1997; Kolp et al., 2001; Yu, 2001; Liu and Liu, 2002]. One view of agents (strong agency) requires that agents have mental attitudes such as beliefs, goals and intentions [Dennett, 1987]. The intentional stance takes the view that as systems become more complex, their behaviour can be predicted more easily by moving away from examining how it achieves its goals and focus instead on examining what are its goals and beliefs.

Definition 3.7: *A goal (also termed task, aim, objective or desire) is something that the agent is currently engaged in or working towards.*

Often goals are related to a state of the environment which the agent wants to achieve as the current state. However, this is not the only type of goal (there are other types such as maintenance, query, perform goals) [Thangarajah et al., 2001; Braubach et al., 2004]. Shen [Shen, 2005] lists several attributes of goals such as:

- goal level – difficulty associated in achieving a goal.
- goal distance – the closeness of the present state of the agent to the desired end state.
- goal specificity – the degree to which the goal is associated with the current context.
- goal complexity – the number of sub-goals that must be achieved before the overall goal is achieved
- goal conflict – the degree to which the achievement of one goal leads to the non-satisfaction of other goals (for example, this could be due to competition for resources among different actions).
- goal completeness – the degree to which a goal has been completely fulfilled.
- goal cost – the time, money, resource that must be expended to achieve the goal
- goal achievement/benefit – the recognized benefit of achieving the goal.

2.1.7.1 Types of goal models

Definition 3.8: *An agent goal model refers to the method used to abstract and represent the goals of an agent.*

Agent goal models are task-oriented or state-oriented [Rosenschein and Zlotkin, 94]. The task-oriented goal is a list of task; the goal is reached when all the tasks are completed. The state-oriented model views the goal as a sequence of states that the agent must go through. Macfadzean [MacFadzean and Barber, 1996] used a tree structure to decompose goals into higher abstract goals that breakdown into lower level sub-goals. Current research has focused on static goal relationships [Macfadzean and Barber, 1996; Park et al., 2000; Kolp et al., 2001].

2.1.7.2 Goal driven action selection

In pure reactive agent architectures, action selection is basically a response to percepts from the environment. In deliberative and hybrid architectures, goals have to be taken into account during action selection. Several types of action selection mechanisms have been proposed (see Table 2.2). Tyrell [Tyrell, 1993] examines different computational mechanisms for action selection in his thesis.

Definition 3.9: *Action selection refers to the agent's ability to select the most appropriate action or behaviour for the current context.*

Current agent systems generally make use of one action selection mechanism throughout its life-cycle. Goal net [Shen, 2005] allows for more flexibility in the agents; the agent can switch to different action selection mechanisms in the context of

the current goal being pursued. This approach models the way humans decide on their course of action.

In this research, we will focus on fast computation based mechanisms for action selection rather than learning/reasoning based mechanisms.

Table 2.2. Goal Driven Action Selection Mechanisms

COMPUTATION BASED		
Description	Examples	Reference
Action selection mechanism based on mathematics (such as fuzzy theory, heuristic search) or AI theory (such as neural network and fuzzy cognitive maps)	Neural-network based	[Ohtani et al., 2001]
	FCM based	[Miao et al., 1998]
LEARNING / REASONING BASED		
Description	Examples	Reference
Action selection mechanism based on reasoning mechanisms such as rule-based reasoning (RBR), case-based reasoning (CBR), and practical reasoning (BDI)	RBR, action selection using rules base and firing of condition-action rules	[Jennings et al., 1998]
	CBR, action selection uses knowledge of past experience to solve new problems.	[Lenz et al., 1998]
	BDI, action selection based on belief updated from percepts, possible actions and selected plans to be executed.	[Rao and Georgeff, 1995]

2.1.8 Agent architectures

There is no universally accepted definition for the term “software architecture”. Likewise, we do not have a universally accepted definition for “agent architecture.” We shall look at some working definitions of software architecture and agent architecture.

Definition 2.10: *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*

Definition 2.10 is taken from the book Software Architecture in Practice [Bass et al., 2003]. First, architecture defines elements. In nearly all modern systems, elements interact with each other by means of interfaces that partition details about an element into public and private parts. Architecture is concerned with the public side (“externally visible”) of this division; private details of elements are not architectural. The behaviour of each element is part of the architecture insofar as that behaviour can be observed or discerned from the point of view of another element. Software architecture is a “sketchy map” of the system. Software architecture describes the coarse grain element or components of the system. The connectors between these components describe the communication, which are explicit and pictured in a relatively detailed way. In the implementation phase, the coarse components are refined into “actual components”, e.g., classes and objects.

Definition 2.11: *Agent architecture is the structural model of the components that constitute an agent, the relationships among the components, and the computational model that implements the basic capabilities of the agent.*

Many agent architectures have been created from the rich history of agent research. Three main architectural approaches have been adopted for creating agents: the deliberative architecture, the reactive architecture and the hybrids [Wooldridge and

Jennings, 1995; Nwana and Ndumu, 1999]

The agent architectures in Table 2.3 are not an exhaustive list. They give an indication of different in designing agent architectures.

Table 2.3. Agent architectures

Type of architecture	Agent architecture	Area of Application	Reference
Deliberative	BDI	BDI agents have influenced other architectures such as IRMA and PRS.	[Rao and Geogreff, 1995]
Deliberative	IRMA	Experimental scenario Tileworld	[Bratman et al., 1988]
Deliberative	PRS	Simulation of maintenance procedure for the space shuttle	[Ingrand et al., 1995]
Deliberative	SOAR	SOAR is architecture for a system that is intended to be capable of general intelligence. TacAir-SOAR is an extension of SOAR that controls aircraft in military simulations	[Laird et al., 1987]
Deliberative	HOMER	Navigation (submarine) and simple tasks in virtual world (Seaworld)	[Vere and Bickmore, 1990]
Deliberative	GRATE	Electricity transportation management	[Jennings et al., 1998]
Deliberative	BB1	Blackboard system for general intelligent agent control.	[Hayes-Roth, 1984]
Reactive	Subsumption	control mechanisms for autonomous mobile robots	[Brooks, 1986]
Reactive	PENGI	simulated computer game	[Agre and Chapman, 1986]
Reactive	Agent Network Architecture	The network architecture is an approach to solving the problem of action selection in an agent.	[Maes, 1991]
Hybrid	Touring Machine	Architecture for controlling and coordinating the actions of autonomous agents in dynamic multiagent worlds.	[Ferguson, 1992]
Hybrid	INTERRAP	Model resource bounded BDI agents that interact with others in a dynamic environment.	[Müller et al, 1995]

Agent architectures are characterized using the following attributes (Table 2.4).

Table 2.4. Agent architecture characterization

Attributes	Description
Decision process	This attribute characterizes the decision making process of the agent. Agent may deliberate over alternative plans and goals. Some researchers focus on rational decisions by the agents. Other researchers have supported motivated, emotional agents as an alternative to rational decision processes. Clearly, emotions are invaluable for creating believable virtual agents and avatars in a virtual environment.
Autonomy	This attribute characterizes the degree of independence the agent exhibits from human intervention and how it interacts with its environment. A purely reactive agent simply responds to stimuli from the environment. A pro-active agent performs actions that satisfy some goal. Pro-active agents are more complex.
Knowledge representation	Agent knowledge may be stored in a knowledge base or encoded in the agent code. Knowledge structure may be symbolic. Alternatively, knowledge may be encoded in sub-symbolic forms such as neural network. Among the knowledge that must be represented are the plans, the goals, and the relationships between goals and percepts.
Intelligence	The intelligence of the agent is closely related to the decision process. For example, the agent may make use of a utility function to choose among several possible plans. The agent's reasoning ability should also improve if the agent has the ability to learn from its past experience.
Life-cycle management	This characterizes the agent life-time. The agent must be activated at some instant and occasionally persisted to storage for error recovery in case of system crashes. The life-cycle of the agent's data (which may be very large over its entire life-time) needs to be managed as well.
Control flow	This characterizes the control flow within the agent. Some agents have a sequential flow of control, executing one step at a time. Other agents require concurrent or parallel execution models.
Resource bounds	This characterizes the way agent architectures manage agent execution within a resource bound environment.

2.1.8.1 Deliberative

Deliberative architecture [Hayes-Roth, 1984; Laird et al., 1987; Rao and Geogreff, 1995] uses symbolic A.I. It is based on the physical symbol hypothesis. It

contains an explicitly represented symbolic model of the world, and decisions are made via logical reasoning, based on pattern matching and symbolic manipulation.

Two main issues deliberative architecture faces are:

- I. Transduction problem — how to translate the real world into an accurate representation
- II. Representation and reasoning problem — how to represent the real world and determine decisions about the world in a timely fashion.

2.1.8.2 Reactive

Reactive architecture [Brooks, 1986; Maes, 1991] breaks away from the classical A.I. model in deliberative architectures. Reactive architectures are based on the assumptions that: (1) intelligent behaviour can be generated without explicit representations, (2) intelligent behaviour can be generated without abstract reasoning, and (3) intelligence is an emergent property of certain complex systems.

2.1.8.3 Hybrid

Hybrid architectures [Ferguson, 1992; Müller et al, 1995] attempt to derive the benefits of both the deliberative architecture and reactive architecture. It is normally based on ad-hoc principles; no clear rule or principle for separation of concerns for the deliberative and reactive portions of the system. There is normally a hierarchy of control, with the deliberative architecture implementing the planning and higher orders of control. The reactive architecture portion takes care of the lower-level control. However, there is no clear rule for when the reactive portion ends and the deliberative

portion begins.

A hybrid architecture [Norman et al., 2003] that is of special interest was proposed by in the context of autonomic computing. In the architecture, the agent is divided into three levels: (1) Reaction, (2) Routine, and (3) Reflection (see Figure 2.5).

Figure 2.5. Three Level Model

The Reaction level provides simple responses to external stimuli and events. It provides event-condition-action type of processing behaviours in response to external events very quickly. Feedback from the Reaction level to higher levels such as the Routine and Reflection levels may trigger higher levels of processing to select an appropriate response. Inputs from the higher levels may be used to dampen or modify the actions that the Reaction level uses to respond to external stimuli.

The Routine level is where most common higher level processing occurs to select appropriate behaviour. The inference/deliberative, knowledge, planning and action components of the agent are located in the Routine level. The Routine level also receives inputs from the Reaction level and control signals which either inhibit or

activate actions from the Reflection level. The Routine level passes inputs to the Reflection level when it encounters discrepancies from norms or routine expectations.

The Reflection level models the “thinking about thinking” or meta-cognition that occurs in human beings. This level allows the agent to self-monitor or examine its own operations. The Reflection level does not have any direct sensory inputs, nor does it influence behaviours of the agent directly. However, inputs from lower levels can affect Reflection-level processing. The Reflection layer only has indirect control over behaviour through control signals that inhibit or activate behaviours to the lower levels.

2.1.9 Agent implementations

Based on the agent architectures described in the previous section, many agent toolkits and frameworks have been developed to simplify the development of agent-based software systems. Using existing frameworks or toolkits also help to increase the productivity of developers who can use the standard agent design patterns encoded within the frameworks and toolkits. Table 2.5 [JADE, 2007; RETSINA, 2007; BEEGENT, 2007; JACK, 2007] compares some of the agent toolkits that have been developed and used in various research projects. There are some commercial agent toolkits that have been deployed in products [Bussetta et al., 2000; Bigus et. al., 2002].

Table 2.5. Comparison of JADE, RETSINA, Bee-gent

	JADE	RETSINA	Bee-gent
Objective	FIPA-Standard	Middle agents allow heterogeneous agent types to interoperate successfully.	To provide a coordination mechanism for already-existing applications and databases
Version	JADE 3.5/LEAP 3.5	AFC SDK 1.10	Bee-gent 2.0

An Agent Based Approach Towards Autonomic Services

Chapter 2

	JADE	RETSINA	Bee-gent
Organization	TILAB (Telecom Italia Lab)	CMU	TOSHIBA Corp.
Language	Java	C++, C, Java	Java
Agent Intelligence			
Interaction Protocol	Useful behavioural patterns are provided. Also, FIPA-defined protocols are implemented in advance.	A specific Interaction Protocol handling module cannot be found.	IPs are described in UML state diagram and sequence diagram in the visual editor. Also, some patterns are provided.
Planning & Reasoning facility	Provided by JESS (Java shell of CLIPS)	Generic planning facilities are provided	Provided. It's similar to JACK
BDI model	Agents' mental states are implicitly expressed at the execution of Interaction Protocol	Belief DB is present for agents and developers to use	Provided. It's similar to JACK
Agent utilities / tools			
Naming / Directory Service	Federated FIPA-DF with GUI	ANS (currently not federated), UPnP discovery as alternative to ANS.	LDAP wrapper is provided.
Others	JSP support Servlet support	Matchmaker, User profile management, GPS tool, NLP, Grid display	
System features			
Message format	FIPA ACL S-expression, XML representation, and Bit-efficient encoding (for wireless). As a content language, SLO is supported.	KQML is the current ACL used by AFC agents. XML support is under development.	FIPA ACL XML representation. But any specific content language is not provided.
Message transport (MTP)	IIOP (CORBA), HTTP, TCP/IP socket for inter-platforms. RMI for intra-platform. SMTP and WAP are under development.	TCP/IP socket	HTTP
Mobility	Possible	No support	Possible
Device support	LEAP (a subset of JADE) is for J2ME-CLDC and pJava.	Smallest agent ever built: 32K for PalmOS	Now implementing for cellular phones which installed J2ME-CLDC
Standard specification	FIPA	None	FIPA (agent message only)

2.1.10 Autonomic Computing and Agents

The rapidly mushrooming complexity of integrating and managing computing systems such as the grid threatens to overwhelm the capabilities of software developers and administrators. Autonomic Computing (AC) [Horn, 2001; Kephart, 2003] is emerging as a significant new strategic and holistic approach to the design of computer systems. Its goal is the production of systems that are self-managing through key aspects such as self-configuring, self-optimizing, self-healing, and self-protecting (see Table 2.6).

Definition 2.1: *Autonomic computing (AC) is an approach to self-managing systems with minimal human interference. It is modelled after the human body's autonomic nervous system, which controls key functions without conscious thought or involvement.*

2.2 Most Related

The following sub-sections give the most closely related research to our work.

2.2.1 Autonomic Systems and Services

The complexity of large heterogeneous system such as the Grid has motivated the creation of self-managing Grids using of autonomic computing principles. Several research efforts [Agarwal et al., 2003; Pattnaik et al., 2003; Poellabauer et al., 2003] have focused on the creation of autonomic applications in a Grid environment. One of the pressing challenges is to define new software architectures and methodologies for

the self-managing systems.

The increase in the popularity of mobile computing has been fuelled by more powerful CPUs in mobile devices. Concurrently, the communication infrastructure has improved. Today it is possible to connect mobile computing devices to with high-speed wireless networks. This has led to the gradual inclusion of mobile devices as a significant part of the grid infrastructure, forming what is termed the pervasive grid or ubiquitous grid.

Nature has evolved to cope with scale, complexity, heterogeneity, dynamism, unpredictability and lack of guarantees through self-configuring, self-adapting, self-optimizing, self-healing, self-protecting, highly decentralized, heterogeneous architectures that work. One approach to Autonomic Computing Systems (ACS) is to mimic biological systems like the autonomic nervous system; which regulate body function such as heart rate and blood flow, and functions without the conscious decision making processes.

Figure 2.6. Autonomic manager

Table 2.6. Autonomic Computing System Concepts

Characteristic	Present Day	Autonomic Systems
Self-monitoring	Performance and fault management monitoring performed. Analysis and reaction to monitored data require skilled analysts to decipher the data	System is aware of significance of monitored data and intelligently reacts to the most critical data first.
Self-configuration	Human involved in planning the configuration and deployment. Migration of system from one version to another requires human tuning of parameters.	Automatic configuration of systems and applications with policies and high-level rules set by humans
Self-optimization	System parameters are manually tuned. Change of environment requires future analysis to determine new optimal values.	Individual components and the system as a whole attempt to maximize utility of users. Continuous improvement process is automatically executed by the system and its components.
Self-healing	Debugging or problem determination in a large software system is performed by programmers. This is a time consuming, and labour-intensive process.	Problem determination and isolation is automatic. System is able to repair localized software faults.
Self-protection	Detection of attacks and recovery is manual. Cascading failures need human intervention.	System automatically reacts to attacks. System prevents failures from cascading.

A basic concept in Autonomic Systems is control loops (see Figure 2.6). Essentially, a control loop in a self-managing system monitors some managed element (software or hardware component) and autonomously tries to keep its parameters within a desired range. Hundreds or even thousands of these control loops are expected to work in a large-scale self-managing computer system (see Figure 2.7).

Figure 2.7. Autonomic network

Table 2.7 indicates the breadth of current research efforts in the ACS area. The convergence of agent-based computing and grid computing has been receiving considerable attention in the last few years.

Table 2.7. Autonomic Computing Research Projects

Project	Description	Results/Findings
Berkeley University of California: Recovery-Oriented Computing	The Recovery-Oriented Computing (ROC) project is a joint Berkeley/Stanford research project that is investigating novel techniques for building highly-dependable Internet services. ROC emphasized recovery from failures rather than failure-avoidance. URL – http://roc.cs.berkeley.edu/	Prototype system that provide a system-wide undo facility for administrators and operators. Undo functionality for an e-mail store service.
Berkeley University of California: OceanStore	OceanStore is a global persistent data store designed to scale to billions of users. It provides a consistent, highly-available, and durable storage utility atop an infrastructure comprised of untrusted servers. Any computer can join the infrastructure -- users need only subscribe to a single OceanStore service provider, although they may consume storage and bandwidth from many different providers. URL – http://oceanstore.cs.berkeley.edu/	Many components of OceanStore are functioning in isolation. A complete prototype is under development.

Project	Description	Results/Findings
Rutgers, State University of New Jersey: Project Automate	The overall objective of Project AutoMate is to investigate key technologies to enable the development of autonomic Grid applications that are context aware and are capable of self-configuring, self-composing, self-optimizing and self-adapting. Specifically, it will investigate the definition of autonomic components, the development of autonomic applications as dynamic composition of autonomic components, and the design of key enhancements to existing Grid middleware and runtime services to support these applications. URL – http://automate.rutgers.edu/	Prototypes created to enable autonomic Grid services
University of Arizona: AUTONOMIA	Autonomia that provides dynamically programmable control and management services to support the development and deployment of smart (intelligent) applications. The AUTONOMIA environment provides the application developers with all the tools required to specify the appropriate control and management schemes to maintain any quality of service requirement or application attribute / functionality (e.g., performance, fault, security, etc.) and the core autonomic middleware services to maintain the autonomic requirements of a wide range of network applications and services. URL – http://www.ece.arizona.edu/~hpdc/projects/AUTONOMIA/	Prototype system developed for an autonomic computing environment.
University of Southampton: ANS	The ANS (Autonomic Networked System) is a ubiquitous computing management tool which is designed to mimic the ANS (Autonomic Nervous System) of living creatures. The organic ANS is the part of the nervous system controlling many organs and muscles within the body. It is flexible, constantly in operation and that it happens in the background without our interference or knowledge of its mechanism. This metaphor is being adapted to support ubiquitous computing environments, especially in the application of the intelligent home and medical applications where constant technical support is impossible. Such a system will provide the intelligence to optimize its operation through constant monitoring and tuning to achieve its goal. URL – http://www.ecs.soton.ac.uk/research/projects/NextWave-ANS	On going research into agent-based autonomic networks.
ETH Zurich: JOpera	JOpera is a rapid service composition tool offering a visual language and an autonomic execution platform for building distributed applications out of reusable services, which include but are not strictly limited to Web services. URL – http://www.iks.ethz.ch/jopera/	Released version 1.63 of the software.

Project	Description	Results/Findings
University of Florida: In-VIGO System	In-VIGO provides a distributed environment where multiple application instances can coexist in virtual or physical resources, such that clients are unaware of the complexities inherent to grid computing. URL – http://invigo.acis.ufl.edu/Login	Portal available for the application. In-VIGO version 1.0.4 released.

Two well known researchers made an analogy between agents and grid as “brain meets brawn” [Foster et al., 2004]. A similar convergence is observed between service-oriented computing (SOC) and grids; and between SOC and agents [Petrie and Bussler, 2003; Poellabauer et al., 2003; Singh and Huhns, 2005]. The need for autonomic computing is most felt in the area where this convergence is happening; that is, very large-scale, open, heterogeneous pervasive system.

2.2.2 Agents and Autonomic Services

Definition 2.5: *Autonomy refers to the characteristic a system that enables it to take flexible actions independent of human direction. The system is able to plan actions to take towards achieving its goals.*

The autonomous characteristic of agents is useful in the grid environment. In a grid, where there are potentially millions of processors, it is beyond the capability of the human programmer to efficiently partition the program into various components to run on different processors in the grid. An autonomous agent acts on its own to satisfy a set of goals without the programmer specifying the exact specific steps involved in achieving those goals. The programmer needs to specify the utility or value of various goals to the agents. This is in contrast to traditional programming languages such as C/C++ and Java, where the programmer has to provide a step-by-step set of

instructions in the program in order for the program to achieve its objectives.

2.2.3 Agent Oriented Software Engineering for SOC

AOSE is one of the most active areas of agent research in recent years. This is because it has been identified as one of the key ingredients needed to successfully deploy practical agents on a wider scale. There is a need for a service-oriented software engineering (SOSE) methodology that allows for the design and deployment of services. The methodology should address the entire SOA lifecycle from analysis and modelling of the environment and goals; and translating the model into services design; to deployment and management of the services system.

Service-Oriented Computing (SOC) increases the speed of system development through loose coupling between system components, discovery of services and consolidating multiple heterogeneous services. However, conventional SOC implementations are inflexible, vulnerable and inefficient in complex and dynamic processing. Conventional SOC systems are built using a client-server approach using relatively static pre-selected and contracted service providers. SOC in complex and dynamic environments need to rely on agent-mediated functions such as trust negotiation, autonomic behaviours, semantics-driven service matching, dynamic composition, and goal-directed behaviours. By capitalizing on their individual technical strengths, a combination of both services-based and agent-based software engineering may lead towards better approaches in creating large-scale distributed service-oriented software systems.

2.2.4 Software Engineering for Agents

Definition 2.6: *Agent oriented software engineering (AOSE) refers to software development practices, principles and theory that apply to the creation of agent based computing systems.*

To move agents from the laboratory to real-world applications, several AOSE methodologies have been proposed (see Table 2.8). Some of the methodologies in the table only support analysis and design. Some of the methodologies only cater to specific agent architectures such as BDI. Several AOSE comparisons have been made as well [Inglesias et al., 1998; Tveit, 2001; Jennings, 2000; Dam and Winikoff, 2003; Sturm and Shehory, 2003; Kavi, 2004; Luck et al., 2004].

Table 2.8. AOSE methodologies

Methodology	Agent concepts supported	Types of agents	Phases ⁺
Gaia ROADMAP [Zambonelli et al., 2003]	agents, roles, intentions, knowledge, services, acquaintances	General agents	A,D
MESSAGE [Caire et al., 2001]	agents, roles, organization, goals, interactions, knowledge	General agents	A
Tropos [Castro et al., 2002]	actors, plans, goals, interactions, resource, capabilities	BDI agents	A,D,I
Prometheus [Padgham and Winikoff, 2003]	goals, events, plans, percepts, belief, interactions, capabilities	BDI agents	A,D,I
Passi [Cossentino and Potts, 2002]	knowledge, societies, agents, roles	Robotics	A,D,I
CommonKADS [Schreiber et al., 1994]	organizations, agents, knowledge, interaction	Knowledge centred applications	A,D
MASSIVE [Lind, 2001]	agents, roles, organizations, views, interactions, capabilities	MAS	A,D,I

An Agent Based Approach Towards Autonomic Services Chapter 2

MaSE [Wood, 2001]	agents, roles, goals, interactions	Heterogeneous MAS	A,D,I
GO [Shen, 2005]	agents, goals, actions, goal net, knowledge	GO Agents, MAS	A,D,I

*Note: A=Analysis, D=Design, I=Implementation, V&V=Validation & Verification

AOSE methodologies need to take the following into consideration:

- Service-Orientation. AOSE needs to help model agents that are used to create service-oriented computing (SOC) system.
- Legacy Integration. AOSE needs to provide for the integration of legacy code into agent based systems.
- Large scale systems. AOSE needs to address issues of large scale systems such as security, autonomicity and management.
- System validation and verification. AOSE needs to provide for the validation and verification of the complex systems designed.

2.2.5 Agent-mediated Services

The SwinDeW [Shen et al., 2006] system combines a service-oriented architecture with a peer-to-peer infrastructure; and uses agents to provide the intelligent processing required for coordinating and executing the service workflows. It is able to overcome some of the limitations of traditional SOC approaches. For example, it shows greater flexibility in deployment in being able to support peer-to-peer interactions instead of relying on server-centric infrastructure; that are single-point of failures. SwinDeW uses agents to autonomously recover from failures of service providers by re-routing around failures; or re-negotiating and selecting new service providers. The decentralized process enactment using distributed peers also improves the efficiency of the process, as well as removing single points of failure. SwinDeW-S [Yan et al., 2006] improves

the process of service composition by using semantics based service matching.

2.3 Summary

Agents are an active area of research. Although they have been identified as a key technology to enabling autonomic computing, grid computing and service-oriented computing, the practical implementations of agent systems are not numerous. To the best of our knowledge, little research work has been reported on AOSE methodology for autonomic services.

CHAPTER 3

FCGN AGENT THEORY AND APPROACH

3.1 FCM, DCN and Behaviour Networks

3.1.1 Fuzzy cognitive map

FCMs are by no means a novel concept. R. Axelrod first used cognitive maps as a formal way of representing social scientific knowledge and modelling decision making in social and political systems [Axelrod, 1976]. Later, B. Kosko enhanced cognitive maps by considering fuzzy values for them [Kosko, 1986]. FCMs combine the robust properties of fuzzy logic and neural networks.

Definition 3.1: *A fuzzy cognitive map (FCM) is a directed graph (DAG) that describes the behaviour of a system in terms of concept nodes and edges which represent the relation between those nodes. The edges are directed. The “fuzzy” in FCMs indicate that concepts that can be represented as fuzzy sets and the causal relations between the concepts can be fuzzy implications.*

A FCM describes the behaviour of a system in terms of concepts; each concept represents a state or a characteristic of the system. A FCM can avoid many of the knowledge-extraction problems which are usually posed by rule based systems. Variable concepts are represented by nodes in a directed graph. The graph's edges are the casual influences between the concepts. The value of a node reflects the degree to which the concept is active in the system at a particular time.

Definition 3.2: A FCM which consists of n concepts is represented mathematically by a n -state vector $[c_1 c_2 c_3 \dots c_n]$.

Definition 3.3: The relations in the FCM between concepts can be represented by a $n \times n$ adjacency matrix M . Each element m_{ij} of the matrix indicates the value of the weight between concepts c_i and c_j . A directed edge m_{ij} from concept c_i to concept c_j measures how much c_i causes c_j .

Definition 3.4: The activation level of a concept node is limited by a threshold function f . The activation level is calculated by the following rule.

$$c_i^{new} = f\left(\sum_{j=1}^n c_j^{old} m_{ji}\right) \quad (1)$$

(f is a threshold curve function (for example, sigmoid, tanh, erf, or erfc functions).

Definition 3.5: A multiple iterations of the update computation, the system may converge to a set of values for the concept vector. This is called the fixed point. (For example, when $C^{new} = C^{old}$)

Definition 3.6: A multiple iterations of the update computation, the system may not converge to a set of values for the concept vector. However, the values of the concept vector keep repeating in a cyclic pattern. This is called a limit cycle.

Fuzzy cognitive maps have been used in virtual world by researchers such as Dickerson and Kosko [Dickerson and Kosko, 1994]. Fuzzy cognitive maps can

structure time-varying virtual worlds. A FCM models the relationships between causal events, actors, values, goals, and trends in a fuzzy feedback dynamical system. Other researchers such as Parenthoën have used FCMs to model the behaviour of autonomous virtual actors within the virtual environment [Parenthoën et al., 2001]. FCMs have also been used to model self-perception of virtual agents [Maffre et al., 2001]. In addition, FCMs were used to simulate virtual humans across diverse situations [MacNamee et al., 2003].

3.1.2 Goal Net

Definition 3.6: *Goal-Net is a modelling technique for agent goals and actions. The Goal-Net can be represented as a tuple $\langle S, T, A, FG, FA \rangle$. S is the set of goal states, T is the set of transitions, A is the set of arcs between states and transitions, FG is the set of goal selection mechanisms, FA is the set of action selection mechanisms.*

A Goal Net is composed of a few basic objects: states, transitions, arcs, and branches. With reference to Figure 3.1, the various Goal Net objects are:

- States: States may be atomic (not decomposed) or composite (decomposed to a sub goal net linked via branches). States represent the goals in the Goal Net model.
- Branches: A composite state is linked to a sub goal net via branches (left and right).
- Transitions: A transition represents actions that bring the Goal Net from one state to another state. They represent progression in achieving the overall goal of the agent. There are several types of transitions. Rectangles represent direct transitions where the logic is purely sequential. Diamonds represent conditional

transitions. Hexagons represent probabilistic transitions where probabilistic inference is used to select actions.

- **Arcs:** Transitions and states are joined via arcs. The relationship between two arcs in the model is represented by the end shape. If the end shape is an arrow, it represents a “or” relationship. If the end-shape is a diamond, it represents an “and” relationship. These relationships determine whether an alternate path is taken or execution proceeds concurrently. The arc type also determines the conditions necessary to proceed when arcs merge, whether any one or all conditions need to be satisfied.

Figure 3.1. Goal net model elements

There are different types of arcs as shown in Figure 3.1. Table 3.1 summarizes the meaning of arc types in different situations. There are 4 types of situations where arcs are used as connectors: 1) fan-out situation, where there is one transition that has arcs that reach multiple goals, 2) fan-out situation, where there is one goal that has arcs that reach multiple transitions, 3) join/merge situation, where there are multiple transitions that merge at a single goal; and 4) join/merge situation, where there are multiple goal that merge at a single transition.

We can use the notation $F(T, \{G\}, \text{AND})$ to represent a fan-out relationship between transition T , and set of Goals $\{G\}$, where all arcs are of AND-type.

$F(T, \{G\}, OR)$ to represent a fan-out relationship between transition T , and set of Goals $\{G\}$, where all arcs are of OR-type. Notice that there is more than one way to represent concurrent paths through the goal net: 1) $F(T, \{G\}, AND)$, 2) $F(T, \{G\}, OR)$ or $F(G, \{T\}, AND)$.

There is a similar set of relationship when arcs merge at a goal or transition. We can use the notation $J(\{T\}, G, AND)$ to represent a join relationship between a set of transitions $\{T\}$, and a goal G , where all arcs are of AND-type. $J(\{T\}, G, OR)$ to represent a join relationship between a set of transitions $\{T\}$, and a goal G , where all arcs are of OR-type.

Table 3.1. Arc Relationships

FAN-OUT (F)		JOIN (J)	
AND	OR	AND	OR
$F(T, \{G\}, AND)$ Multiple goals activated (NOT Used)	$F(T, \{G\}, OR)$ Multiple goals activated (concurrent)	$J(\{G\}, T, AND)$ Transition fired when all guard goals are true (NOT Used)	$J(\{G\}, T, OR)$ Transition fired when all guard goals are true (synchronization)
$F(G, \{T\}, AND)$ Multiple transitions fired	$F(G, \{T\}, OR)$ Only one transition is fired (choice)	$J(\{T\}, G, AND)$ Goal is true only when all in transitions are fired	$J(\{T\}, G, OR)$ Goal is true only when any in transition is fired

The transition action selection mechanism affects the agent behaviour, which comprises the agent actions. The agent behaviour may be simple or complex depending upon the topology of the goal net. The possible transition types and their usage are summarized in Table 3.2. Some of the possible combinations are not used (redundant).

Goal-Net model has been shown to be expressive model in a variety of applications [Shen et al., 2004a]. However, the current Goal-Net model does not elaborate on the use of fuzzy goals or fuzzy relationships between goals. This research builds on the Goal-Net model and extends it with the inclusion of fuzzy goals and fuzzy relations between goals

Table 3.2. Transition types

Transition type	Symbol	Action selection mechanism	Usage
Direct		Unconditional transition. Sequential action	Used for sequencing of actions. Execution is unconditional.
Rule-based		Conditional transition	Used where a crisp logic rule may applied to test conditions
Probabilistic		Probabilistic inference is used to select actions	Inference from available evidence is used to fire off action which are most likely or have highest executability

Fuzzy modelling for goals has the typical advantages of other fuzzy modelling techniques:

- Uses linguistic variables and rules which are more intuitive compared with mathematical formulae and equations.
- It is able to capture non-precise information and quantities.
- It approximates the manner human reasoning works and hence it is easier to capture the knowledge of human experts.

In this thesis, the Goal Net model notation is changed to use UML stereotypes in order to take advantage of available modelling tools. In addition, we introduce new elements into the goal net model such as the use of fuzzy cognitive transitions. The stereotypes and notations used are further explained in the section on methodology. The Table 3.3 shows the stereotypes that are used in our fuzzy cognitive model.

In the case of goal modelling in the Goal-Net model, fuzzy techniques also have the following additional advantages over normal crisp goal states:

- In many real-life problems, the goals of the system cannot be quantified precisely. This removes the need to distinguish between hard-goals and soft-goals such as what is found in the Tropos methodology. Goals are just treated with different degrees of precisely.
- Real world conditions which determine the context for goal-setting are also naturally fuzzy. For example, “take a bus if the rain is not too heavy.”

Other goal models have been proposed. For example, in Tropos, the goals are represented as a tree in the goal diagram. There are AND-decompositions and OR-decompositions of the goals described in the goal tree. The advantage of Goal-Net is that it is more flexible and rich in its representation, in the sense that its basic structure is a graph and not a tree. Hence, Goal-Net can better represent the context of the goals and its relationship to other goals. In addition, Goal-Net also allows for the decomposition of the graph into different hierarchy.

3.2 Fuzzy and goal-oriented modelling

There has been an increased recognition that goal modelling is an important and effective mean of eliciting the requirements of agent based systems [Anton, 1996; Anton, 1997; Kolp et al., 2001; Yu, 2001; Liu and Liu, 2002].

Shen [Shen, 2005] proposes the goal net (GN) model as a mechanism to capture both static and dynamic relationships between the goals. However, the linkages between the goals and external percepts have not been fully addressed.

Table 3.3. Fuzzy Cognitive Goal Net UML Stereotypes

Stereotype (short form)	UML metamodel	Description
Goals		
Composite goal (comp goal)	Package	Composite goal represents a goal which can be decomposed into a sub-goal net. The sub-goal net is represented as elements within the composite goal boundary.
Atomic Goal	Class	Atomic goal that is not decomposed into sub-goals.
Branch start (bs)	Class	Represents start of composite goal or sub-goal net.
Branch end (be)	Class	Represents end of composite goal or sub-goal net.
Transitions		
Sequential Transition (seq trans)	Class	Direct transition that allows task execution without complex processing. Frequently used to sequence operations.
Conditional Transition (cond trans)	Class	Rules are executed to determine conditional execution and transition exit point
Probabilistic Transition (prob trans)	Class	Transition execution is based on probabilistic action selection.
Fuzzy Cognitive (fuzzy trans)	Class	Transition execution is based on fuzzy cognitive map action selection.
Relationships		
OR relationship (OR)	Directed Association	Represent a choice between OR relationship arcs. This is similar to OR relationships in Goal Net [Shen, 2005]
AND relationship (AND)	Directed Association	Represent a simultaneous satisfaction of AND relationship arcs. This is similar to AND relationships in Goal Net [Shen, 2005]
Concurrent (conc)	Directed Association	Represents a fork in the goal net where two or more concurrent transitions or goals can occur.
Synchronize (sync)	Directed Association	The converse of concurrent, when multiple input transitions or goals must fire before the next goal or transition is taken.
Choice	Directed Association	Represents a fork in the goal net where a choice of taking 1 out of many transitions or goals must be taken.
Join	Directed Association	The converse of choice where only 1 out of many converging goals or transitions need to be fired before the next goal or transition is fired.

Stereotype (short form)	UML metamodel	Description
Events		
Event	Class	Represents input from the external environment.
Percept	Class	Represents inputs from sensors in the physical external environment.
Route	Dependency	Represents to signal route of the percept/event to the target goal. It also represents the positive or negative impact of the events/percepts on goals.

In this research, we will extend the goal net model to support dynamic inputs from external sensors. We will also explore the use of fuzzy goals and partial fuzzy goal fulfilment.

3.3 FCGN: Goal Net with FCM

We first examine the model for an individual agent in the MAS. Our work extends from work done by Shen [Shen et al., 2004a] using Goal Net to create a goal autonomous agent architecture. In addition to Goal Net, FCMs are the basic abstractions we use in the FCGN. A new type of transition, fuzzy cognitive transition, is introduced to the Goal Net model for dynamic action selection. The fuzzy cognitive transitions are marked as ovals. These transitions use FCM inference to drive their action selection. We have incorporated some concepts from Winikoff [Winikoff et al., 2001] to simplify the agent.

Definition 3.10: *A fuzzy cognitive goal autonomous agent can be formally specified as a tuple, $FCGN = \langle P, E, K, G, M, FG, FA, A \rangle$ where*

P percepts, the inputs that the agent senses about the environment

E events, set significant occurrences extracted from percepts, which are used to trigger reactive actions

- K is the knowledge of the agent represented by extended fuzzy cognitive maps
- G goal set, a set of agent goal states
- M is an agent goal model represented by the Goal Net model
- FG is the functions for goal selection. FG determine the goal autonomy of the agent
- FA is the functions for action selection. FA define the behaviour autonomy based on the selected goals
- A is a set of actions defining agent behaviours

FCGN agents are not only able to choose the right action towards its current goal, but also able to reason the next goal to achieve towards its overall goal. These characteristics are critical in a dynamic Grid environment where resources are frequently changing. The FCGN agent has the capacity to adapt by changing its goal in contrast to agents, which only exhibit behaviour autonomy. The MAS model extends from the model for the individual FCGN agent.

3.3.1 FCGN MAS

Definition 3.11: The FCGN MAS can be formally specified as a tuple, $FCGNMAS = \langle A, C, O, K, E \rangle$ where

- $A = \{ agent_i \mid i=1,2,\dots,n \}$ is a set of FCGN agents.
- C is a set of communication channels
- O is an ontology server
- K is the knowledge base of the agents
- E is the agent environment

The main components of the MAS are the FCGN agents, A . In each agent $_i$, the goal selection and action selection are done by the functions FG and the functions FA. The functions FG decide the next goal of the agent, based on FCM inference. The functions FA decide the next group of actions the agent needs to perform, based on FCM inference.

Figure 3.2. Agent knowledge model

Figure 3.2 shows an example of the agent's knowledge model using fuzzy cognitive maps (FCM). Each oval represents a fuzzy concept node within the FCM. The directed arcs represent relationships between the concept nodes. The directed arcs have weighted values to represent the strength of the relationship between the concepts. A large positive value represents a strong correlation, while large negative values represent strong negative correlation. The initial values of the concept nodes as well as the weights for all the edges are obtained from expert knowledge. FCM inference computation (see Definition 3.4) is then used to obtain the final values of the concepts.

The multi-agent system environment is where those agents live in. It defines the

agents running environment including system environment, communication infrastructure, and agent management environment. The knowledge base is the common knowledge shared by all agents in the MAS. Each individual agent also contains its own knowledge base for its own knowledge. The ontology server provides higher-level information. Ontologies maintained by the ontology server provide a common definition of the semantics of application areas to allow agents to communicate about those areas. The communication channels define the communication mechanisms between the agents.

3.3.2 FCGN MAS Behaviour

The FCGN Multi-agent System (MAS) consists of FCGN agents that are autonomous but interact with one another. Shen demonstrates the partitioning of a complex Goal Net (GN) into a group of agents in a MAS [Shen 2005]. The GN MAS is generally co-operative in nature since the original single, large, complex master Goal Net derived from requirements is decomposed into a number of independent, simpler Goal Nets to be executed in different agents within the MAS. The FCGN MAS proposed in this thesis adopts Shen's method of Goal Net decomposition and hence inherits the co-operative nature of the GN MAS. The extension we have proposed is the support of fuzzy cognitive transitions within GN.

It is possible to represent some of the agents within the FCGN MAS having competing goals. FCGN and GN MAS can contain choice transitions where one path through the Goal Net is chosen amongst several possible options. If each path contains Goal Subnets which are executing in different agents, then these agents could be competing. If we have several concurrent paths, potentially we could have several agents simultaneously competing to complete their goals ahead of the other agents.

Complex interacting systems such as FCGNMAAS may result in emergent behaviour being displayed by the MAS. An emergent behaviour or emergent property can appear when a number of simple entities (agents) operate in an environment, forming more complex behaviours as a collective; which cannot be predicted through analyzing behaviour of individual components in this complex system. Emergent behaviour may interfere with process properties such as ergodicity [Dessales 2007].

Stationarity and ergodicity cannot be ascertained in the FCGNMAAS from formal analysis. A stationary process is a stochastic process whose joint probability distribution does not change when shifted in time or space. An ergodic process is stochastic process where the time average is equal to the ensemble average. However, constructing a formal model and correctness proof of a complex distributed interacting computing system is infeasible. Wegner [Wegner 1997] proves this based on the fact that computing systems using interaction are more powerful problem solving engines than mere algorithms. Empirical and simulations may be used to verify the system behaviour. To date, research in complex system (such as self-organising emergent systems) has mainly analysed their solutions empirically by performing a large number of simulation experiments and measuring the macroscopic variables to obtain statistical results [De Wolf 2005, Kevrekidis 2004]. System behaviours are typically quantified with measurable variables which we define as macroscopic variables. A large number of simulation experiments, under varying external conditions, for varying degrees of completeness of agent knowledge, have to be conducted and the macroscopic variables measured to obtain statistical results.

3.4 Fuzzy goal selection mechanism

The goal net model [Shen, 2005] allows for different algorithms to be used for

goal selection. Shen proposed 3 anytime algorithms which considered the cost and achievement values of each goal. The first algorithm was based upon Dijkstra's algorithm. The second and third algorithms added constraint functions to the first algorithm and preferred routes between goals (trusted relationships) to the first algorithm. The algorithms were shown to satisfy the constraints and optimized the index values derived from the cost and achievement values of each goal, if sufficient time were given for the computation to complete.

The goal selection mechanism in FCGN is determined using the FCM inference mechanism. One key input is the environmental factors or sensor inputs. The environmental factors have a significant effect on the relevance of different goals within the goal net. A goal should be selected on the basis of two key ideas. First is the usefulness or utility of the goal. This can be derived using a utility function. The utility function summarizes various factor and may be fuzzy or non-fuzzy based. Second, the relevance of the goal to the current situation must be considered. Relevance is a more dynamic factor compared with utility. Relevance depends heavily on what the agent senses about its current context as well as its beliefs about the world.

Definition 3.12: *The fuzzy goal selection process can be modelled as a tuple with the elements $\langle G, T, P, E, R, FC \rangle$ where*

G is a set of fuzzy goals

T is a set of transitions

P is a set of inputs from sensors

E is a set of events

R is a set of context relations that map sensor input and event to goals

FC is a FCM that is constructed using $\langle G, P, E \rangle$ as FCM nodes, and $\langle T, R \rangle$ as edges of the FCM.

The concepts about the sensor inputs are represented as nodes in a FCM. The correct context for goal selection is modelled as a clamping input to the current goal in the goal net. A FCM is built using the sensory nodes, event nodes and all the goal states represented as nodes in the FCM. For illustrative the concept of FCM in combination with goal net, we shall use a highly simplified example. Assume that a person A wants to travel from Singapore to Bali in the safest, most comfortable and cost-effective manner. He has choice of airlines AL1 and AL2. Alternatively, he could travel by cruise line CL1. Goal g0 is the root goal – travel from Singapore to Bali. In Figure 3.3, each element represents:

- g0.s** Start of main goal sub-net (Branch start – short form bs)
- g0.e** End of main goal sub-net (Branch end – short form be)
- g0** overall goal – travel from Singapore to Bali
- g1** safety – travel safely
- g2** comfort – travel comfortably
- g3** budget – travel economically
- g4** favourite – travel on preferred airline
- t1** use AL1
- t2** use CL1
- t3** use AL2
- t4** arrival – with AL1
- t5** arrival – with CL1
- t6** arrival – with AL2
- t7** use AL3 (preferred airline)
- t8** arrival – with AL3 (preferred airline)

The following are what the percept and event nodes represent:

- P1** war declared
- P2** arrest of terrorist cell leader
- P3** fare hike
- P4** increase in fuel price
- P5** holiday season starts
- E1** threat of hijacking
- E2** flight overbooked

Dotted arrows are used to join percept and event nodes to goals and transitions that are causally related to those nodes. For example, the “threat of hijack” event E1 heightens

the importance of safety (goal g1) while reducing the relative importance of travelling on a budget airline.

A fuzzy cognitive map (FCM) is created using the elements shown in Figure 3.3, goals, percepts and events are matched to concept nodes in the FCM. Transitions which are responsible for action selection are not matched with FCM concept nodes. The transitions like t1 in Figure 3.3 are used as causal links to related goals. The new FCM is shown in Figure 3.4.

Figure 3.3. Goal Net, Percepts and Events

Using the FCM, the relevance of goals $g_1..g_3$ can be computed using the FCMs inference mechanism. The goal is selected based on the overall importance index. The importance index of the goal is computed from the input sensor values, and event values multiplied by the relationship (arcs) weight value for the goal. (see Figure 3.4)

Node g_4 is different from $g_1..g_3$ because it has no external inputs (see Figure 3.4). Hence, it has intrinsic static value as a goal which is not dependent on external factors. We don't include such nodes in the FCM computation.

Figure 3.4. FCM mapping percepts to goals

Another important factor is the external input values from the world. These values are fed as clamping values into the percept nodes. Events are activated by percepts since events are conceptually significant percepts or group of percepts that require a response. In Figure 3.4, these external input values have a different arrow head and they are placed at the bottom of the figure. The external inputs are incident on percept nodes ($P_1..P_5$).

3.5 FCM driven action selection mechanism

In goal net [Shen, 2005], Shen proposes the use of several action selection mechanisms (ASM) such as Bayesian Networks, Probability and Rule Based. In the FCGN, we have extended the goal net ASM to include a fuzzy cognitive map based ASM.

The transition action selection mechanism for fuzzy transitions (ovals in the goal net diagram) is determined through the use of fuzzy inference mechanisms. Specifically we use FCMs to increase the speed of inference by taking advantage of the non-symbolic, iterative, mathematic computations used in the FCM inference mechanism. In FCGN, the transitions may contain many task actions/behaviours that drive the agent from one goal state to another goal state. The choice of possible task actions/behaviours to be taken is determined by the action selection mechanism.

Definition 3.13: *The FCM action selection mechanism for a transition can be modelled as a tuple with the elements $\langle G, B, A, FC \rangle$ where*

G is a set of fuzzy goals (predecessor and successor goals of a transition)

B is a set of task actions or behaviours

A is a set of arc relationships

FC is a FCM that is constructed using $\langle G, B \rangle$ as FCM nodes, $\langle A \rangle$ as edges of the FCM.

The transition action selection mechanism should take into account the number of high importance index goals that are successor neighbours of the transition in the goal net. This back-propagation of the index value of the goal to the transitions from the neighbouring goals is important in ensuring that actions are selected that will most

likely enable the most important goals currently. Similarly, the inhibition of transitions that activate non-important goals should also be considered.

From the goal net in Figure 3.3, we can build an FCM. The initial index values of the goal nodes are computed using the FCM in section 3.2.3. Based on definition 3.13, we can build a FCM for the action selection mechanism of transition $t1$.

Note that the percepts and event nodes need not be considered in this FCM since their inputs are already accounted for by the index values of the goal nodes. To facilitate the back-propagation, the direction of the arcs in the goal-net are reversed for the successor goal nodes of transition $t1$ in the action selection FCM. This allows us to infer backwards from the most relevant goals to the currently most effective actions. A highly effective action is one that fires more than one relevant goal in the goal-net.

The algorithm considers all the transitions and states being at the same level. The start/end state of the parent composite state is represented for the current n^{th} level of the goal-net. Composite states take on their net values computed by applying the same algorithm recursively at the $n+1^{\text{th}}$ level.

Figure 3.5. FCM for action selection

The action selection mechanism would compute the index value of the task actions/behaviours using the FCM described above. The task action/behaviour nodes with the highest activation values are list in order of priority. The transitions that have the highest activation values are fired if their pre-conditions are satisfied.

3.6 Mathematical Representation

FCGN is an extension of Goal Net (GN) with fuzzy cognitive map (FCM) used for the goal selection and action selection functions. Goal-Net [Shen, 2005] is a modelling technique for agent goals and actions. GN is formally defined in this section with many sorted algebra to increase its expressiveness in handling different data types (See Appendix C for mathematical background on many sorted algebra).

3.6.1 Goal Net and Agent

A state [Shen, 2005] is a tuple (P, V, F, r) where P is a set of variables that define the profile of the state; V is a set of application variables; F is a set of internal functions that define behaviours on the state; and r is a time stamp. A transition [Shen, 2005] is a tuple (P, V, F, T, K, r) where P is a set of variables that define the profile of the transition; V is a set of application specific variables; F is a set of internal functions that define behaviours of the transition; T is a finite set of task lists; K is an action selection mechanism; and r is a time stamp. An arc [Shen, 2005] is a tuple (P, i, o) where P is a set of variables that define the profile of the arc object; i is a link to the input state or transition; o is a link to the output state or transition.

GAA: An goal autonomous agent [Shen, 2005] can be formally specified as a tuple, $GAA = (S, A, M, FG, FA, K, R, E)$, where

- S is a set of states defining agent goals,
- A is a set of actions defining agent behaviours,
- M is an agent goal model represented by the composite state goal model, i.e. Goal Net,

- K is the knowledge of the agent,
 R is a set of situation-action rules defining reactive behaviours of the agent,
 FG is the functions for goal selection defining goal autonomy of the agent,
 FA is the functions for action selection defining behaviour autonomy based on the selected goals,
 E is the agent environment that the agent lives in and perceives.

A process [Shen, 2005] is defined as a tuple $P = (S, T, A)$ where:

- S is a finite set of states;
 T is a finite set of transitions;
 $A \subseteq S \times T \cup T \times S$ is a finite set of arcs that joint states and transitions;
 $S \cap T = \emptyset$ and $S \cup T \neq \emptyset$;
 $\exists s_s \in S$ where s_s is an atomic state and is the only one that has no input but at least one output. The state s_s is called start state;
 $\exists s_e \in S$ where s_e is an atomic state and is the only one that has no output but at least one input. The state s_e is called end state.

A branch [Shen, 2005] is a tuple (P, V, F, s, l) where P is a set of variables that define the profile of the branch; V is a set of variables; F is a set of internal functions that define behaviors of the branch; s is a link to a composite state; and l is a link to the start state or end state of a process.

A composite state [Shen, 2005] is defined as a tuple $Sc = (P, s_i, s_t, B)$ where:

- P is a process;
 s_i is the initial state of the composite state;
 s_t is the target state of the composite state;
 $B = \{b_l, b_r\}$ is a two element set containing two branches. The left branch b_l joins s_i and the start state of P whereas the right branch b_r joins s_t and the end state of P.

A goal [Shen, 2005] is a desired state an agent intends to achieve. The goal of an agent is the root composite state of a Goal Net.

A Goal Net [Shen, 2005] is a hierarchical net. It is defined as a tuple: $GN = (C, R, H, R_0)$ where:

- C is a set of composite states, that is, goals;
 $R \in C$ is the only root of the net;
 H is level number of the structure;
 R_0 is the initial settings of the net;

Goal Net Scheme (GNS) is a structure given by

$$\mathbf{GNS} = (GN, NG, Sig, V, Sort, An, Mo)$$

Where

- GN is the Goal Net tuple
- $NG = \{P, T, F\}$ is a net graph, with
 - P a finite set of nodes, called Goal States
 - T a finite set of nodes, called Transitions (disjoint from P)
 - $F \subseteq (P \times T) \cup (T \times P)$ a set of directed edges called arcs, known as the flow relation

- $Sig = (S, O)$ is a Boolean signature. S is set of Sorts, O is a set of operators.
- V is S -indexed set of variables, disjoint from O
- $Sort : P \rightarrow S$ is a function which assigns sorts to goals states
- $An = (a, TC)$ is a pair of net annotations
 - a is a function that annotates each arc with a term of the same sort as that of the associated goal state.
 $a : F \rightarrow TERM(O \cup V)$ such that $(p, t), (t', p) \in F, a(p, t), a(t', p) \in TERM(O \cup V)_{Sort(p)}$.
 - TC is a function that annotates transitions with Boolean terms.
 $TC : T \rightarrow TERM(O \cup V)_{Bool}$
- Mo is the initial marking function which associates a ground term with each goal state. The ground term has the same sort as the goal state.
 $Mo : P \rightarrow TERM(O)$ such that $\forall p \in P, m_o(p) \in TERM(O)_{Sort(p)}$

μP is the set of multisets over the set of goal states, P (See Appendix for definition of multisets). Marking of a state is collection of elements (data items) chosen from the goal state's type (sort of data) and associated with the goal state. Repetition of items is allowed. The items associated with goal states are called tokens. A net marking comprises the set of all goal state markings of the net.

Marking M of the GNS is a multiset, $M \in \mu P$

Key concepts governing the execution of the GN is the flow of tokens within the GN. Controlling the flow of tokens are enabling of transitions and the occurrence of transitions defined by the Transition Rule. A transition is enabled with respect to the current net marking M . Enabled transitions can occur or fire.

When a transition occurs, tokens are removed from its input states, and tokens are added to its output states. An input state of a transition is a state which is connected to the transition by an arc leading from that state to the transition. An output state of a transition is a state which is connected to the transition by an arc directed from the transition to the state. For GN, the Transition Rule varies according to the type of Transition (direct, conditional, or probabilistic).

For direct transition, the input state must contain at least a token for the transition to fire. For conditional transition, a rule is specified to relate the tokens in

the input states to the firing of the transition. For probabilistic transitions, a Bayesian network is used to compute the firing of the transition. The occurrence of the transition in GN is linked to the action selection mechanism used.

3.6.2 Fuzzy Cognitive Goal Net and Agent

FCGN agent: A fuzzy cognitive goal autonomous agent can be formally specified as a tuple, $FCGN = \langle P, E, K, G, M, FG, FA, A \rangle$ where

- P percepts, the inputs that the agent senses about the environment
- E events, set significant occurrences extracted from percepts, which are used to trigger reactive actions
- K is the knowledge of the agent represented by extended fuzzy cognitive maps
- G goal set, a set of agent goal states
- M is an agent goal model represented by the **FCGNG**
- FG is the function for goal selection. FG determine the goal autonomy of the agent
- FA is the function for action selection. FA define the behaviour autonomy based on the selected goals
- A is a set of actions defining agent behaviours

FCGNG: A fuzzy cognitive goal net graph (FCGNG) can be formally specified as a tuple, $FCGNG = \text{tuple} \langle S, T, F, FG, FA, P, E, K \rangle$ where

- S is the set of goal states
- T is the set of transitions (disjoint from S, $S \cap T = \emptyset$)
- F is the set of directed edges called arcs between states and transitions
 $F \subseteq (S \times T) \cup (T \times S)$
- FG is the functions for goal selection. FG determine the goal autonomy of the agent
- FA is the functions for action selection. FA define the behaviour autonomy based on the selected goals
- P percepts, the inputs that the agent senses about the environment
- E events, set significant occurrences extracted from percepts, which are used to trigger reactive actions
- K is the knowledge of the agent represented by extended fuzzy cognitive maps (FCM)

FG: The fuzzy goal selection process can be modelled as a tuple with the elements $\langle G, T, P, E, R, FC, NG, M \rangle$ where

- G is a set of fuzzy goals
- T is a set of transitions
- P is a set of inputs from sensors
- E is a set of events
- R is a set of context relations that map sensor input and event to goals
- FC is a FCM that is constructed using $\langle G, P, E \rangle$ as FCM nodes, and $\langle T, R \rangle$ as edges of the FCM.

- NG is the Net Graph. $NG = (G, T, F)$ where G is the set of goal states, T is the set of transitions, and F is the set of directed edges called arcs between goal states and transitions. $F \subseteq (G \times T) \cup (T \times G)$
- M is the Marking of the Net. The Marking of the Net is collection of tokens contained in each goal state in the Net graph

FA: The FCM action selection mechanism for a transition can be modelled as a tuple with the elements $\langle G, B, A, FC, NG, M \rangle$ where

- G is a set of fuzzy goals (predecessor and successor goals of a transition)
- B is a set of task actions or behaviours
- A is a set of arc relationships
- FC is a FCM that is constructed using $\langle G, B \rangle$ as FCM nodes, $\langle A \rangle$ as edges of the FCM.
- NG is the Net Graph. $NG = (G, T, F)$ where G is the set of goal states, T is the set of transitions, and F is the set of directed edges called arcs between goal states and transitions. $F \subseteq (G \times T) \cup (T \times G)$
- M is the Marking of the Net. The Marking of the Net is collection of tokens contained in each goal state in the Net graph

FG and FA are not modelled using algebraic structures but using an algorithmic approach. The following indicates the algorithms for the FG and FA functions.

Algorithm: *FCGN Agent Control Loop*

1. $G := G_0$ /* G_0 is the initial goals */
2. $FC := F_0$ /* F_0 is the initial FCM */
3. $M := M_0$ /* M_0 is the initial tokens */
4. while not root_succeeded(G) do
5. update_fcm(FC);
6. update_environment_sense(P, E, R); /*
7. $G := \mathbf{FG}(G, T, P, E, R, FC, NG, M)$; /* goal selection*/
8. $T := \mathbf{FA}(G, B, A, FC, NG, M)$ /*action selection */
9. $M := \text{update_marking}(T)$; /* add/remove tokens fired */
10. $L := \text{actions}(T)$ /* actions list from transitions */
11. schedule(L) /* schedule actions */
12. end while

Algorithm: *Goal Selection*

1. function $\mathbf{FG}(G, T, P, E, R, FC, NG, M) : G : \text{goals}$
2. begin
3. $G = \emptyset$ /* empty */
4. $P = \text{next_goals_enabled}(NG, M)$;
5. foreach p in P do
6. update_fcm(p); /* update FCM associated with p */
7. if (high_value_goal(p))
8. $G = G + p$; /* update of goals selected */
9. end if
10. end foreach
11. return G
12. end function $\mathbf{FG}$

Algorithm: *Action Selection*

1. function $\mathbf{FA}(G, B, A, FC, NG, M) : T : \text{transitions}$
2. begin
3. $T = \emptyset$ /* empty */

```

4.   P = next_transitions_enabled(NG,M);
5.   foreach  $\tau$  in P do
6.       update_fcm( $\tau$ ); /* update FCM associated with  $\tau$  */
7.        $T := T + \tau$ ; /* add  $\tau$  to list of new transitions */
8.        $m := \text{fire\_transition}(\tau)$ ;
9.   end foreach
10.  return T
11. end function FA

```

A FCM which consists of n concepts is represented mathematically by a n state vector $[c_1 \ c_2 \ c_3 \ \dots \ c_n]$. The relations in the FCM between concepts can be represented by a $n \times n$ adjacency matrix M . Each element m_{ij} of the matrix indicates the value of the weight between concepts c_i and c_j . The activation level is calculated by the following rule:

$$c_i^{new} = f\left(\sum_{j=1}^n c_j^{old} m_{ji}\right)$$

Essentially, FG and FA change the Marking of the Net as the FCGN executes. They can be defined as transition functions.

FG: $M_1 \rightarrow M_2$

FA: $M_2 \rightarrow M_3$

Where M_1, M_2, M_3 are different markings of the Net.

FG and FA are not modelled using algebraic structures but using an algorithmic approach. From the pseudo-code, we can see that FG and FA depend on many other parameters such as the Net Graph structure and the FCM network constructed for the goal states and transitions.

FG and FA can model different conditions such as:

- sequence
- loops
- choice
- non-determinism (with choice flows)
- concurrency
- synchronization

The mathematical expressiveness of FG and FA are limited by:

- lack of function to detect non-existence of token
- inability to model time dimension

3.7 Expressiveness and Limitations of FCGN Model

The Goal-Net (GN) model can be formally defined with many-sorted algebra to increase its expressiveness in handling different data types. FCGN is an extension of GN with fuzzy cognitive map (FCM) used for the goal selection and action selection functions. We introduce a new type of Transitions in FCGN called Fuzzy Cognitive Transitions.

The concept of many sorted algebra, originating from the theory of abstract algebra in modern mathematics serves as a data model language [Goguen et al. 1976, Futasugi et al. 1985, Jaffar et al. 1986]. The many sorted algebra has the features of being object-oriented, declarative, operational, well defined, and implementation-independent [Breu 1991]. As compared with traditional algebra, which allows one data type only, many sorted algebra can handle multiple data types and possess many preferable properties. Cheng et al. [Cheng et al 2003] proposed a many-sorted general framework to incorporate algebraic computation with logical reasoning, which equally encompasses following systems as special cases: lattice-valued fuzzy logic, operator fuzzy logic, operator fuzzy logic for belief, operator fuzzy logic for argumentation, fuzzy logic, probabilistic logic, annotated logic, language of signed formulas, autoepistemic logic.

Tokens in GN can be represented using many sorted algebra. GN Transitions which have a rich variety of methods by which they are enabled and fired. However, the enabling of GN Transitions cannot be expressed with only Transition Conditions

consisting of algebraic terms. The GN Transitions can be Direct, Conditional, or Probabilistic. FCGN is an extension of GN with fuzzy cognitive map (FCM) to create the new Fuzzy Cognitive Transitions.

The expressiveness of GN and FCGN is similar to High-level Petri Net (HLPN) with many-sorted algebra. GN and FCGN can support multiple data types. They can support the control flows such as sequence, loops, and choice. They can model non-determinism (with choice flows), concurrency and synchronization. In addition to the previous features, both GN and FCGN can support hierarchy to decompose complex net graphs into a hierarchy of simpler net graphs. The extended feature in FCGN is fuzzy cognitive transitions that use fuzzy sets which are not limited to only discrete sets or real values. In this sense, FCGN is more expressive than the GN which uses Direct, Conditional and Probabilistic transitions.

Both GN and FCGN did not specify inhibitor arcs which are able to test for empty markings in states. For HLPN, the use of inhibitor arcs makes HLPN to have equivalent expressive power as Turing machines.

3.8 Summary

Goal-oriented agents can be modelled using the Fuzzy Cognitive Goal Net (FCGN) model. A FCGN includes features from Goal Net (GN) and Fuzzy Cognitive Maps (FCM).

CHAPTER 4

FCGN AGENT DESIGN AND IMPLEMENTATION

4.1 Overview

This chapter describes the design and implementation of the FCGN agents. Chapter 4 is intended to show one single concrete example that the FCGN agent specifications and models can be practically implemented and executed using a common agent platform such as JADE.

Chapter 4 provides one instance of the many possible feasible implementation strategies. To evaluate the different possibilities, an example of the comparison between different agent platforms is provided in Section 2.1.9–Agent implementations. The comparison is not specific to FCGN implementation in section 2.1.9. In general, an evaluation of possible agent platforms for FCGN should consider the attributes (for example, Language, Interaction Protocol, Planning & Reasoning, Standards) in Table 2.5. A comparison table containing the agent platform attributes and score values is created. A score value (such as 1 to 10, with 1 being the lowest score) and importance weight should be associated with each attribute for each agent platform. The total score is the sum of the weighted score values. The different agent platforms to be considered should be ranked according to the total score for each agent platform. (Please refer to Chapter 6, in which we conclude some methods to evaluate the strategies.)

The FGCN and FCM models have graphical forms and notations for presentation which act as visual communication aids. The SGNML mark up language and FCMML are used to store the FCGN net graph model and FCM model in a textual form. The textual form is formatted as XML documents to facilitate parsing of the information

stored.

The transfer formats SGNML and FCMML encode the FCGN models designed during the architectural phase in a form that facilitates storage and exchange. The SGNML and FCMML are intended to be transfer formats for the storage and exchange of the models between different tools, or between a modeller and the execution platform. The transfer format has been designed to be extensible and open for future variants of FCGN and FCM and possibly for other use; such as, the transfer of results associated with the analysis of FCGN models.

The transfer format SGNML and FCMML encode the FCGN models designed during the architectural phase in a form that facilitates storage and exchange.

4.2 FCGN Agent Implementation

We first examine the architecture of the major component of the FCGNMAAS, the individual FCGN agents. The FCGN agent architecture consists of eight units, which include sense unit, goals unit, control unit, scheduler unit, communication unit, knowledge unit and decision unit as shown in Figure 4.1.

- The **sense unit** senses the agent environment. It contains a list of states to indicate the status of the environment. If the environment changes, the sense unit will update the database. Moreover, if the sense unit of an agent detects significant occurrences in the environment, the events generated can enable the reactive reflex action in response to the changes.
- The **goals unit** contains the goal(s) of an agent and their relationships (Goal nets). The Goal nets will be loaded into the goals unit from the knowledge base after the agent starts to work.
- The **scheduler unit** schedules all reactive reflex actions and selected actions

for execution.

- The **decision unit** defines the goal selection function and the action selection functions. The goal selection algorithms, action selection algorithms and action selection mechanisms are implemented in the decision unit. The control unit will call the functions in this unit to select the next goal and actions according to the goal model given in the goals unit.
- The **knowledge unit** maintains the knowledge of the agent, which is used to handle the real world problems.
- The **control unit** controls the agent execution cycle. The initiates the sense-think-act cycle. It is also responsible for coordinating the different units.
- The **communication unit** defines the communication mechanism between agents.

A FCGN agent is always trying to pursue its current goal while inferring its next goal and actions based on the current situation.

Figure 4.1. The FCGN agent architecture

The following sections will focus on the implementation of the FCGN agent implementation using a java based framework.

4.3 FCGN Agent Execution Framework

The FCGN agent framework was written in java. Java is able to support a variety of platforms: Windows, Linux, MacOS. Java supports different editions for different application environments: 1) J2ME, for small and mobile devices, 2) J2SE, for personal computers, and 3) J2EE, for server-side programming. To ensure portability across the different Java editions, we have to limit to Java APIs used to the narrower set used by J2ME for the core FCGN engine (in fcgn.core package). We use separate packages for portions of the execution framework that are intended to run

4.4 Simplified Goal net Markup Language (SGNML)

The simplified goal net markup language (SGNML) allows us to model the goal net using XML. We use an attribute based encoding for the data because it facilitate simpler parsing using the SAX parser (only XML parser defined currently for J2ME). This also reduces the computational load while loading the goal-net. We can serialize the goal net model in memory into an SGNML file.

The sgtml element contains the name of the goal net (**name** attribute). All element contain an **id** attribute which is unique in the goal net. By convention ids follow the following convention: 1) for states ids are of the form sN, where N is an integer, 2) ids for transitions are of the form tX, where X is an integer, 3) ids for arcs are of the form aZ, where Z is an integer, 4) ids for tasks are of the form taskY, where Y is an integer. The **id** field is mandatory. Each element also has an optional name (**name** attribute) and optional description (**desc** attribute).

The **gnet** element has an attribute that indicates the root state (**rootId** attribute). The **goalContinuous** attribute is by default false. If it is set to true, it indicates that the goal should be automatically restarted after it is achieved. This is useful for maintenance type goals.

```
<?xml version="1.0"?>
<sgnml name="gnml0">
  <gnet id="gn1" rootId="s1" goalContinuous="false">
    <state id="s1" name="root" desc="goal" startId="s2" endId="s5" root="true" composite="true" />
    <state id="s2" name="START" desc="start" parentid="s1" />
    <state id="s3" name="MID1" desc="mid1" parentid="s1" />
    <state id="s4" name="MID2" desc="mid2" parentid="s1" />
    <state id="s5" name="END" desc="end" parentid="s1" />
    <transition id="t1" name="transition1" desc="dummy" actionSelection="sequential">
      <task id="task10" />
    </transition>
    <transition id="t2" name="transition2" desc="dummy2" actionSelection="sequential">
      <task id="task20" />
    </transition>
    <transition id="t3" name="transition3" desc="dummy3" actionSelection="sequential">
      <task id="task30" />
    </transition>
    <arc id="a1" name="arc_id1" desc="arc_s2t1" arcType="or" state_in="s2" transition_out="t1" />
    <arc id="a2" name="arc_id2" desc="arc_t1s3" arcType="or" transition_in="t1" state_out="s3" />
    <arc id="a3" name="arc_id3" desc="arc_s3t2" arcType="or" state_in="s3" transition_out="t2" />
    <arc id="a4" name="arc_id4" desc="arc_t2s4" arcType="or" transition_in="t2" state_out="s4" />
    <arc id="a5" name="arc_id5" desc="arc_s4t3" arcType="or" state_in="s4" transition_out="t3" />
    <arc id="a6" name="arc_id6" desc="arc_t3s5" arcType="or" transition_in="t3" state_out="s5" />
  </gnet>
</sgnml>
```

Figure 4.2. SGNML

The state element contains **root**="false" and **composite**="false" by default. The **startId** and **endId** are mandatory for composite states. For a composite state, the **root**="true" and **composite**="true" must also be set. These indicate the start of the sub-goal net and the end of the sub-goal net. For non-composite states, the **parentid** must be set to the correct parent goal id.

The transition element contains its associated tasks as a nested element. The **actionSelection** attribute determines the transition action selection mechanism used.

The example SGNML shows only **actionSelection**="sequential". Other possible values for this attribute include “probabilistic”, “fuzzy” and “rulebased”.

The arc element is used to connect transitions and goals. The arc relationships are indicated by the attribute **arcType**. If the arc corresponds to an edge from a state to a transition, then the attributes **state_in** and **transition_out** must be set. The attributes store the ids of the state and transition respectively. If the arc corresponds to an edge from a transition to a state, then the attributes **transition_in** and **state_out** must be set.

4.5 FCM Markup Language (FCMML)

```
<?xml version="1.0"?>
<fcmml>
  <fcm id="fcm1">
    <node id="node0" value="1.0" clamp="1.0" />
    <node id="node1" value="2.0" />
    <node id="node2" value="2.0" />
    <node id="node3" />
    <node id="node4" />
    <edge id="edge13" from="node1" to="node3" />
    <edge id="edge10" from="node1" to="node0" weight="8.8" />
    <edge id="edge02" from="node0" to="node2" />
    <edge id="edge32" from="node3" to="node2" />
    <edge id="edge24" from="node2" to="node4" />
    <edge id="edge41" from="node4" to="node1" />
  </fcm>
</fcmml>
```

Figure 4.3. FCMML

The concept nodes of the fuzzy cognitive map are represented by the node element. The node element has a mandatory **id** attribute. By convention, the value of the node is of the form nodeZ, where Z is an integer. The **value** attribute indicates the default activation value of the concept node. By default, the value attribute has a zero value. External clamping values are indicated using the **clamp** attribute.

The edges connecting the concept nodes are represented by the edge element.

Since FCMs are directed graphs, each edge element has a **from** and **to** attribute that represent the start and end nodes that are incident to that edge. Each edge also has a mandatory **id** attribute. By convention, the value of the edge is of the form edgeY, where Y is an integer. The weight of the edge is indicated by the **weight** attribute. By default the weight of the edge is zero.

One problem faced with J2ME and possibly other devices is that J2ME does not provide floating point number support for many older devices. This means that we have to use a fixed point representation to approximate floating point numbers. Depending on the number of bits used in the fixed point representation, there will be round-off errors. A larger number of bits used, reduces the significance of the round-off errors since we are able to represent smaller quantities.

4.6 Modules

Modules are the services portion of the FCGN agents. Service invocations are supported through modules which encapsulate the logic and program code required to invoke a service. Modules also allow us to publish services through web service interfaces.

4.7 Agent Implementation Platforms

The FCGN agents described in the previous sub-sections may be implemented on a variety of possible agent implementation platforms. The different agent implementation platforms have their own specific constructs. The broad approaches for agent implementation are as follows:

1. Using custom code for the agents.

- All code for agent execution and control are implemented without re-using previous code.
 - There is maximum flexibility in the design of the agent and the execution of the agent model.
 - Performance is good if code is optimized for the execution of the agents.
 - Minimal redundant code for unused services.
 - Portability across different hardware platforms requires extra coding effort for different hardware.
2. Using libraries or agent development frameworks.
- Some amount of code re-use is possible.
 - Development coding effort is reduced through the re-use of libraries and framework code.
 - Performance is good if libraries are optimized.
 - Portability across different hardware platforms requires libraries to be portable or multiple versions of libraries to be available.
 - Poorly implemented common agent services such as naming and discovery by the user may result in poor performance.
3. Using agent middleware or agent development kits.
- Many similarities to the library or agent framework approach. But, in addition, common agent services such as naming are provided as part of the middleware.
 - Re-use of common agent support sub-systems such as naming, discovery, and messaging.
 - Performance for the system is optimized since the common agent components are re-used.
 - Portability across different platforms is enabled if the ADK is designed

for different hardware platforms (for example, JADE)

4. Using an agent programming language.

- Depends on compiler or interpreter mechanism to convert specialized agent programming language (such as INDUS) to executable byte codes.
- Implementation may depend on virtual machine (VM) for execution of compiled code. Performance may be poorer because of the overheads associated with the VM.
- Portability across different hardware platforms requires libraries to be portable or multiple versions of libraries to be available.
- Code re-use is possible.
- Maintenance of the software requires developers to know specialized agent programming language syntax. Hence, there is some limitation to its maintainability.
- Flexibility to the system may be limited by the basic services supported by the compiler. For example, it may not be possible to add new services or new mechanisms to the agent platform without modification to the compiler.

5. Declarative or non-programming based agents.

- The developer uses a visual or graphical programming tool to declare the actions to be performed by the agent.
- Alternatively, the behaviour of the agent is declared using some form of mark-up language.
- This approach is similar to the specialized agent programming language approach, except that it does not use a clearly visible specialized programming agent language.
- Virtual machines (VM) may be used for execution of compiled code and

this may affect performance of the agent system.

- Portability is supported if libraries and a compiler are available for different platforms
- Flexibility is limited by the support given by the compiler.
- Maintainability for the visual programming platform is better than using specialized agent programming languages since graphical notations are more intuitive.

To maximize portability, performance, and code re-use while providing the flexibility to make future enhancements, we have decided to use a java-based agent development kit (ADK) or middleware approach for implementing the FCGN agents. There are several possible middleware or ADKs such JACK, FIPA-OS, COUGAAR, A-globe, and JADE.

JACK [JACK, 2007] was developed to provide agent-oriented specific extension to Java. JACK source code is first compiled into Java by the JACK agent compiler. The code for the agent then runs on a normal Java Virtual Machine (JVM). Modern JVM provide Just-In-Time (JIT) optimization and can achieve good performance. JACK is based on the BDI agent model. JACK agents can be considered autonomous software components that have explicit goals to achieve or events to handle. To describe how they should go about achieving their goals, these agents are programmed with a set of plans. Each plan describes how to achieve a goal under varying circumstances. Set to work, the agent pursues its goals, adopting appropriate plans according its current data or beliefs about the state of the world. To support BDI agents, JACK has 5 main platform specific constructs: (I) Agents – The agents define the intelligent behaviour. They have methods, data members, capabilities, database relations, description of events and plans; (II) Capabilities – The capabilities

encapsulate and collect events, plans, database relations and other capabilities into reusable “parts” or agent modules. Capabilities provide a mechanism to re-use functionality in different agents; (III) Database Relations – The database relations store agent beliefs and data; (IV) Events – The events are important circumstances or messages to which the agent must respond; (V) Plans – The plans are instructions the agent follows to try to achieve its goals and executes to handle events. JACK is capable of running on any system on which the Java Virtual Machine is available, from PDAs, through PCs to high-end multi-CPU servers. Currently, JACK does not support Java ME (MIDP) for mobile phones. JACK does not support agent mobility.

FIPA-OS [FIPA-OS, 2003] is a component-based toolkit enabling rapid development of FIPA compliant agents. It has some support for agent mobility. There is a microFIPA-OS version which supports PDAs, but it does not support Java ME (MIDP) for mobile phones. The platform supports communication between multiple agents using an agent communication language which conforms to the FIPA (Foundation for Intelligent Physical Agents) agent standards. A key focus of the platform is that it supports openness. The design of the platform itself supports openness through parts with loose coupling and support for extensions. The openness is further emphasized in that the platform software is distributed and managed under an open-source licensing scheme. It has been demonstrated to interoperate with other heterogeneous FIPA compliant platforms and is in use in numerous institutions around the world.

COUGAAR is a Java-based architecture for the construction of large-scale distributed agent-based applications. It is a product of two consecutive, multi-year DARPA research programs into large-scale agent systems spanning eight years of effort. The first program conclusively demonstrated the feasibility of using advanced

agent-based technology to conduct rapid, large scale, distributed logistics planning and re-planning. The second program is developing information technologies to enhance the survivability of these distributed agent-based systems operating in extremely chaotic environments. The resultant architecture, COUGAAR, provides developers with a framework to implement large-scale distributed agent applications with minimal consideration for the underlying architecture and infrastructure.

A-globe [A-GLOBE, 2007] is an agent platform for fast prototyping and application development of multiagent systems. A-globe provides many of the common agent services such as communication infrastructure and directory services. The main focus of A-globe is on large scale agent-based simulations. A-globe has a lightweight communication infrastructure which allows it to scale to a large numbers of autonomous agents. A-globe has support for agent migration. It requires the Java virtual machine for execution. There is no support for Java ME to execute the A-globe agents on mobile phone.

JADE [JADE, 2007] (Java Agent DEvelopment Framework) is a software framework fully implemented in Java language. It simplifies the implementation of multi-agent systems through a middle-ware that claims to comply with the FIPA specifications and through a set of tools that supports the debugging and deployment phase. The agent platform can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another one, as and when required.

4.8 FCGN Agents on JADE

The agent implementation platform we selected is JADE. It is actively

maintained and new versions of the JADE agent development kit are released regularly. The integration of FCGN agents with the JADE platform is shown in Figure 4.4.

Figure 4.4. FCGN Agent-JADE Integration

The integration of JADE and FCGN Agents is divided into 3 layers: (i) The JADE Platform level, (ii) the Adapter Agent level, and (iii) the FCGN Agent level. The Adapter agent is a JADE agent which is able to make use of all the common services that are provided by the JADE agent platform. The Adapter agent is also a wrapper around the FCGN Agent to allow it to execute within the JADE platform. The interaction between the Adapter agent and the FCGN Agent occur through the Adapter Agent Interface (AAI) and the FCGN Agent Interface (FAI). Both sides are only able to interact using these interfaces in order to minimize the dependencies. A similar strategy could be used to adapt the FCGN agents to other agent platforms by changing the Adapter Agent.

The Adapter Agent Interface provides mechanisms to send and receive

messages. It also provides notifications of events. The FCGN Agent Interface provides methods to make use of the FCGN reasoning mechanisms and behaviours. All the available JADE functionalities could still be used by the FCGN Agent Tasks.

The JADE Adapter Agent creates an instance of the FCGN agent when it is started. The FCGN Agent will initialize itself from available configuration files. The JADE Adapter Agent implements several JADE behaviours which then invoke suitable methods in the FCGN Agent through the FCGN Agent interface. For example, a separate behaviour is used to implement timing behaviour such as adding timeout events to the event list. Implementing functionalities into separate behaviours allows for flexibility in changing the behaviours or replacing them with improved replacement behaviours. For example, alternate scheduling mechanisms could be explored by changing the behaviour implementing the scheduling functions.

The execution model for the FCGN agent has been discussed in previous sections. However, to implement the FCGN agent execution model within the JADE Adapter Agent, we need to separate out the components of the FCGN Agent execution model to implementation constructs. Three distinct elements (see Table 4.1) are used to implement the FCGN execution: (i) a message receiver, (iii) a reasoning engine and (iii) a scheduler.

Table 4.1. FCGN Execution Model Implementation Elements

Execution Element	FCGN Agent Component	Adapter Agent
Message Receiver	Sense Unit (Percept, Events, Reflex Actions)	Behaviour for processing received JADE messages
Execution Engine	Goals Unit Knowledge Unit Decision Unit Control unit Communications Unit	Behaviour to dispatch FCGN core logic execution cycle

	(Selected Actions)	
Scheduler	Scheduler Unit	Behaviour to dispatch FCGN tasks
	(Reflex Actions, Selected Actions)	
	Communications Unit	

Figure 4.5 shows the FCGN execution model in a flowchart form. Messages from other JADE agents in the JADE platform must be filtered from the JADE message queue. Messages meant for the FCGN Agent are directed to the Message Receiver component. Messages meant for the FCGN Agent are directed to the Message Receiver component.

Figure 4.5. FCGN Execution Model in Adapter Agent

The FCGN agent is implemented using mark-up SGNML and FCMML (see section 4.2 and 4.3) to describe the structure of the goal-net and fuzzy cognitive maps. The SGNML represents the goals and relationships between the goals. The FCMML describes the fuzzy cognitive maps representing relationships between significant concepts necessary for the reasoning performed by the FCGN agent. Each FCGN agent must be represented by one SGNML file and one or more FCMML file.

The task actions performed by the agent are implemented by FCGNAction java class. The FCGNAction abstract java class is extended to form a class hierarchy of java classes which implement the specific task actions needed in the FCGN agent.

Each FCGN agent contains one or more capabilities (see Figure 4.6). The capabilities represent the modular portions of the agent which can be re-used in different FCGN agents, which are primarily the SGNML, FCMML and FCGNAction classes. Hence, capabilities are essentially collections of the mark-up files and FCGNAction classes in our implementation. The FCGN agent core logic provides the reasoning engine to use the content of the units of capabilities to perform the pro-active goal-oriented behaviour and actions.

Figure 4.6. FCGN Agent Capability Meta-model

The role of the Service Oriented Architecture (SOA) is going to be more important for the design of new systems. It is essential that the FCGN Agent system is able to act as client for services provided by other components; as well as being able to act as a service provider. Web services are a common mechanism for service-oriented architectures. Simple Object Access Protocol (SOAP) is used as the message format in exchanges between the client and provider of a Web service. The most common transport used for the SOAP messages is HTTP (but it is not restricted to HTTP). JADE has a plug-in named the Web Services Integration Gateway (WSIG) which allows the JADE agents to export their agent services as web-service. The WSIG also allows for JADE agents to invoke external web-services. The Figure 4.7 shows the various components of the WSIG.

Figure 4.7. Web Service Integration Gateway

The WSIG has agents which register themselves as “web-service” with the JADE Directory Facilitator (DF). These web-service agents are able to export services usable to external web-service clients. Or, they may act as proxies for service requests to external web-services. The WSIG depends on the JADE (WSIG) gateway agent to act as a middleman between the web-service container and the web-service agents in the JADE container. The agents use FIPA ACL to communicate while web-services use SOAP/XML syntax. The codecs translate between the different message formats used in the web-services and agent services. Besides allowing interoperability at the message and operations level, the WSIG also uses the JADE WSIG Gateway agent to act as middleman between two directory services: (i) UDDI – the directory services used by web-services and (ii) DF – the directory services used by agents. Agent Service Descriptions (SD) registered by the web-service agents with the JADE DF are translated to WSDL by the WSIG Gateway agent, registered as a corresponding web-service in the UDDI registry, and have an endpoint created for the web-service on the Axis web-server to allow for external (SOAP/XML) clients to invoke the agent service. In the reverse direction, new WSDL definitions for external web-services can be registered with the UDDI registry, and these registrations are forwarded to the WSIG Gateway which translates the new SD for registration with the JADE DF. On the new web-services are registered with the DF, other agent will see the web-services as though they are exported agent services. When an agent wishes to use the new service, it makes an agent request using FIPA ACL to the registered agent service. This is forwarded to the WSIG Gateway which then translates it into web-service invocation. The response to the web-service request is translated back to a FIPA ACL INFORM message by the WSIG Gateway and sent back to the agent service requestor. This middleman gateway approach is also adopted by other JADE to Web-service

integration frameworks such as WS2JADE and WSDL2JADE. One potential problem with the WSIG is that the JADE WSIG Gateway agent is a potential bottleneck and a unique point of failure. WS2JADE proposes multi-threaded gateway agents to overcome the problem of performance bottleneck.

Peer-to-peer (P2P) systems are a popular alternative to centralized client-server system. The key advantage of P2P technology lies in its ability to utilize large-scale under-utilized or free resources at the edge of the network. The application areas for P2P include distributed computing, network resource sharing, content sharing and collaboration. In its pure form, a peer-to-peer system has no servers and no functionality is centralized. All nodes of the network are equal peers called servants (*server+client*). Self-adaptation, load-balancing, self-organization, fault-tolerance and the ability to pool together large amount of resources, are some of the key benefits of using peer-to-peer systems. In addition, there are many advantages to combining agent technology with P2P systems such as improving the scalability of the system, increasing the flexibility of the agent system to use different communications networks; and improving the self-healing/self-adaptation and autonomic characteristics of the system.

Interfacing a P2P (see Figure 4.8) network could be implemented through extending the JADE platform services to include a JXTA [JXTA, 2007] (or other P2P platform API) interface. This method of implementation has the advantage of transparency of the agent to agent communication messages; the agent themselves may not be aware of the underlying P2P transport. However, it requires that the agent platform to implement an extensible set of transport services. In our case, the JADE platform does provide a kernel with extensible message transport mechanisms and services.

Alternatively, a gateway agent could be implemented to act as an intermediary for

the JADE agents on one platform and other agents located elsewhere in the P2P network. Such as gateway agent would use the normal JADE messaging framework to communicate with other agents on the platform; and also possess and JXTA (or other P2P platform API) interface to communicate with other gateway agents over the P2P overlay network. This method of interfacing with the P2P overlap network is reproducible on other agent implementation platforms which do not support extension of their message transport services. The disadvantage is that the agent to agent communication is not transparent over the P2P network. The agents that need to communicate must be aware of the gateway agents. Gateway agents must perform agent discovery and routing to appropriate gateway agents.

Figure 4.8. JADE and P2P Interface

For the FCGN Agent implementations we used the extended JADE service approach to take advantage of the extensible JADE kernel. This also avoids the need to

replicate the agent directory services protocols implemented in the JADE platform with another set of similar protocols working between gateways agents running over the peer-to-peer network.

4.9 Summary

Goal-oriented agents can be modelled using the Fuzzy Cognitive Goal Net (FCGN) model. The agent implementation framework discussed allows rapid implementation of the FCGN agents using declaratively defined behaviour in XML (in FCML and SGNML). Service invocations are supported through modules which encapsulate the logic and program code required to invoke a service. Modules also allow us to publish services through web service interfaces.

CHAPTER 5

GOFAAS – GOAL-ORIENTED AND FCM FOR AGENT-MEDIATED AUTONOMIC SERVICES METHODOLOGY

This chapter describes the methodology adopted in applying FCGN agents to services to enable autonomic behaviours. This new methodology is named GOFAAS (Goal-Oriented and FCM for Agent-mediated Autonomic Services) methodology.

5.1 Overview

Definition 4.1: A software methodology is defined by the following components:

- A methodology is a codified set of principles and practices that may be repeatedly carried out to produce software.
- It comprises notations and diagramming (models) used to represent different aspects of the problem space and system (solution space) under consideration. For example the use-case model or structural/class model
- A set of methods to transform the problem space models into solution space models. There maybe several stages between the problem space model and the final solution space model (see figure below).
- A practical software methodology also includes a set of heuristics and guidelines to aid the software creation processes.
- A definition of the problem space for which the methodology applies. A methodology may be generally applicable to a wide range of applications. Or it may apply to a class of applications (for example, real-time system

methodology). A methodology may be applied for software within a particular domain (for example, software in a production manufacturing environment). For a generally applicable methodology, the problem space definition is optional.

Figure 5.1. Transformation of models in methodology

5.2 Phases

The FCGN agent model addresses several pressing issues in the creation of autonomic applications and services for dynamic and complex service environments. FCGN agents are the result of a hybrid of the Goal-Net (GN) model and Fuzzy Cognitive Map (FCM) model. Shen [Shen, 2005] described a practical goal-oriented (GO) methodology for developing agents based on the goal-net (GN) model. The methodology addresses the entire agent based software life-cycle from analysis and modelling, to implementation. The GO methodology did not address the inclusion of fuzzy goals and fuzzy relationships. In this research, we propose an extension to the GO methodology that uses both goal-net and FCM models named the GOFAAS methodology.

The lifecycle of the service development (see Table 5.1) is divided into the

analysis, specification, architecture, design, implementation; and verification phases.

We will use a work-example of a typical service-oriented application to illustrate the use of our methodology. The problem space of the GOFASS methodology is divided into:

- Goal-Oriented Requirement Analysis
- Agent Specification
- Agent Organization and Architectural Design
- Agent Detailed Design
- Agent Implementation
- Agent Verification

Table 5.1. GOFAAS Overview

GOFAAS Phases	Problem Space	GOFAAS Tasks	Methods and Heuristics	Models, Notations and Diagrams
Analysis	Goal Oriented Requirement Analysis	<ul style="list-style-type: none"> • Use cases • Goal-Environment-Task (GET) Cards • Goal identification / capturing • Environment model • Agent Identification • Interactions Identification • Services identification • Resources identification 	<ul style="list-style-type: none"> • Using use cases to identify actors, agents and functionalities • Using Goal-Environment-Task (GET) Cards to identify goals, tasks and environment relationships • Mapping environment variables to goals and goal diagram • Analyze variability in the environment and map into alternatives in the goal model. This is reflected in the goal diagram. • Actors, services, interactions and resources and their dependencies are analyzed and captured in the Actor diagram 	<ul style="list-style-type: none"> • Use Cases • Actor Diagram • Goal Diagram

An Agent Based Approach Towards Autonomic Services

Chapter 5

Specification	Agent Specification	<ul style="list-style-type: none"> • Preliminary Goal Nets defined • Fuzzy cognitive model of action selection mechanism • Preliminary FCMs defined • Agent communication defined. 	<ul style="list-style-type: none"> • The Goal Nets are refined from the goal relationships captured by the goal-diagram • The FCM knowledge based is derived from subject matter expert inputs. • Suitable agent communication protocol is selected based on the requirements of the task. 	<ul style="list-style-type: none"> • Fuzzy Cognitive Goal Net (FCGN) Model • Agent Identification Diagram
Architecture	Agent Organization and Architectural Design	<ul style="list-style-type: none"> • Goal Net Splitting • Agent organization defined • Agent communication protocol refined 	<ul style="list-style-type: none"> • Goal Net Splitting is used to identify agents and functionalities of agents. This is used to define the agent organization. The group of agents form a multiagent system (MAS) • Agent communication protocol is further refined based on re-allocation of functionalities to different agents within the MAS 	<ul style="list-style-type: none"> • Agent Organization Diagram • FCGN-Agent Allocation Mapping and FCGN Model for each agent • Updated Agent Identification diagram
Design	Agent Detailed Design	<ul style="list-style-type: none"> • Goal Design • Transition Design • Protocol Design • FCM design • Adaptation Design 	<ul style="list-style-type: none"> • The FCGN model for each is mapped into a suitable agent classes, data resources, and capabilities. • Agent interactions are mapped into a set of messages. • Adaptation mechanisms enable autonomic properties to be manifested by the system. Changes in the goals affected by environment and other self-* characteristics are translated to re-configuration and behaviour adaptation in the detailed software design. 	<ul style="list-style-type: none"> • Detailed software design document
Implementation	Agent Implementation	<ul style="list-style-type: none"> • Goal Implementation • FCM implementation • Transition Selection Mechanism • Task/Action Implementation 	<ul style="list-style-type: none"> • Agent implementation • Interface to existing system 	<ul style="list-style-type: none"> • Source code • Configuration files • Meta data • Data resources

Verification	Agent Verification	<ul style="list-style-type: none"> • System test • User acceptance Test 	<ul style="list-style-type: none"> • Protocol monitoring and tracing to analyze correct interactions between agents. • Agent monitoring and logging to determine correct behaviour of individual agents. 	<ul style="list-style-type: none"> • Test Plan • Test Specification and Test Cases
--------------	--------------------	---	--	--

5.3 Analysis

The analysis phase focuses on the intentions of the stakeholders and domain modelling. Relevant actors in the organization are identified and analyzed in terms of their goals and intentions. In the GOFAAS methodology, we make use of use cases and GET cards to derive the preliminary functionalities, goals and actors in the system.

The GOFASS methodology tries achieve close alignment with the techniques in conventional software engineering (SE) methodologies in order to make it more understandable and accessible to practising software developers. However, in order to support autonomic services through agent-mediated functions, GOFASS methodology needs some additional procedures and guidelines not used in conventional SE methodologies.

Table 5.2. Comparison of Analysis Phase

Conventional Software Engineering (e.g. Unified Process, Agile/Scrum)	GOFASS
Functional and non-functional requirements is elicited from stakeholders and domain experts is performed via: <ol style="list-style-type: none"> 1) Focus groups 2) User interviews 3) Surveys/Questionnaires 4) Use Cases 5) Quality Function Deployment (QFD) 6) Feature-oriented domain analysis (FODA) 7) Accelerated Requirements Method (ARM) 	Task, Services and Resources <ul style="list-style-type: none"> • Requirements for tasks and functions to be performed, common services, and resources found in the system environment can be derived using similar techniques to conventional SE. • For e.g. User interviews, Use Cases, QFD, FODA, ARM.
Goal-oriented requirements analysis is not commonly used for conventional SE.	Goals and Soft goals <ul style="list-style-type: none"> • AOSE methodologies such as Tropos have models for goals and soft-goals and the possible goal relationships and associations. • GET cards (similar to CRC) are used to

	<p>help identify the goals and soft-goals.</p> <ul style="list-style-type: none"> • A scenario based approach using Use-Case Maps (UCM) [Liu and Yu 2001] may be used in conjunction with the goal requirements analysis to refine the goals. • The goal based models can be transformed to UML notation and models [Alencar et al 2000]
--	--

From the table above, it is observed that for task oriented and functional requirements, both GOFASS and conventional software engineering use similar techniques. For example, detailed Use Cases can be used to derive the requirements for both methodologies. Use cases describe the interaction between a primary Actor (the initiator of the interaction) and the system itself. Each use case is a complete series of events, described from the point of view of the Actor. However, since GOFASS is a methodology for autonomic services, we also have to identify the services that must be provided by the system and identify the resources available in the environment. Both services and resources can also be identified from further refined use cases. In most cases, we can consider resources to be a special type of actor. However, more abstract resource such as bandwidth should not be considered an actor. Services are closely related to a Use Cases. A high-level Use Case may encompass one or more services. However, after further refinement, lower level Use Cases may have a one-to-one correspondence to Service.

The second half of the table shows requirements elicitation which is quite different from conventional software engineering. This reflects the autonomous and proactive nature of agents which are different from conventional software and objects. Goals represent higher level desirable outcomes we want to derive from the system. Shen [Shen, 2005] proposed the use of Goal-Environment-Task cards in his methodology. The GET cards are similar to Class Responsibility Collaborator (CRC) Cards. Each GET cards list out the Goals of an agent or agents, the Environmental

conditions and the Tasks to be performed. Goals can also be identified by examining the various scenarios for the system. One possible technique is Use Case Maps (UCM) [Liu and Yu, 2001]. UCM help visualizing behaviour combined with structure, above the level of message exchanges and component behaviour. The basic idea of UCM is causal paths cutting across organizational structures. The notation represents causal paths as sets of wiggly lines that enable a person to visualize scenarios threading through a system without the scenarios actually being specified in any detailed way (e.g. with messages).

Use cases (see Figure 5.2) are used in UML to derive the different scenarios for the usage of the system. The use cases identify the actors and functionalities of the system.

Figure 5.2. Example of Use Case

From the use case diagram, we can identify possible agents by grouping the functionalities or use cases. This is illustrated in Figure 5.3 which indicates the agents

identified from the above use case.

Figure 5.3. Agent Identification from Use Case

GET cards (see Figure 5.4) are a method developed for the goal-oriented methodology [Shen, 2005] to derive a high-level view of the goals, tasks and environment factors in the system. The GET cards are analyzed to identify groups of goals and their relationships to tasks and environment variables.

Goal: Travel Fast	
Environment	Tasks
Security Alerts	Travel by airplane
Weather	Travel by bullet train
Holiday season	Travel by cruise liner

Figure 5.4. Example of GET card

Common functionalities and tasks are also identified. The results of this GET card analysis may be documented using the actor and goal diagrams based on the Tropos/GRL/i* notations. The Tropos notation uses the following concepts (See Figure 5.6):

Actors

- Actor — an entity that has strategic goals and intentionality within the system or the organizational setting. Actors can be specialized into agents, roles and positions. Actor closely matches the UML Use Case Actor.
- Agent — software agent, that is, a software having properties such as autonomy, social ability, reactivity, pro-activity.
- Role — an abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavour.
- Position — an entity that has strategic goals and intentionality within the system or the organizational setting.

Intentional Elements

- Goal — actors' strategic interests. There are distinguished hard goals from soft goals, the second having no clear-cut definition and/or criteria for deciding whether they are satisfied or not.
- Plan/Task — represents, at an abstract level, a way of doing something. The execution of plan can be a means for satisfying a goal or a soft goal.
- Resource — a physical or an informational entity.
- Belief — the actor's knowledge of the world.

Intentional Relationships

- Dependency — a relation between two actors, which indicates that one actor depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource. The former actor is called the depender, while the latter is called the dependee. The object around which the dependency centres is called dependum. Dependum can be either goal, or resource, or task.
- Decomposition — a relationship between goals or plans representing AND/OR decomposition of root goal/plan into subgoals/subplans. It defines what other elements need to be achieved.
- Means-end — a relationship between goals or plans describing how goals are achieved.
- Contribution — a relationship between goals or plans representing how and how much goals or plans can contribute, positively or negatively, in the fulfilment of the goal.

During the analysis phase, the goals of the system must be examined and decomposed into sub-goals. The goals of the system may be partitioned into 2 main

groups: (1) the functional goals which are aligned to the business goals and processes; and (2) the soft-goals which include non-functional goals such as reliability and self-management/autonomic behaviour. One of the key steps in the analysis phase is to derive the goal tree for the hard-goals and soft-goals. For autonomic behaviour, we must consider high-variability in the soft-goals to account for changes in system configuration necessary in a dynamic environment. Figure 5.5 and Figure 5.6 show graphical notations from the Tropos methodology.

Figure 5.5. Tropos Notations

Figure 5.6. Tropos Actor Notations

For GOFASS, we have decided to align the graphical notations used in Tropos to UML. A set of stereotypes are defined to accommodate GOFASS and Tropos concepts within UML (see Table 5.3). Table 5.3 shows the notations used for the analysis phase of GOFASS. Additional notations introduced for GOFASS related to the environment interface. The environment provides inputs which are perceived by the system through percepts (or sensors/probes/monitors) and events. The system performs actions on the

environment through effectors (or actuators). Some significant occurrences may not be caused by physical effects; such occurrences are modelled as events (for example, the communication received from a new agent about a change in policy; or newly discovered services and resources). For autonomic system, the sensing and actuating components are two key components in the MAPE (Monitor-Analyze-Plan-Execute) control loop. Resource discovery is also an important aspect of autonomic systems, and this can be modelled through the events.

Table 5.3. Analysis GOFAAS UML Stereotypes

Stereotype (short form)	UML metamodel	Description
Actor		
Actor	Actor	An actor is an active entity that carries out actions to achieve goals by exercising its know-how. An actor may optionally have a boundary, with intentional elements inside.
Agent	Actor	Software agent, that is, a software having properties such as autonomy, social ability, reactivity, pro-activity.
Role	Actor	An abstract characterization of the behaviour of a social actor within some specialized context or domain of endeavour.
Position	Actor	An entity that has strategic goals and intentionality within the system or the organizational setting.
Elements		
Goal	Class	Actors' strategic interests. Hard goals have a clear-cut definition and/or criteria for success. They normally model the functional requirements.
Softgoal	Class	Softgoals are goals with no clear-cut definition and/or criteria for deciding whether they are satisfied or not. Softgoals normally model the non-functional requirements of the system.
Plan / Task	Use Case	A task specifies a particular way of doing something.
Resource	Class	A physical or an informational entity.
Belief / Knowledge	Class	The actor's knowledge of the world.
Relationships		

Stereotype (short form)	UML metamodel	Description
OR Decomposition (OR dec)	Aggregation	The decomposition relationship provides the ability to define what other elements need to be achieved or available in order for a higher goal to be achieved. OR decomposition means that only 1 or more sub-goals need to be achieved for the higher goal to be achieved.
AND Decomposition (AND dec)	Composition	AND decomposition means the sub-goals all need to be achieved for the higher goal to be achieved.
Means-ends (mean)	Realization	A relationship between goals or plans describing how goals are achieved.
+ Contribution (+)	Directed Association	A relationship between goals or plans representing how and how much goals or plans can contribute positively in the fulfilment of the goal.
- Contribution (-)	Directed Association	A relationship between goals or plans representing how and how much goals or plans can contribute negatively in the fulfilment of the goal.
Dependency (dep)	Dependency	A relation between two actors, which indicates that one actor that depends, for some reason, on the other in order to attain some goal, execute some plan, or deliver a resource.

Environment Interface

Percept	Class	Input from sensors in the environment.
Event	Class	Significant occurrences that are not directly related to physical effects in the environment. It could be a logical changes or policy changes.
Effectors	Class	Output from the system changes the environment through effectors.

The Actor diagram describes the actors, their goals and the network of dependency relationships among actors. This diagram type is used either to define needs and intentional relationships of the business actors or users of the system.

Figure 5.7. GOFASS Actor Diagram Notation

The set of stereotypes, tagged values, and constraints are defined to accommodate the agent based concepts within UML for GOFASS is shown in Table 5.3. The goal diagram shows the relationship between goals and how top-level goals (super goals) can be decomposed into lower level goals (sub goals). The type of decomposition could be OR decomposition where any of the sub goals can contribute to the overall success of the super goal. Alternatively, the decomposition could be AND decomposition which requires all the sub goals to be fulfilled before the super goal can be achieved (see Figure 5.8).

Figure 5.8. GOFASS Goal Diagram Notation

In addition, autonomic behaviour depends on a sense-control loop between the environment and the system. Hence, we have to analyze the environment surrounding the system. The analyst needs to determine to possible events in the environment and changes in the surrounding resources that may occur. Goal priority and tasks may need to be changed in response to changes in the environment. Several approaches to

autonomicity have been proposed; these are (i) designing the system with multiple possible configurations and behaviours that can satisfy a set of highly variable system goals through re-configuration and behaviour modification during runtime, (ii) using intelligent agents (with planning capabilities and social skills) that augment its own capabilities through cooperation with other agents in a MAS, and (iii) using evolutionary computing to generate multiple system configurations. The approach taken by Tropos and GOFASS is the first approach. The variability in the system environment is reflected as OR/AND options or choices within the system's goal models.

The goal model must be translated in to a system with multiple configurations and behaviours that is able to satisfy all the goals through run-time modifications. Goal Net and Fuzzy Cognitive Goal Nets are able to model the high-variability of the goals and also the corresponding behaviours. This is described in more detail in the next subsection (specification).

5.4 Specification

The specification phase results in a requirements specification which describes all the functional and non-functional requirements for the system-to-be. After the analysis phase, we need to derive the agent model and agent interactions from the goals, softgoals, tasks, services and resources identified during the analysis phase. During the specification phase, one of the main diagrams is the Fuzzy Cognitive Goal Net diagram (see Figure 5.9).

5.4.1 Agent Model

The FCGN agent model is elaborated in more detail in Chapter 3 of this thesis. During the specification phase, we need to derive the complete FCGN model for the

system. The fuzzy cognitive knowledge maps for all fuzzy cognitive transitions also need to be defined. Some preliminary allocation of the FCGN models to agents can be derived by matching the FCGN model functions and the agents identified in the Agent Diagram of the analysis phase.

However, there may be further partitioning of the agents into separate agents if the FCGN model for a single agent is overly complex. This is deferred to the architecture phase when we will define the agent organization and MAS model.

Figure 5.9. GOFASS FCGN Diagram

For the enabling of autonomic services, the high variability system goals identified during the analysis phase must be reflected as alternative goals in the goal diagram of the analysis phase. In the specification phase, these alternative goals in the goals diagram must also be translated into suitable goals relationships within the FCGN model to facilitate the re-configuration of the system behaviour during run-time.

Non-functional requirements captured as softgoals may be used to decide

between alternative behaviours that lead to the achievement of the main goals of the system. In addition, inputs from the environment in the form of percepts or events may also influence the alternative behaviours. To illustrate this, let us suppose that there are security alerts again travel via railway because of dangers from terrorists attacks. This should cause an autonomic application or service to self-adapt away from suggesting railway as a possible mode of transportation. With reference to Figure 5.9, there would be a higher level goal net (see Figure 5.10) that decides between different modes of transportation. This higher level goal-net would also have inputs from security alerts events.

Figure 5.10. Higher Level FCGN

5.4.2 Interactions

Agent model and interactions can be derived from the goals, roles and responsibilities identified earlier in the FCGN model. Interactions between agents occur because the agents involved have certain goals to achieve, and the interactions are a means of achieving the agents' goals.

The ability of agents to interact with other agents is essential, and it is desirable for agent interactions to be flexible and robust. Agent interactions are traditionally specified in terms of interaction protocols, expressed in notations such as Agent-UML, Petri nets, or finite state machines. Interaction protocols are at a low level of abstraction and are message-centric in nature since they are defined in terms of legal message sequences.

The initial step in deriving the agent interactions is the identification of roles and interaction goals. The roles are dependent on the participants of the interactions and the goals can be seen as high level achievement goals needed for the interaction to be successful. The process for developing the interaction goals and roles involves taking the use case scenarios from the analysis phase and examining the use case realizations. For each use case realization that requires interaction between agents (or with the environment), we identify the high-level goal of the interaction, the roles played by the agents, and the type of interaction.

Interaction diagrams are borrowed from object-oriented design to show interactions between agents rather than objects. The Environment could be identified as one party in the interaction. The Interaction diagram is further refined in subsequent steps by further breaking down larger interaction goals into sub-goals. At this stage, the initial interaction diagram is very high-level and lacks details for implementation.

The second step is the refinement and organisation of the interaction goals identified in the previous step. Where possible, the interaction goals identified are broken down into smaller sub-goals and are organised in a hierarchy. Finally, the discrete leaf goals in the goal tree are mapped into specific actions.

The next two steps of the methodology deals with actions. An action is a discrete step towards achieving a goal which is taken by a single agent. The actions are assigned to roles and are structured in a flexible sequence of execution. Once action

sequences have been determined, the remaining step deals with identifying interaction messages and their format. The message format to use could be KQML, FIPA-ACL, SOAP, or the message formats provided by the agent implementation platform.

Figure 5.11. Interaction Diagram

Interactions are also limited by resources (e.g. connectivity) and the skills/capabilities of the agents. This is part of the design constraints that the architect must take into consideration as the development moves from analysis to design.

The technique of decomposing Goal-net into sub-goals is used to translate analysis phase artifacts (Goal nets) into agent models and agent interactions. Each high-level Goal-net must be refined into further sub-Goal nets and sub-Goals. For each lower-level sub-Goal, we must determine if the goal can be achieved by the sole decision of single agent, or the goal can only be achieved through the joint-decision made by a collection of agents. If the decision is not a sole decision of a single agent, this is a clear indication that some sort of interaction must occur with other agents and goal itself can be further refined into one or more interaction goals. For example, we may have a large goal-net for manufacturing, and after decomposing the large goal-net, we have one of the sub-goals as “Negotiate for best price for shaft”. Since the goal is to negotiate for the best price, the interaction goals can be first, providing a suitable starting offer; second, receiving all counter-offers; and lastly, confirming the

successful counter-offer.

5.5 Architecture

The system architecture is a small manageable model of the system structure which describes how the system appears to outside users and its interfaces. It also catalogs the internal components of the system and their relationship to each other. One main task during this stage is to choose a suitable architectural style (for example, based on the organizational structure) that meets the various functional and non-functional goals. The architecture of the system may be derived from the actor diagram and FCGN model from the specification phase. The FCGN model for an agent may be partitioned into more agents by separating out the sub-goal nets of composite goals to be execute by separate agents. For example, in Figure 5.9, the FCGN model can be partitioned into 2 agents. One agent (Main Ticketing Agent) executes the main goal net. Another agent (Air Ticketing Agent) executes the Air Ticket composite goal subnet.

The repartitioning of the FCGN models may create additional agents and require some updates to the agent interaction diagrams. New communications may be required between agents. Referring to Figure 5.9, the creation of the Air Ticketing Agent means there is need to coordinate the actions of the Main Ticketing Agent and the Air Ticketing Agents. The interaction between these two agents can be defined in an interaction diagram.

5.5.1 Mapping and Partitioning

Using the actor diagram and FCGN model from the specification phase, we need to transform or map those analysis artifacts into a suitable architecture. The actor

diagrams are used to identify agents which may take on the roles and responsibilities of one or more analysis actors. If the role of the actor is too complex, it may be divided into several agent roles. This process of refinement must take into account non-functional requirements of the real-system such as performance. For instance, we may decide to implement a simple functionality for 100,000 users using 100,000 simple agents. However, in reality, most agent containers are not able to perform efficiently with such a large number of agents. Hence, the system architect must judiciously assign the roles and responsibilities of analysis actor to actual executing agents.

The FCGN model is used to define the behaviour of FCGN agents. The FCGN model must be further refined to take into account the partitioning of the system into multiple agents. For instance, a complex Goal-Net may be subdivided into several levels of simple Goal-Nets. One of the simple Goal-Nets may be partitioned off to be executed on an independent FCGN agent. The original agent will now interact with this newly instantiated FCGN agent to get some of the results it needs to satisfy its goals. The FCGN agents must execute within an agent container which may be designed using conventional software architecture pattern/style. Just as the refinement process from actor diagrams need to consider resource constraints, when partitioning the FCGN models into agents, we must consider the complexity of each agent that results and perform some balancing. Normally, the architect must avoid have just one large highly complex FCGN agent, and a swarm of simple FCGN agents (The exception being the case of highly specialized hardware platforms such as swarm robotics).

5.5.2 Patterns

Many of the current conventional software architectures are based on supporting object-oriented programs and architecture. However, these may still be of value in

agent based systems. In many systems, different architectural patterns/styles may be used for different portions or sub-systems. In order to implement different parts of agent execution platforms, some of the possible conventional architectural patterns/styles include:

- Client-server: Client/server describes the relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfils the request.
- Pipes and filters: A filter is a computer program to process a data stream. Pipes form a chain of processing elements where the output of one element is the input of the next processing element.
- Plug-in: A plug-in is a computer program that interacts with a host application to provide a certain, usually very specific, function "on demand".
- Model-View-Controller: Model-View-Controller (MVC) is an architectural pattern where an application is split into separate layers that run on different computers: presentation (view & controller), domain logic (model), and data access (model). In MVC, the presentation layer is further separated into view and controller.
- Repository/Data-centric architecture: A data centric architecture is a software architecture in which databases play a crucial role.
- Event Driven architecture: Event-driven architecture is a software architecture pattern promoting the production, detection, consumption of, and reaction to events (defined as "a significant change in state").
- Layered architecture: A logical division of the software into multiple levels of abstraction. The level of abstraction increases as we move from the lowest level/layer towards the upper levels. Lower layers are often concerned with physical resources such as disk storage space.

- Tiered architecture: Multi-tier architecture is a distributed architecture in which an application is executed by more than one tier of software. A tier is a physical structuring mechanism for the system infrastructure.
- Component-based architecture: The system architecture is built from functional or logical components with well-defined interfaces used for communication across the components. Components are considered to be a higher level of abstraction than objects.
- Service-Oriented Architecture: Service-Oriented Architecture is a software architecture where functionality is grouped around business processes and packaged as interoperable services.

For example, the JADE platform used in our implementation has layered software architecture. In addition, the JADE platform adopts a Distributed Composition Filter approach in its architecture. An incoming filter chain has filters invoked whenever the object receives a method call. An outgoing filter chain has filters that are invoked when the object is about to call another object's method. Each Service is "sliced" over the distributed nodes of the platform.

Agents are organized into a multiagent system (MAS). Pattern/styles for agent and MAS architecture are different from the conventional software because agents are at higher levels of abstraction when compared with objects. Agents are autonomous, pro-active, goal-directed, reactive and communicative. Some patterns/styles for agent-based systems include:

- Hierarchical organization: This model mirrors a bureaucracy in the real world organizations. There is a hierarchy of agents with different roles and responsibilities.
- Blackboard: In the blackboard architecture, a common knowledge base (the

"blackboard") is iteratively updated by a diverse group of specialist knowledge sources, starting with a problem specification and ending with a solution.

- Space based architecture (tuple based coordination space): With a space-based architecture, applications are built out of a set of self-sufficient units, known as processing-units (PU). PUs communicate via a tuple space. A tuple space is an implementation of the associative memory paradigm for parallel/distributed computing.
- Stigmergy based model: Stigmergy is a mechanism of spontaneous, indirect coordination between agents or actions, where the trace left in the environment by an action stimulates the performance of a subsequent action, by the same or a different agent.
- Economy-based model: In the economy-based architecture, all services and resources have a value a value denoted in some currency unit within the virtual economy. The pricing of the agents is related to the utility derived from the agent services. The prices in the virtual economy are used as the basis for allocation of resources and negotiations.

FCGN agents may use one or more of these agent based architectural patterns/styles depending on the requirements of the problem domain.

5.6 Design

This phase of the development involves detailed design of the software artifacts derived in the software architecture phase. The detailed design phase is intended to introduce additional details to each of the architectural components derived in the previous phase. The software design must be refined to a stage where it provides

sufficiently concrete information for implementation to begin. Figure 5.25 shows the relationship between the design and implementation artifacts.

The architectural artifacts must be further elaborated into detailed design artifacts that could be implemented. The FCGN model which was an abstract model during the specification phase and architectural phase must now be refined to a more concrete form in the design phase. The mapping is indicated in the table below.

Table 5.4. Design Phase Artifacts and Processes

Architectural artifacts	Design artifacts	Software Engineering Process
FCGN Net Graph	Agent (represented by instance of class)	<ul style="list-style-type: none"> Agent classes are designed as part of the executing framework.
FCGN States, FCM	FCGN States (represented in SGNML file) FCM (represented in FCMML file)	<ul style="list-style-type: none"> Executing framework designed to read information files and execute the FCGN states. The SGNML transfer format is used to encode the FCGN states and transitions. The FCM transfer format is used to encode the FCM
FCGN Transitions, FCM	FCGN Transitions (represented in SGNML file) FCM (represented in FCMML file)	<ul style="list-style-type: none"> Executing framework designed to read information files and execute the FCGN transitions. The SGNML transfer format is used to encode the FCGN states and transitions. The FCM transfer format is used to encode the FCM
Environment Events / Percepts	FCGN Events / Percepts	<ul style="list-style-type: none"> The interface to the environment is very specific to the surrounding conditions. The event and percept information are designed as classes. These classes may need to interact with physical sensors and instruments.
FCGN Tasks	FCGN Tasks Service-oriented artifacts such as service description, service interface, etc.	<ul style="list-style-type: none"> The interface to the environment is very specific to the surrounding conditions. Actions which influence the environment are designed as classes which may need to interact with physical actuators. Actions which only affect internal states are designed as classes (may be implemented as compiled language or as interpreted script). The design for service interfaces

		and description follow conventional SE techniques. The format of service interface messages may be SOAP or HTTP/REST based.
--	--	---

5.7 Implementation

This phase involves creating the necessary source code for the agent based system. An objective of the system is to allow many behaviours of the system to be defined declaratively, for example through mark-up language such as XML.

The implementation phase in agent development is process of creating source codes to be compiled and executed on a suitable agent implementation platform. In our case, we have chosen the JADE agent development platform. The JADE platform is just as one of many possible implementation possibilities. During the implementation phase, the FCGN specific execution framework is coded and interfaces are provided to execute the FCGN agent in the JADE agent container. The JADE agent container is implemented using the Java language and requires the Java Virtual Machine (JVM) to execute. The amount of implementation code for the FCGN execution framework is drastically reduced by using JADE because the agent container provides many of the basic routines needed for agent instantiation, agent passivation, agent communications, agent directory, and services discovery. The FCGN execution framework allows for the declaration of FCGN agent control flow and behaviour using the SGNML and FCMML transfer format. The capabilities of the agent are coded as Java classes. These Java classes implement the library of behaviour or actions supported by the agent. The scheduling and executing of these actions is implemented by the FCGN execution framework code.

Java allows for scripting engines to be added to the JVM. JSR 223 (Scripting for the Java Platform) is a specification to describe mechanisms allowing scripting

language programs to access information developed in the Java Platform. JSR 233 is included in Java version 6 and above. For the FCGN execution framework, it is possible to include actions that are implemented in one of the scripting languages. For example, we could execute Python script (using Jython), Ruby script (using JRuby), interpreted Java (using BeanShell), Javascript (using Rhino), Groovy or Judoscript.

5.8 Testing and Verification

In this phase, the system behaviours are exercised to determine if it satisfies the goals that were identified during the analysis phase. Both functional goals and non-functional goals should be verified. In a complex system, 100% test coverage of every single path through the system is not feasible or economically viable. New research suggests that the best strategy is an iterative one. Beginning with a random or stochastic choice of different environment conditions during the tests, we use the test results to select conditions that yield the most error conditions for the next round of testing. Gradually, the number of failure conditions will as the developer fixes the problems that occur in the most common states of the system. Recent research indicates that most systems only reside in a few common states even when the total number of possible states is very large.

Testing and verification are crucial in agent systems. One of the benefits of using executable specifications from the verification viewpoint is that we can use simulation techniques to run the FCGN model defined in SGNML and FCML transfer syntax to predict the possible behaviours of the system. The test cases for the actual system should also cover the scenarios expected from the simulation results. The simulation results could be compared with the actual system behaviour after implementation.

Section 5.10 contains a worked example (see section 5.10) of the various

GOFASS phases applied in the context of a scenario of dynamic goal-oriented environment. For the analysis phase, the Use Cases, Tasks, Service, Resources, Goals and Agents are identified. In the specification phase, the FCGN are derived. The initial FCGN are further refined. During design, we design the agent classes, and encode the necessary information about the FCGN in SGNML and FCM. The tasks and actions of the agents must be implemented as class methods or using scripts to be executed using the scripting engine. In the test and verification phases, we can verify the actual behaviour of the system against that specified with the FCGN.

5.9 GOFASS Modelling Language (GOFASSML)

The modelling language is at the core of the methodology. In this section, the abstract syntax of the language is defined in terms of the UML metamodel. The following UML standard approaches are used. The metamodel is organized into 4 levels.

Table 5.5. GOFASS Model Levels

Level	Description	Examples
Meta-Metamodel	Specifies language structure	Entity
Metamodel	An instance of the Meta-metamodel Defines knowledge level entities	Actor, Agent, Goal, Plan, Beliefs, Events
Domain	An instance of the Metamodel Models application domain concepts	Agent A
Instance	An instance of the Domain model	nodeAgentA: instance of Agent A

The Meta-Metamodel level provides the basis for metamodel extensions. The Metamodel level provides constructs for modelling knowledge level entities and concepts. The Domain level contains representations of application domain specific entities and concepts. The Domain level contains instances of the Metamodel level. The Instance level contains instances of the Domain level.

A GOFASS model is essentially a directed graph whose nodes are instances of metaclasses of the metamodel. The arcs are instances of metaclasses representing relationships between them. Each element in the model has a graphical representation (based on extended standard UML graphical representations). We use several type of diagram for visualizing the model.

An actor diagram shows the dependencies between entities called actors. Arcs connecting two nodes indicate dependency between the two nodes. The arc has a dependum in between the dependent nodes.

A goal diagram shows the perspective of a specific actor. It is drawn as graph where the arcs indicate the relationship between goals (for example, AND decomposition, OR decomposition).

AUML activity diagrams and AUML interaction diagrams are used to represent the agent's capabilities and plans; and the agent's interactions.

According to the specific process development phase we are considering, we can define different views of the model. For instance, the early requirement view of the model will consist of a set of actor and goal diagrams. The detailed design view will be composed of a set of AUML diagrams specifying the agent's microlevel.

FCGN Diagram is drawn as a directed graph of the goals and relationships between them. The FCGN graph nodes are either goal states or transitions. The FCGN arcs represent relationships between the nodes. FCGN Goal states may be simple goals or composite goals. They represent some intended ends to be achieved. FCGN

Composite goals are goals that can be decomposed into a lower level sub-goal net. The FCGN transitions represent behaviour or actions that must be taken in order to achieve the goals.

5.9.1 Actor

A portion of the GOFASS metamodel concerning the concept of actor is shown in the UML class diagram (see Figure 5.12). Actor is represented as a UML class. An actor can have 0..n goals. The UML class Goal represents both hard goals and soft goals. A goal is wanted by 0..n actors, as specified by the UML association relationship. An actor can have 0..n beliefs and, conversely, beliefs are believed by 1..n actors. An actor dependency is a 4-way relationship represented as a UML class. A dependency related a depender, dependee, and dependum.

An actor dependency is a 4-way relationship represented as a UML class. A dependency related a depender, dependee, and dependum.

Figure 5.12. Actor Concept

5.9.2 Goal

The concept of goal is represented by the class Goal in the UML class diagram (see Figure 5.13). The distinction between hard and soft goals is captured through a specialization of Goal in Hardgoal and Softgoal subclasses.

Goals can be analyzed from the point of view of an actor, performing means-end analysis, contribution analysis and AND/OR decomposition.

Means-end Analysis is a ternary relationship defined among an actor, whose point of view is represented in the analysis, a goal (the end) and a plan, resource or goal (the means). Means-ends analysis is a weak form of analysis, consisting of a discovery of goals, plans or resources that can provide the means for achieving a goal.

Contribution analysis is a ternary relationship between an actor, whose point of view is being represented in the analysis, and two goals. Contribution analysis strives to identify goals that can contribute positively or negatively towards the fulfilment of a goal. A contribution can be annotated with +,++,--,-- for positive and negative contributions. Contribution analysis is often used to evaluate non-functional (quality) requirements.

AND/OR decomposition is a ternary relationship between a root goal and its subgoals. This relationship defines an AND or OR decomposition of a root goal into subgoals.

Figure 5.13. Goal Concept (Decomposition Relationship)

Figure 5.14. Goal Concept (Means-Ends Relationship)

Figure 5.15. Goal Concept (Contribution Relationship)

5.9.3 Plan

The concept of plan in GOFASS is specified by the class diagram (see Figure 5.16). Means-ends analysis and AND/OR decomposition, defined above for goals, can be applied to plans also. In particular, AND/OR decomposition allows for modelling the plan structure.

Figure 5.16. Plan Concept

5.9.4 FCGN Goal

Fuzzy Cognitive Goal Net (FCGN) is visualized as a graph where the nodes are goals or transitions. The nodes are linked by arcs which indicate relationships between the nodes. Goals are divided in to atomic or simple indivisible goals; and composite goals which can be decomposed further into yet another sub-level of FCGN.

5.9.5 FCGN Transition

Transitions indicate the possible action selection or behaviours manifested by the agent. The types of FCGN transitions defined are (i) sequential, (ii) conditional, (iii) probabilistic, and (iv) fuzzy cognitive. Each of these transition type use different action

selection mechanisms to decide on appropriate behaviour. Sequential transitions essentially perform an unconditional execution of a list of tasks sequentially. This is intended for simple procedures that may need to be taken. Conditional transitions use a series of rules or IF-THEN conditions to decide on tasks performed. Probabilistic transitions use Bayesian techniques to decide on actions based on the expected outcomes.

5.9.6 FCGN Arc

FCGN Arcs indicate the relationships between goals and transitions. In the case of diverging or fan-out arc connections there are two possible types of relationships indicated by the FCGN arcs. Concurrent (conc in short) indicate simultaneous execution of parallel paths. Choice indicates that only one among multiple paths must be taken.

In the case, of merging or fan-in arc connections, there are two types of relationships. Synchronize (sync in short) indicate that all conditions on the merging arcs must be satisfied (i.e. all goals are true or all transitions are fired) before the next action is performed. Join indicates that only one or more of the conditions on the merging arcs need to be satisfied before performing the next action.

Figure 5.17. Fuzzy Cognitive Goal Net (FCGN) Concept

5.9.7 FCGN Event, Percepts and Route

The FCGN Event and Percepts nodes represent interfaces to the external world. Percepts are signals sensed from the external environment. Events are significant outcomes that may be triggered by particular percepts. Events may also be triggered by logical conditions not connected to external world inputs. Routes are arcs that connect percepts and events to goals. They show the signal routes from the external inputs to goals that are affected by them.

The FCGN event affected FCGN goal nodes are mapped to FCM concept nodes. As explained in Chapter 3, the FCM concepts nodes are used for FCM computation to

determine suitable goals during the execution cycle of the fuzzy cognitive goal net.

Figure 5.18. Events, Percepts and Routes Concept

5.10 Worked Example

This section illustrates how the GOFASS methodology is used through a worked example. The motivating scenario for the example is described followed by a step-by-step application of the GOFASS methodology.

5.10.1 Motivating Scenario

FCGN agents are able to take advantage of their goal-autonomy to solve service orchestration problems in a highly dynamic grid environment. This application scenario describes the use of a FCGN multi-agent system (MAS) in the allocation of resources for a composite web service among a number of geographically distributed grid nodes.

Figure 5.19. Application scenario for FCGN MAS

Let us assume there is a composite service (see Figure 5.19 above) which performs a complex search algorithm over a large data-set. Three services, services A, B and C may be composed together to provide the composite service. Only 2 of the services need to be used at any time. A FCGN agent is used to allocate the services to be used to provide the composite service. The agent knowledge model for the FCGN is shown in Figure 5.20.

The agent should respond rationally to the change of environment when there is a sudden loss of bandwidth to a cluster of powerful compute nodes. The load should migrate to the resource which is most available. This is accomplished through the interaction of the agents.

Service orchestration using the Business Process Execution Language for Web services (BPEL4WS) [Andrews et al., 2003] had been proposed by IBM, Microsoft and BEA. BPEL4WS is for describing Web services composition flow models in business process workflows. Even though BPEL4WS provides means for the XML specification of the underlying process, BPEL4WS is basically a static composition approach whereby the specifications are written manually and no dynamic assembling of complex flows from atomic message exchanges. A dynamic orchestration method is

needed for autonomic services orchestration, and the FCGN based service orchestration helps to fill this need.

Figure 5.20. Agent knowledge model for scenario 1

5.10.2 Detailed Worked Example

Analysis

The use cases for the example are as follows:

Figure 5.21. Use Case based on scenario 1

Figure 5.22. Agent Identification based on scenario 1

Figure 5.23. Goal Diagram for scenario 1

Specification

The primary goal of the agent is to deliver maximum value through composite service. The Goal Net model for the FCGN is shown in Figure 5.24. We introduce a new type of transitions into the Goal Net model (marked as fuzzy). These are fuzzy cognitive transitions. The action selections within the fuzzy cognitive transitions are determined through FCM inference functions.

Figure 5.24 shows the Goal Net for allocating the grid service nodes. The labels in the model are as follows:

- g0:** Allocate service node. This is the overall goal of the FCGN agent
- g0.s:** Branch start for sub-goal net
- g0.e:** Branch end for sub-goal net
- g1:** Service nodes discovered
- g2:** Service nodes allocated (with link C up)
- g3:** Service nodes allocated (with link C down)
- t1:** Discover available service nodes for the composite service
- t2:** Request service nodes (link C up)
- t3:** Request service nodes (link C down)
- t4:** Confirm service nodes.

Goal g1 (in Figure 5.24) has to make a choice of transition t2 or transition t3. It makes use of the FCM model of the value of the service provided. When the link C environment variable is detected to be disrupted, the value of goal g3 would increase relative to goal g2. The speed of switching goals occurs rapidly because of the fast convergence of the FCM.

Figure 5.24. FCGN Diagram based on scenario 1

Figure 5.25 illustrates how the super-goal g1 is decomposed into a number of sub-goals. The sub-goals are used in turn to identify 3 agents: Agent A, Agent B and Agent C (only Agent C expanded in the figure). Each of these agents is executed at Grid node A, Grid node B and Grid node C respectively.

Figure 5.25. Sub-goals of g1 and agents

The labels in the model are as follows:

- g1:** Service nodes discovered
- g1.s:** Branch start for sub-goal net
- g1.e:** Branch end for sub-goal net
- g1a:** Service nodes discovered
- g1b:** Service nodes allocated (with link C up)
- g1c:** Service nodes allocated (with link C down)
- t11:** Discover available service nodes for the composite service
- t15:** Complete the service

- g1c:** Service nodes discovered
- g1c.s:** Branch start for sub-goal net
- g1c.e:** Branch end for sub-goal net
- g11:** Service nodes discovered
- t12:** Service nodes allocated (with link C up)
- t13:** Service nodes allocated (with link C down)

When a composite goal and subnet are identified as a new child agent, a connection state (atomic state) is left on the Goal Net of the parent agent. In Figure 5.25, the connect state for Agent C is labelled “ConnectC”. The connection state acts as a synchronization point between the parent agent and the child agent. When each of

the child agents has discovered the resources available on its grid node, the results are collected back at the parent node (which is currently evaluating composite goal g1).

Architecture

In architectural design, we define the global system in terms of the subsystems and the interconnections, consisting of data and control flows, between the subsystems. The system actors are mapped to a set of software agents with particular capabilities. The mapping of actors/agents to capabilities can be shown by a capabilities table (see Table 5.6).

Table 5.6. Capabilities Table

Actor/Agent	Capabilities
Agent A Agent B Agent C	1. get information about link 2. get information about 3. store service node beliefs 4. execute service node goal net
Service Management Agent	5. get information about availability of nodes 6. store composite service beliefs 7. execute composite service goal net

The term “capability” is used for different purposes within the agent research community. For us, the term “capability” is used to denote a modular group of beliefs, goals, plans and capabilities. This concept of an agent module (and the term capability) was proposed by Busetta et al. [Busetta et al., 2000]. Capabilities allow for the packaging of beliefs, goals and plans into agent modules which can be reused wherever they are needed. Essentially, the capability does not contain reasoning or decision process. In an agent, with several capabilities, the reasoning mechanism is shared by the different capabilities. A capability can be reused in several different

agents.

Design

During the detailed design, we focus on the formation of individual agent capabilities and interactions. The software agent capabilities and plans are mapped to suitable constructs for implementation.

Implementation

Figure 5.26. Mapping FCGN Agent Design To Implementation

During the implementation phase, the detailed design software notions such as agents and goals are mapped into platform specific constructs such as code and

declarative configuration files. Note that an agent-oriented platform may not always be used to implement the design. However, in our worked example, we have assumed that the JADE platform will be used as the foundation layer for our implementation.

Testing and Verification

During the testing and verification phase, we must test the system to ensure that it performs according to the original specifications. In addition, there is a need to check that the original user requirements have been addressed by the system. In the case of our motivating scenario, we must test the response of the system when there are link failures to each of the service nodes. The response times taken for the system to react to failures and to take action are also critical.

5.11 Summary

This chapter walks through each phase of the GOFAAS methodology step-by-step. Beginning with an analysis of the user needs from the system-to-be and the surrounding environment, we are able to gradually derive models, architecture and design of the system-to-be to meet the user needs. The non-functional requirements such as the autonomic behaviours are also derived and catered for in the design.

CHAPTER 6

EVALUATION

6.1 *Evaluation of AOSE*

AOSE is a key factor for introducing agent-based systems to the industry. Several agent-oriented methodologies have been proposed over the last decade. However, the evaluation of these different AOSE has only recently attracted the attention of the research community [Sturm and Shehory, 2003]. This chapter focus on evaluating the GOFASS methodology using an evaluation framework based on metrics/feature analysis. In addition, we shall use empirical methods (case studies and surveys) to obtain objective data in our evaluation of the GOFASS methodology.

6.1.1 Comparison of AOSE evaluation techniques

Sturm and Shehory [Sturm and Shehory, 2003] performed a feature-based evaluation of several Agent Oriented Software Engineering (AOSE) methodologies using criteria that included software engineering related criteria and criteria relating to agent concepts.

Dam and Winikoff proposed additional metrics for software engineering and management decisions [Dam and Winikoff, 2003]. These additional metrics were introduced in the pragmatics component of the evaluation framework.

Sabas, Badri, and Delisle [Sabas et al, 2002] suggest a multi-dimensional framework containing criteria within each of the following aspects: methodology, representation, organization, cooperation, and technology. These criteria are used as differentiators for comparison purposes. The results of comparisons are described by a

two-dimensional array containing criteria (row wise) and methodological names (column wise). Each intersection is marked: "Y" for Yes, "N" for No, "P" for possible, or simply blank.

A framework using goal-question-metric (GQM) is proposed by Cernuzzi and Rossi [Cernuzzi and Rossi, 2002]. The proposal makes use of feature-based evaluation techniques but metrics and quantitative evaluations are also introduced. The significance of the framework is the construction of an attribute tree, where each node of the tree represents a software engineering criterion or a characteristic of agent-based system. Each attribute is assigned a score and the score of attributes on the node is calculated based on those of its children. They have applied that framework to evaluate and compare two AOSE methodologies: the Agent Modelling Techniques for Systems of BDI (Belief, Desire and Intention) Agents and MAS-CommonKADS.

In [O'Malley et al., 2002], O'Malley and DeLoach proposed a number of criteria for evaluating methodologies with a view to allowing organisations to decide whether to adopt AOSE methodologies or use existing OO methodologies. Although they performed a survey to validate their criteria, they do not provide detailed guidelines or a method for assessing methodologies against their criteria. Their example comparison (between MaSE and Booch) gives ratings against the criteria without justifying them. Their work is useful in that it provides a systematic method of taking a set of criteria, weightings for these criteria (determined on a case by case basis), and an assessment of a number of methodologies and determining an overall ranking and an indication of which criteria are critical to the result.

Lin et al. [Lin et al., 2007] presents a recent methodology for the evaluation of agent oriented software engineering techniques. Lin describes what properties are necessary to form an Agent society with the express purpose of achieving system-wide goals in MAS.

6.1.2 Proposed Evaluation of GOFASS

There are several major evaluation techniques [Siau and Rossi, 1998] for methodologies:

- Feature-based comparison — a comparison that is based on a set of features to be examined.
 - Pros: The advantage of this technique is that it has been applied successfully in many cases and that it can be performed independently of external resources such as an active industry partner.
 - Cons: The major shortcoming of this technique is its subjectivity in the selection of features and in the evaluation itself.
- Metrics — the metrics approach analyzes the complexity of a method. This may consist of analyzing the number of constructs. A construct is a building block of the methodology (for example, in UML a class).
 - Pros: It can be performed independently of external resources.
 - Cons: Subjectivity in deciding upon the constructs to be considered and their granularity. In addition, a lot of empirical work is needed to validate the metrics.
- Empirical evaluation techniques — include surveys, laboratory and field experiments, and case studies.
 - Pros: Objective, empirical data collected.
 - Cons: These techniques usually require the participation of organizations and practitioners. This is very difficult to achieve and requires many resources. When applying these techniques, with only minor cooperation of the organization and the practitioners, this would place a major obstacle.

- Meta-modeling — the meta-modeling approach for evaluating analysis and design methodologies is used as a formal approach to compare methodologies. Such an evaluation is done by (1) building a general meta-model for the ultimate methodology; (2) building a meta-model of the evaluated methodology (or using an existing meta-model thereof); and (3) comparing the aforementioned meta-models. This technique is similar to the feature-based comparison but introduces a more objective comparison technique.
 - Pros: The comparison is done in a formal way (i.e., using the meta-model), the comparison itself is objective.
 - Cons: Common accepted meta-model for MAS for unbiased comparison.
- Ontological evaluation - the ontological evaluation approach proposes to use ontological concepts to evaluate methodologies. The constructs within a modelling method are mapped to the ontological constructs.
 - Pros: The advantage of this technique is that an external, objective body defines the required constructs and that the evaluation is required to identify the constructs within the evaluated modeling method.
 - Cons: Doubts concerning choice of the model for describing a system's ontology. Moreover, there are no guidelines for identifying constructs within a methodology.

The proposed evaluation of GOFASS uses a combination of metrics approach together with the case-studies and survey. We introduce more objectivity in the comparisons of the metrics by using the goal-question-metric (GQM) attribute tree [Cernuzzi and Rossi, 2002]. The case-studies and surveys provide objective empirical data in our evaluation of the GOFASS methodology.

Step one in our evaluation is the application of the Goal-Question-Metric GQM

[Cernuzzi and Rossi, 2002]. The main purpose of this step is to determine measurements and criteria to take into account to reach the objectives. The main objective of the evaluation is to: “Highlight the strengths and weakness of a methodology for the analysis, design, implementation and deployment of Multi-Agent Autonomic Services by considering different perspectives”.

Beginning with the results of the GQM, an attributes tree is created (see Table 6.1). The objective of creating an attributes tree is to identify the more general criteria and then to sub-divide them into finer criteria. The finer criteria are a set of quantifiable ones (these are shaded in Table 6.1). The discrete values are entered directly according to the scoring scale proposed below. For the more general criteria which have the average attribute type, their values are the average of all the sub-criteria. For example, if sub-criteria 1.1.1 is 4, 1.1.2 is 3, and 1.1.3 is 2; the average value for higher sub-criteria 1.1 is $(4+3+2)/3 = 2$. We may consider using a weighted average instead of simple average in future versions of the evaluation framework.

Table 6.1 shows the attributes tree model we used for the evaluation. The attributes are divided into 4 major sections [Sturm and Shehory, 2003]: (1) Concepts and Properties, (2) Notation and Modelling Techniques, (3) Process and (4) Pragmatics. We have introduced additional attributes from Cernuzzi [Cernuzzi and Rossi, 2002], Lin [Lin et al., 2007], and Dam [Dam and Winikoff, 2003]. In addition, to convert the qualitative assessment to quantitative metric values, we use numerical values for the discrete evaluation type. We propose a scale of 0 to 5 as follows:

0. Indicates that the methodology does not address the property.
1. Indicates that the methodology refers to the property but no details are provided.
2. Indicates that the methodology addresses the property to a limited extent. That is, many issues that are related to the specific property are not addressed.
3. Indicates that the methodology addresses the property, yet some major issues are lacking.
4. Indicates that the methodology addresses the property with minor deficiencies.

5. Indicates that the methodology fully addresses the property.

To simplify the process of comparison, we define a normalised scale for the attributes. A Ratio Scale type is adopted for the following reasons:

- It preserves the ordering, the size of intervals between entities and the ratios between entities.
- All arithmetic can be meaningfully applied to the classes in the range of the mapping.

To each attribute numeric values may be assigned, in the range of 0 to 10, that may be mapped according to the following rules. For those discrete evaluation types, the formula applied is:

$$M = (V_n * 10) / \text{Max}$$

M represents the mapping result

V_n represents the previously evaluated value

Max represents the maximum evaluated value between methods

For those measurement that present inverse results (greater value implies worst behaviour), the inverse formula is applied:

$$A = (\text{Min} * 10) / V_n$$

A represents the mapping result

V_n represents the previously evaluated value

Min represents the minimum evaluated value between methods

For non-discrete evaluation values, we use a simple average of the sub-divided criteria to obtain the normalized values. The normalized scale allows for evaluation criteria that are absolute continuous values (i.e. not a 0-5 scale) to be used together with the discrete values. For example, we may have a criterion using mean-time-between-failure (MTBF). MTBF is measured in hours (with a larger value being better).

Applying the normalizing scale, we can combine it with other discrete values.

Table 6.1. Attributes Tree Model

Attributes Tree		Evaluation Type	Detailed Question(s)
1	Concepts and Properties	Average	A concept is an abstraction or a notion inferred or derived from the problem domain. A property is a special capability or characteristic. Here we are interested to know the degree to which agent concepts and properties are core to the methodology.
1.1	Autonomy	Discrete	Agents encapsulate some state and make decisions about what to do based on this state and their own objectives
1.2	Reactiveness	Discrete	Agents are able to respond in a timely fashion to changes that occur in their environment
1.3	Proactiveness	Discrete	Agents are able to act in anticipation of future goals by taking the initiative
1.4	Mental notions	Average	Use of mental notions in methodology
1.4.1	Belief	Discrete	Agents have to keep information about the environment, the internal states that may hold and the actions it may perform
1.4.2	Goal/Desires	Discrete	Agents may adopt a set of goals that may depend on the actual internal state
1.4.3	Action/Intentions	Discrete	Agents may have plans they may employ to achieve their goals or respond to events they perceive
1.5	Interactions	Average	Agents interact between themselves and with the environment.
1.5.1	Sociality	Average	Agent interactions with each other
1.5.1.1	Organisational relationships among agents	Discrete	When agents interact there is an organisational context that defines the nature of the relationships between the agents and shapes their behaviours.
1.5.1.2	Types of agent interactions	Discrete	May vary from information exchanges, to performing a particular action, to co-operation, to negotiation and competition
1.5.1.3	Collaboration	Discrete	Agent has methods to cooperate with other agents to achieve goals. Agents have obligations (conditions to comply) and authorizations about their relationships with other agents
1.5.1.4	Communications	Discrete	There are protocols or mechanisms defined for agent interactions. Different agent interactions (e.g. negotiation, competition) imply different conversation process and agent-communication language.
1.5.2	Interactions with the environment	Discrete	Agents receive inputs related to the state of their environment and they may modify their environment through effectors
1.5.3	Concurrency	Discrete	Agents have to perform multiple tasks concurrently
1.5.4	Multiple control / interests	Discrete	Multiple loci of control (no central control). Agents make decisions at runtime to satisfy their own goals. These may be in conflict with the goals of other agents. Some

An Agent Based Approach Towards Autonomic Services

Chapter 6

			governance mechanism is imperative.
1.5.5	Subsystems interaction	Discrete	Agents may be grouped into subsystems that interact among themselves.
1.6	Autonomicity	Average	Degree to which system can sense their environment and make automatic changes to adapt to new conditions in order to return to a desired operating region.
1.6.1	MAPE loop	Average	Degree to which the MAPE (Monitor, Analyze, Plan, Effectors) loop is a part of the system
1.6.1.1	Monitor/Sensing	Discrete	Sensing the environment
1.6.1.2	Analysis	Discrete	Analyze events for significant triggers
1.6.1.3	Planning	Discrete	Planning a suitable course of actions
1.6.1.4	Effectors/Actuators	Discrete	Ability to change the environment and internal state
1.6.1.5	Knowledge	Discrete	Information about its internal state and the state of the environment.
1.6.2	Adaptive	Discrete	Capability to adjust behaviour according to changes
1.6.3	Aware	Discrete	Ability to sense the environment and its changes
1.6.4	Automatic	Discrete	Ability to perform autonomous actions
1.7	Service orientation	Average	Degree to which system supports services concept.
1.7.1	Composability	Discrete	Ability to aggregate or combine services
1.7.2	Orchestration and choreography	Discrete	Degree of coordination between services
1.7.3	Virtualization	Discrete	Degree to which physical resources are abstracted
1.8	Business process support	Average	Support for business process management
1.8.1	Event notification	Discrete	Events drive processes
1.8.2	Long-lived transactions	Discrete	Processes and transactions are persistent for extended periods of time
1.8.3	Trust relationships	Discrete	Relationships between elements are determined by different levels of trust.
2	Notation and Modelling Techniques	Average	Notations are a technical system of symbols. A modelling technique is a set of models that depict the system at different levels of abstraction and different system aspects.
2.1	Accessibility	Discrete	Easy to use, or the simplicity, of understanding and using the method
2.2	Analyzability	Discrete	The capability to check for internal consistency or implications of the models or to identify aspects that are ambiguous. This capability is usually supported by automated tools.
2.3	Complexity management (abstraction)	Average	Ability to deal with different levels of abstractions (i.e. various levels of detail).
2.3.1	Abstraction inside each phase	Discrete	Each phase of the methodology should support multiple layers of abstractions to manage complexity
2.3.2	Existence of design primitives and high level abstraction mechanisms	Discrete	Existence of design primitive and also high level abstractions.
2.4	Executability (and testability)	Discrete	Capability of performing a simulation or generating a prototype of some aspect of

An Agent Based Approach Towards Autonomic Services

Chapter 6

			the specification. These would demonstrate the behaviours of the system and help developers determine where the intended requirements are expressed.
2.5	Expressiveness (and applicability to multiple domains)	Discrete	The capability of presenting the system concepts that refers to: <ul style="list-style-type: none"> - structure of the system - knowledge encapsulated in the system - system's ontology - data flow - control flow - concurrent activities - resource constraints (e.g. CPU, memory) - system physical architecture - agent mobility - interactions with external systems - user interface definitions
2.6	Modularity	Average	The ability to specify the system in an iterative and incremental manner. That is, when new requirements are added is should not affect the existing specifications, but may use them
2.6.1	Decomposition	Discrete	Model should support the partitioning of the problem into smaller and more manageable parts.
2.6.2	Model's dependence	Discrete	Dependency or coupling between different models. A high dependency on some specific models in the modelling method may imply they are not well designed.
2.7	Communications support	Average	Ability to communicate the models to others.
2.7.1	Clear and precise models	Discrete	Models must be precise and unambiguous.
2.7.2	Systematic transition	Discrete	A good modelling method should provide guidelines for simple transitions between models.
2.8	Refinement	Discrete	A modelling technique permits refinement of goals into sub-goals, or roles into sub-roles.
2.9	Traceability	Discrete	Traceability across refinement boundaries is provided.
3	Process	Average	Concerned with the process development aspects of the methodology.
3.1	Supporting development context	Discrete	Specifies whether a methodology is useful in creating new software, re-engineering existing software, prototyping, or design for or with reuse of components.
3.2	Lifecycle coverage	Average	How well does the methodology cover the entire lifecycle development? Methodology that cover all aspects are more likely to be chosen because of the completeness and consistency provided.
3.2.1	Requirements	Discrete	Specify the necessities from the system
3.2.2	Analysis	Discrete	Outwardly observable characteristics (e.g. functionality)
3.2.3	Implementation	Discrete	Convert design to software executables
3.2.4	Testing	Discrete	Ensure deliverables conform to specifications
3.3	Estimating and quality assurance guidelines	Discrete	Methodology covers process guidelines for software quality assurance.

An Agent Based Approach Towards Autonomic Services

Chapter 6

3.4	Architecture design	Discrete	The methodology provides mechanisms to facilitate design by using patterns and modules.
3.5	Implementation tools	Discrete	The methodology provides suggestions on how to implement software in the system.
3.6	Deployment	Discrete	The methodology provides support for practical deployment of the software.
4	Pragmatics	Average	Pragmatics refers to the practical aspects of the methodology.
4.1	Tools support	Discrete	What resources are available to support the methodology? For example, a textbook or website. Are automated tools (CASE tools) available (e.g. graphical editors, code generators, checkers)?
4.2	Pre-requisite Knowledge	Discrete	What is the required background of those learning the methodology? Does the methodology assume a level of mathematical sophistication?
4.3	Modelling suitability	Discrete	Is methodology targeted at a particular implementation language or platform? For example, a methodology may be limited to BDI based applications. Are paradigms/architectures suitable in general cases?
4.4	Domain applicability	Discrete	Is the use of the methodology suitable for a particular application domain (e.g. real-time)? Does the methodology adhere to the intended problem domain? Methodology that is applicable to a wide range of software domains has higher weighting (preferred).
4.5	Scalability	Discrete	Can the methodology be used to handle various application sizes? For example, can it provide a lightweight version for simpler problems?
4.6	Supporting management decisions	Discrete	Methodology contains guidelines for management decision making.
4.7	Cost estimation	Discrete	Methodology provides guidelines for development costing estimates.

6.2 Metrics Based Evaluation

We applied the evaluation framework to the GOFASS methodology (see Table 6.2). For comparison, we also applied the same evaluation framework to an object-oriented (OO) methodology (UML + RUP).

Table 6.2. Metrics Evaluation Results

EVALUATION FRAMEWORK (MULTIAGENT AUTONOMIC SERVICES)						
Attributes Tree		Evaluation		Evaluation Type	Normalized Values	
		GOFASS	OO		GOFASS	OO
1	Concepts and Properties			Average	9.83	4.27
1.1	Autonomy	5	0	Discrete	10	0
1.2	Reactiveness	5	5	Discrete	10	10
1.3	Proactiveness	5	0	Discrete	10	0
1.4	Mental notions	5	0	Average	10	0
1.4.1	Belief	5	0	Discrete	10	0
1.4.2	Goal/Desires	5	0	Discrete	10	0
1.4.3	Action/Intentions	5	0	Discrete	10	0
1.5	Interactions			Average	10	8
1.5.1	Sociality			Average	10	4
1.5.1.1	Organisational relationships among agents	5	0	Discrete	10	0
1.5.1.2	Types of agent interactions	5	0	Discrete	10	0
1.5.1.3	Collaboration	5	3	Discrete	10	6
1.5.1.4	Communications	5	4	Discrete	10	8
1.5.2	Interactions with the environment	5	5	Discrete	10	10
1.5.3	Concurrency	5	3	Discrete	10	6
1.5.4	Multiple control / interests	5	4	Discrete	10	8
1.5.5	Subsystems interaction	5	5	Discrete	10	10
1.6	Autonomicity			Average	10	6
1.6.1	MAPE loop			Average	10	6
1.6.1.1	Monitor/Sensing	5	3	Discrete	10	6
1.6.1.2	Analysis	5	3	Discrete	10	6
1.6.1.3	Planning	5	3	Discrete	10	6
1.6.1.4	Effectors/Actuators	5	3	Discrete	10	6
1.6.1.5	Knowledge	5	3	Discrete	10	6

An Agent Based Approach Towards Autonomic Services

Chapter 6

1.6.2	Adaptive	5	3	Discrete	10	6
1.6.3	Aware	5	3	Discrete	10	6
1.6.4	Automatic	5	3	Discrete	10	6
1.7	Service orientation			Average	9	5
1.7.1	Composibility	5	2	Discrete	10	4
1.7.2	Orchestration and choreography	5	2	Discrete	10	4
1.7.3	Virtualization	3	3	Discrete	6	6
1.8	Business process support			Average	10	6
1.8.1	Event notification	5	3	Discrete	10	6
1.8.2	Long-lived transactions	5	3	Discrete	10	6
1.8.3	Trust relationships	5	3	Discrete	10	6
2	Notation and Modeling Techniques			Average	7.44	7.11
2.1	Accessibility	3	3	Discrete	6	6
2.2	Analyzability	3	3	Discrete	6	6
2.3	Complexity management (abstraction)			Average	8	8
2.3.1	Abstraction inside each phase	4	4	Discrete	8	8
2.3.2	Existence of design primitives and high level abstraction mechanisms	4	4	Discrete	8	8
2.4	Executability (and testability)	4	4	Discrete	8	8
2.5	Expressiveness (and applicability to multiple domains)	4	4	Discrete	8	8
2.6	Modularity			Average	8	6
2.6.1	Decomposition	4	3	Discrete	8	6
2.6.2	Model's dependence	4	3	Discrete	8	6
2.7	Communications support	4	3	Average	7	6
2.7.1	Clear and precise models	4	3	Discrete	8	6
2.7.2	Systematic transition	3	3	Discrete	6	6
2.8	Refinement	4	4	Discrete	8	8
2.9	Traceability	4	4	Discrete	8	8
3	Process			Average	6.83	7.67
3.1	Development context	3	3	Discrete	6	6
3.2	Lifecycle coverage			Average	9	8
3.2.1	Requirements	5	4	Discrete	10	8
3.2.2	Analysis	5	4	Discrete	10	8
3.2.3	Implementation	5	5	Discrete	10	10
3.2.4	Testing	3	3	Discrete	6	6
3.3	Estimating and quality assurance guidelines	3	3	Discrete	6	6
3.4	Architecture design	3	3	Discrete	6	6
3.5	Implementation tools	3	5	Discrete	6	10
3.6	Deployment	4	5	Discrete	8	10
4	Pragmatics			Average	7.14	8.00
4.1	Tools support	3	5	Discrete	6	10
4.2	Pre-requisite Knowledge	3	4	Discrete	6	8

4.3	Modelling suitability	5	4	Discrete	10	8
4.4	Domain applicability	3	4	Discrete	6	8
4.5	Scalability	5	4	Discrete	10	8
4.6	Supporting management decisions	3	3	Discrete	6	6
4.7	Cost estimation	3	4	Discrete	6	8

6.3 Case Studies and Survey Evaluation

We will apply the autonomic services we create using FCGN agents in two case studies: i) an agent-mediated data exchange service and ii) a mobile digital concierge

6.3.1 Agent-Mediated Data Exchange (ADE)

The scenario that motivates this application is a common problem faced by mobile users when they are travelling on business but need to exchange or share data with their office colleagues using workstations on a fixed infrastructure. The data exchange and sharing service should be seamless even though the user may transition between fixed infrastructures such as a workstation on the local area network, to a smart-phone using a wireless infrastructure. In addition, the user would require assistance in managing a complex web of trust relationships when he needs to share data with close business associates as well as new potential clients.

In Figure 6.1, a business man wants to auction off a piece of used heavy construction equipment to his customers. He also allows for introductions to new clients by his long term customers. He takes photographs of the equipment with his camera phone and sends the photographs together with data sheets which his secretary has prepared for him to his group of customers. A long term customer I did not want to bid for the equipment, but he knew of someone, customer II, who would find it useful.

Customer III was interested but he wanted to include some terms and conditions with his bid. This scenario shows a typical ad-hoc communication pattern. Using current service, the interaction could only be supported with voice and wireless messaging (SMS/ MMS). It would not be possible to have the integration of photo, documentation and even video conveniently using current mobile services.

In the aforementioned scenario, another common way to share information currently is to use mobile e-mail services. Push e-mail services also allow information to reach users without waiting for them to check for new e-mails from a mobile server. However, this type of information sharing suffers from several drawbacks. When a message is sent to several recipients, the information would be copied several time over and deliver over multiple wireless interfaces to multiple devices. Other issues include receiving spam e-mail over the wireless interface.

One related issue in our scenario is the notifying of users of some event. This is typically done through the use the short message service (SMS). However, users to be reached may actually have switched to using instant messaging (IM) services, in which case, it would be faster and more efficient to use the IM services to notify them.

Business users in the digital ecosystems often have to form dynamic groups to exchange information and conduct their business. The formation of ad-hoc group with necessary security and access control over the information is too obtrusive if it requires all the users in the group to manually check permissions and allow interactions and data sharing between the group members.

Figure 6.1. Closed group bidding for product

6.3.2 Analysis, Specification and Architecture

We will examine the analysis, specification and architecture of the system. Figure 6.2. is the use case diagram for the mobile user. Mobile user has to take some photos and thus save inside the phone memory card. The user then starts the application in the phone whereby it is already created and installed. When starting the application, the data connection (3G or GPRS) is automatically activated and that user has to login with his/her username and password to check its authentication purposes with the gateway for the first time only.

Figure 6.2. Use cases for mobile user

When the web server has successfully received the images, it will send a notification like “Upload Images is successful” to the user. At the same time, it will send notification to sender agent. The sender agent will then notify a represented receiver agent about the images that have been sent or upload by the user.

Figure 6.3 is the use case diagram for the user agent. After the receiving agent detected there is something being sent to their user by the sending agent, the receiving agent will send a SMS or an email to notify their user that there is something sent by another user waiting for the user to receive. Before sending the SMS or email, the receiving agent will find out whether the receiving user had blocked the sending user agent, or did not know the sending user. If either of these preceding conditions is true, the receiving agent will block or ignore this message and will not notify their user.

And, eventually the user will get the selected items sent by another user. Then after downloading the user can feel free to logoff, if there were nothing to upload or download.

Figure 6.3. Use cases for user agent

A suitable user interface would be selected for the user based on his current connection to the service. If the user is only viewing information, the user will be given a mobile browser based interface using WML or XHTML-MP mark-up over WAP.

If the user is using a desktop computer, he would be able to access the service using a web browser. The web browser is able to act as a service client through the user of client-side JavaScript programs or through Java applets.

If the user still requires full service on a mobile device using a wireless connection, an appropriate interface such as a Java ME client for mobile phones or PDA client program will be invoked. We have considered the use of mobile agents which as user interface agents. These will be transferred to the mobile phone on demand. However, the benefits of this are marginal over more conventional approaches such as Over-The-Air (OTA).

In considering the requirements of the system, our analysis indicates that the

following autonomic characteristics must be exhibited by the system (see table below).

Table 6.3. ADE Autonomic Characteristics

Capabilities	Self-*/Autonomic Characteristics	Actor/Agent
1. Finds most optimal communication service given the current context of the user (for example WiFi, 3G, GPRS) 2. Switches the APN or Access Point connection to most optimal communication service. 3. Stores the new configuration	Self-optimization	User Agent
4. Select suitable service agent used depending on available communication service. 5. Select suitable service agent used to avoid congestion on mobile proxy servers.	Self-healing	Service Management Agent
6. Increase defensive condition (DEFCON) level after detected multiple intrusion attempts. 7. Prevent instantiation of new services at highest DEFCON level.	Self-protection	Security Agent
8. Archive infrequently accessed information. 9. Archive infrequently accessed accounts. 10. Users need to re-activate accounts to move out of archival state.	Self-management	Archive Agent
11. Notify user of using most suitable service of new uploaded information. 12. Select notification service based on input from Presence servers, Personal preference information, and Calendar information.	Self-optimization	User Agent Personalization Agent Notification Agent
13. Instantiate new mobile proxy agent when the mobile proxy server fails. 14. Switch mobile proxy agent service from previous server to newly replicated agent.	Self-resilience	User Agent Service Management Agent

15. Initiate new DHT reconfiguration when there is a failure of index servers causing network partitioning. 16. Automatically replicate data and DHT index to improve availability and reduce access time. 17. Use nearest replicated copy to access data. 18. Save data to nearest storage and replicate.	Self-healing Self-protection	Replication Agent User Agent Service Management Agent
---	---------------------------------	---

To illustrate the use of the GOFASS methodology, the following paragraphs will focus on deriving the autonomic behaviours for capabilities 11 and 12 from Table 6.3. The scenario is as follows: 1) the user B changes from a mobile handset to using an application on a LAN connected desktop, 2) the location of the user has to be updated, and 3) the notification agent must change the means of notification from SMS to using instant messaging (IM) or e-mail.

Goal: Notify user of new messages	
Environment:	Task:
<ul style="list-style-type: none"> Changes of connection from mobile to LAN. Different notification message transport (IM, E-mail). 	<ul style="list-style-type: none"> Determine the most suitable notification means. Send notifications to user. Wait for acknowledgement Repeat notification after a short delay if no acknowledgement is received. Notification is considered acknowledged when message recipient logs on to the ADE system.

Figure 6.4. ADE GET Card

From the GET card above, we derive the Goal Diagram. The Notify User goal is AND decomposed into two sub-goals: Confirmation and Timeliness. The Goal Diagram also shows the dependency and other relationship between goals.

Figure 6.5. ADE Goal Diagram

From the Goal Diagram, and use cases, we can identify initial agents in the ADE system. This is shown in the following figure. The User Agent tracks the preferences of the receiving user B. If the User B, indicates a “Do Not Disturb” mood with the Presence Agent, the Notification Agent should choose the E-mail service instead of Instant Messaging.

For the sake of conciseness, we will only illustrate the FCGN Diagrams for the Notification Agent. The root goal for the Notification Agent is the Notify User composite goal. Both Confirmation and Timeliness sub-goals must be satisfied before the notification goal is achieved.

Figure 6.6. ADE Agent Identification

Figure 6.7. ADE Fuzzy Cognitive Goal Net

The Timeliness goal can be broken into further sub-goals.

Figure 6.8. ADE Fuzzy Cognitive Goal Net Refinement

We have used a FCM based fuzzy transition for Choose Notify Service. We need to define the FCM model for the transition that is indicated in the next diagram. In the FCM, the location concept (within the box) is affected by the Location event (but we have not drawn the influence of the event because of the lack of space). The Do Not Disturb Mood is influenced by the Presence Mood event. The Services Available event can be decomposed to sub-events which influence the SMS, IM and Email concepts (within the box). The SMS, IM and Email concepts (within the box) control the actions of the fuzzy transition in choosing the appropriate service. The priority of the neighbouring goals is also used to influence the FCM.

Figure 6.9. ADE FCM

Similar steps are carried out for the other agents such as the User Agent, Presence Agent and Location Agent. When all the agents are specified, we can proceed to the implementation phase.

6.3.3 Implementation and Evaluation

We will examine the implementation and testing of the system. In a mobile application such as the data sharing application described in our motivating scenario, the entire system consist of a multi-agent system (MAS) working in unison. This

allows for the system to react to a dynamically changing mobile environment. Many P2P systems had been designed for a fixed infrastructure rather than a dynamic mobile one. The dynamic mobile environment where both service clients and service providers are intermittently connected and the topology of the network is changing constantly exacerbates the situation.

To overcome this, mobile peer-to-peer systems often rely on mobile proxies (see Figure 6.10) in a fixed infrastructure to operate. The proxy is able to interact with other P2P clients on behalf of the mobile client. In the case of agent based system, the mobile proxy would also act as the agent server for the user agents to continue working on behalf of the mobile user even when the mobile user is temporarily disconnected. Wireless connections are often intermittent and hence the uses of mobile proxies help to mitigate the effect of the wireless environment to the P2P system.

The peer-to-peer (P2P) architecture has been used in several large scale internet file sharing application such as Bittorrent. P2P systems do not have a single point of failure as interactions do not go through a central point. P2P systems require a reliable mechanism to discover the location of their peers and determine a communication path to their peers. Mechanisms such as DHT [Zhao et al., 2001] have been used for file-sharing systems. More complex services that need a longer description pose a challenge to large-scale service discovery.

Another approach is to use a hybrid P2P architecture, where there is a control node. The control node is an administrative entity that controls the network community.

Figure 6.10. Pure P2P Architecture

Figure 6.10 shows the pure P2P architecture where all nodes are peers. Figure 6.11 shows the hybrid architecture where there is a control node. The control node is unlikely to be a bottleneck since mobile nodes (normal nodes within the oval) are resource constrained and unlikely to overload the more powerful control node which normally runs on a server in the fixed infrastructure.

Figure 6.11. Hybrid P2P Architecture

In our proposed agent mediated P2P SOA, would use mobile proxies that could also act as agent servers for a groups of mobile terminals. Each mobile proxy node is connected to the fixed infrastructure and acts as a node in the P2P network. There is a separate group (for redundancy and availability) of control nodes that administer the

main working P2P nodes.

Figure 6.12. Mobile proxy

Figure 6.13 shows the different layers in our service-oriented architecture (SOA). There are some functions which are cross-cutting concerns. These are 3 columns to the right of Figure 6.13. These are mainly related to management and monitoring functionality. In the mobile environment, the P2P overlay network management, the service discovery, advertisements and notifications are especially critical. These are considered as cross-cutting functionalities rather than being included within the infrastructure layer. This is because the agent mediated services have to be aware of the changes in the topology and events in the environment. By working in tandem with the intelligent agents, we can reduce variations in the service quality and maintain seamless services to the mobile users.

The infrastructure layer provides basic data communications and networking features, as well as the computation and data resources for executing software components used to provide the services. The core services are the basic computational services required to be performed (for example, calculations, file and database access). These can be built using object-oriented analysis and design (OOAD) techniques rather than agent based techniques.

The higher level services in our SOA are agent mediated to take advantage of the intelligent and autonomous control possible with agents. The agents in these layers

have their goals and behaviours modelled using FCGN models (which we described earlier).

Figure 6.13. SOA Layers

A prototype (see Figure 6.14) is implemented using Java ME clients on 3G mobile phones. A mobile proxy with agent server functionality is implemented using JADE [Bellifemine et al., 2000] agent toolkit and Tomcat Java web server. FCGA agents implemented with JADE are able to take advantage of their goal-autonomy to solve problems in a dynamic mobile environment. In a complex or naturally distributed problem, we can take advantage of the distributed nature and higher processing capacity of MAS. In addition, clients using Internet browsers on desktop computers are implemented to test the seamless of the data exchange services when moving between fixed infrastructure and a mobile infrastructure.

In our service-oriented architecture (SOA), all agent functionality and communications are also exposed as services. However, because we are using resource constrained mobile devices, the client web services are designed using the REST [Fielding, 2000; Khare and Taylor, 2003] style web-service rather than SOAP/WSDL.

This style of implementation suits both Java ME clients as well as browser-based web-service clients using dynamic HTML. On the browser, the web-service clients are implemented using JavaScript.

Notification of events to mobile clients is using SMS. Java ME uses the wireless messaging API (JSR 205) to allow it to control the reception of messages. The Push Registry feature in Java ME could also be used to activate a Java ME MIDlet (an application program) when a message is received on a particular port. An alternative would be to use instant messaging protocols such as Jabber [Jabber, 2007] this has been left for future work.

Figure 6.14. Mobile P2P Data Exchange Prototype

New 3GPP services such as the IP Multimedia System (IMS) [3GPP, 2007] would better support the operation of P2P services by allowing multiple channels of

communications to be simultaneously active. In addition, future mobile phones would support multiple concurrent MIDlets [JSR271, 2007] executing in the Java virtual machine (VM). Current implementations only allow a single MIDlet to be running at any time. Multiple concurrent MIDlets would allow for communication rich functions to be implemented with several interoperable MIDlets rather than one single large MIDlet. This would allow for the application to be more easily maintained.

The main functional requirements of the system; that is, secure agent mediated data exchanges between users, are enabled through a peer-to-peer network of service-oriented agents. Autonomic or self-management capabilities of the agents (see Table 6.3) must also be distributed in this multiagent system. There are several choices for the partitioning of the autonomic capabilities.

One possible choice is to use a hierarchical network of autonomic managers (see Figure 6.15). The lower level autonomic management agents are coordinated through the next higher level of autonomic managers. The lower level autonomic manager feed information to the higher level managers. In return, control actions are dispatched by the higher level managers to the lower level autonomic agents. Lowest level autonomic agents are concerned with direct interface to physical resources such as computers, network and storage resources. Higher level autonomic agents deal with virtualized resources which may be formed by the composition of more basic services. The decision making processes in the system is decentralized and distributed throughout the hierarchy of autonomic agents. However, the coordination of policy is centralized in the sense that the highest level manager is responsible for enforcing global organization wide policies and coordinating its execution through the hierarchy.

Figure 6.15. Hierarchy of Autonomic Managers

Another possible choice in partitioning the network of autonomic managers is to distribute the Monitor, Analyze, Plan, Execute (MAPE) control loop elements in the network (see Figure 6.16). In this configuration, the processing is distributed. However, the intelligent components of the system, the Analysis and Planning elements, are shared by multiple lower elements which perform more basic functions such as monitoring or effecting control actions on the environment of the system. There may be a highest level autonomic manager with the complete MAPE functional elements which is responsible for higher level decisions such as global policy level decisions.

Figure 6.16. Distributed MAPE autonomic elements

A third alternative is to use a peer-to-peer network of autonomic agents similar to the one used for the data exchanges. In this case, a separate overlay network is formed for communications between the peer autonomic agents (see Figure 6.17). This particular partitioning of the system is maximally distributed. All policy data is distributed and there is no centralized coordination. However, there may be unnecessary redundancy as portions of the functionalities of the system are duplicated in different peers in the network of autonomic agents. However, this peer-to-peer autonomic agent network shows better self-protection and self-healing properties because of the additional redundancy built into the system.

A further variation of this is the choice between pure P2P architecture (see Figure 6.10), hybrid P2P architecture (see Figure 6.11), or a structured P2P network (see Figure 6.18).

Figure 6.17. P2P Network of Autonomic Agents

In the structured P2P architecture, a distributed routing algorithm ensures that any node can efficiently route a search to some peer. This implies that there is some inherent structured pattern of overlay links. For example, some structured P2P networks form logical rings or mesh. A distributed hash table (DHT) is a common algorithm used to assist in forming the structured P2P overlay network.

Table 6.4. Comparison of P2P networks

	Pure P2P	Hybrid P2P	Structure P2P
Scalable	No	Yes	Yes
Flexible	Yes	Yes	No
Robust	Yes	Yes	Yes
Manageable	Yes	Yes	Yes

Figure 6.18. Structured P2P Network (Logical Ring)

In our implementation, we chose to use structured P2P network based on DHT. The structured P2P overlay network is used to interconnect the autonomic agents. Although the actual users who perform data exchange form a dynamic environment where users constantly connect and disconnect the service providers for storage and other resources do not change at such a high rate. Hence, the structured P2P fits the requirements for the interconnection of the autonomic agents.

Simulation Results

To understand and verify the behaviour of the FCGN agents, we used the MASON [MASON, 2007] MAS simulator. MASON stands for “Multi-Agent Simulator Of Networks”. We embedded the FCGN agent library as a Jar library accessible to the simulation engine. We created several agents in the simulator and used the network representation mechanism of MASON to indicate the connectivity between the agents. In addition, we used the JXTA [JXTA, 2007] library to connect

the agents to simulated sensors and effectors over the P2P network. The screenshot below shows the simulator running. The acronyms used are: UA is a User Agent; NA is a Notification Agent; LA is a Location Agent; GA is a Grouping Agent; PA is a Presence Agent; and CoA is a Coordinating Agent.

Figure 6.19. ADE Simulation

The environment that the simulation is trying to simulate is shown in the diagram below. We have made an assumption that the location and notification agent is very dependent on network environment changes and the user mobility. In the actual prototype the simulated sensors and effectors are replaced by real ones.

Figure 6.20. ADE Simulation Model

6.3.4 Mobile Digital Concierge (MDC)

The services in this case studies are for use by tourists who visit a country. They may wish to inquire about different types of information such as places of interests, eateries, road directions, etc. The service should be able to automatically self-heal in the case of service provider failure, or when new providers come online. It should also be able to perform not only exact matches but approximate service matches as well.

The Mobile Digital Concierge (MDC) application was conceived as a way to allow foreign visitors to navigate our urban landscape with the ease and comfort and access to on-demand information. The aim of the application is to provide a personalised experience akin to having a personal tour guide while at the same time providing the extensive knowledge of maps, street directories and brochures without actually physically having to resort to these.

There are a variety of commercially available tourist-friendly applications today. These include GMaps and Virtual Earth software for mapping services. The problem with these applications is that they are often desktop-based, and provide little or no

mobile capabilities. Those that do, like Mobile GMaps, rely on external software developers to port the Map service onto handheld devices, with the result being that these have limited functionality and usability.

In addition to the lack of mobile interfaces, there is also a lack of service consolidation or composite services. There are many mobile services that are available to users wishing to seek particular types of information. However, while these services may exist in one form or another, they are usually only available to subscribers of a particular mobile service, and they are often standalone services. Users today may be able to get airline flight data, hotel information or location-specific maps just with their mobile device, but they often are required to subscribe to multiple services to access them.

To fulfil its role as a mobile tour guide, the MDC application aims to be as discreet a service as possible while being fully functional, usable and useful. From analyzing tourist habits, there are three main activities foreign visitors require and engage in during the course of their stay here. These are:

- Transportation – includes travelling by SMRT, bus, plane etc
- Accommodation – hotel lodging
- Sightseeing – visiting places of interest, dining etc.

The main requirement of the MDC application is to replace paper-based maps and street directories. To that end, the Mapping Service allows a user to ascertain his/her general location as well as search for services and/or facilities such as restaurants, SMRT stations or clinics that are located in the vicinity of the user's current location much like they could do with a normal paper map of Singapore.

In addition to these, an Advanced Search feature is also provided with the mapping service. With this functionality, a user who knows all or part of the name the

facilities that fall under the above-mentioned categories need only enter a keyword into the search field to get the location displayed on a dynamically-generated map.

The Location-Based Mapping Service is one of two major services provided by the MDC application. This service needs to perform four main functions:

- Determine a user's location
- Display a map of the user's surroundings relevant to the user's current location
- Display and provides information about nearby services and facilities
- Provide dynamic search of facilities and services in other locations

To accommodate the relatively small screens of mobile devices, the map that has been downloaded onto the user's mobile device will be further programmatically divided into small sections for easy navigation on the user's part. This means that while a relatively large map may be downloaded on request, the user only has visual access to one grid of the map at a time. This also has the added benefit of not having to open multiple connections to the database server every time the user navigates to a portion of the map that is outside the screen display, speeding up viewing time. This concept is illustrated in Figure 6.21.

The third and final function of the Location-Based Mapping Service is to provide for dynamically generated hotspots on the displayed map. Hotspots inform the user about the location and availability of commonly accessed services such as clinics and SMRT (Singapore Mass Rapid Transit) stations. Hotspots are essentially predefined locations of places that are represented as graphical icons on the physical map. The positions of hotspots translate to the real-time co-ordinates of the actual places, and when selected will be able to give the user information about that particular place and its address.

Multiple map grids offloaded onto the mobile device at one time

Figure 6.21. Map navigation on the mobile device

Figure 6.22. Example of multiple hotspot types on the map

The four different types of hotspots that will be implemented in the MDC mapping

service are:

- Restaurants/Diners
- Hospitals/Clinics
- SMRT Stations
- Places of Interest

Multiple hotspot types may be viewed simultaneously on the map, or users can filter the type of hotspots they wish to see. This allows them to concentrate on just one type of service and facility if they already know what they need but are not quite sure where to go. As with the selected Map image format, hotspots will also be stored as GIF images for easier compatibility and minimal size requirements.

Figure 6.23. Example of single hotspot type after filtering

The user services portion of the MDC application has three main aspects that make up the Information Services.

- **Hotel Service:** The Hotel Service is one aspect of the information services that MDC application provides to the users. This hotel service allows users to search for hotels in Singapore and get easy and essential information regarding the hotels of their choice. With this service, users will be assisted with a hotel guide providing users with information such as the hotel location, the number of rooms, price range and not forgetting the quality ranking of the hotel.
- **Flight Services:** The MDC application is set to provide tourist with one stop easy tourist needs and access. MDC provides for tourists easier flight accommodation service information. With these flight services incorporated, users will be able to get live airlines information from, arrival times and departures times. With this service, users will be able to have up to date flight service information to allow them to have an easy access of flight accommodation.
- **SMRT Travel Guide:** This service is targeted to allowing users to travel around Singapore with ease. SMRT being the most easily accessible transport in order to travel from places to places. This service is to give users a detailed SMRT route guide that comes fully with estimated waiting and travelling time. This service mainly aims to give its users the shortest possible route to get to their destination and along side with that, users are given an option to request for places of interest that are located surrounding place of departure or place of destination.

The MDC application must also allow foreign visitors to have access to local promotions and events such as sales and concert screenings. This ensures that foreign

visitors are given options as to what they can do and/or experience during their stay here. This means that mobile devices can become a portable advertising medium for local retailers and event organizers, and gives them an added platform for advertising. With this service, businesses can readily update their advertising regularly without depending on newspaper schedules or paper costs. As can be seen in the diagram on the next page, mobile advertisements are much more cost-effective than traditional methods of advertising, and while it is impossible to fully replace this convention, augmenting paper advertising with mobile ads can serve to reduce costs while reaching out to potential customers and tourists on a personalized medium.

6.3.5 Analysis, Specification and Architecture

We will examine the analysis, specification and architecture of the system.

Figure 6.24. Use case diagram of agent architecture in a network

In our project, we can implement three different kinds of agents namely, the Client Agent, the Middleman Agent and the Server Agent. The Client Agent will be in charge of representing the client in the network and communicating with the Middleman Agent for requests to the server. The task of the Middleman Agent is clearly determined by its name, acting as the middleman between the client and the server, transferring requests and data. The Server Agent will then be in charge of representing the server in the network and communicating with the Middleman Agent for response to the client.

Figure 6.25. Activity diagram of agent architecture

These agents will be able to help reduce the network load by minimizing computation work and executing tasks simultaneously on multiple network hosts. They will be able to adapt quickly to new environments, making the system information more dynamic and catered to each client's specific needs. Only the source needs to be updated on the new information thus providing flexibility in maintenance. The agents can also operate without an active connection between client and server, enhancing the tolerance level of the whole system to network faults. They will thus have the ability to update the client on the necessary data once the connection is being retrieved.

The MDC application provides location-based mapping to aid users in navigating their terrain. This functionality is the strongest aspect of the MDC application in terms of being a replacement tool for paper-based guides, and replicates the function of a normal map on the mobile device.

Figure 6.26 depicts the necessary functions that the location-based mapping service must fulfil. The first function that must be accomplished is the calculation of the user's current location. This in turn allows for the appropriate, location-specific map to be retrieved from the database and displayed to the user.

Figure 6.26. Use case diagram of the location-based mapping service

Figure 6.27. Activity diagram of the overall Location-Based Mapping Service

Figure 6.28. Use case diagram of the Information Service(s)

Figure 6.29. Use case diagram of the Promotions and Events service

Table 6.5. MDC Autonomic Characteristics

Capabilities	Self-*/Autonomic Characteristics	Actor/Agent
1. Finds most optimal communication service (for example WiFi, 3G, GPRS) 2. Switches the most optimal communication service. 3. Stores the new configuration	Self-optimization	User Agent Middleman Agent
4. Switches to most optimal service provider. 5. Select suitable service agent used to avoid congestion on mobile proxy servers. 6. Select suitable service agent used depending on available services matching query requirements.	Self-healing	Middleman Agent Service Agent

7. Instantiate use of new service agent for service provider when location of user changes. This depends on the current location of the user. 8. Switch to new service provider when access is faster or information quality is better or more relevant based on the present location of the user.	Self-protection	User Agent Location Context Agent Middleman Agent Service Agent
9. Promotions and events information that are relevant based on the present user location are selected for notification. 10. Promotion and events information must be filtered based on user preferences.	Self-management	Location Context Agent Promotion Agent Events Agent User Preference Agent Service Agent

Similar to the ADE case, we begin would illustrate the GOFASS methodology by selecting some autonomic behaviours. For the MDC, we shall focus on autonomic behaviours related to capabilities 9 and 10 from Table 6.5. The scenario we would examine is the case when the tourist moves from a PC in his hotel room, to the zoo where he uses a PDA based MDC application. The MDC must select suitable service providers for events and promotion based on his change in location. Simultaneously, content needs to be adjusted to different screen sizes.

Goal: Select Information Provider	
Environment:	Task:
<ul style="list-style-type: none"> Geographic location change Service Providers available change Device change 	<ul style="list-style-type: none"> Determine location Select best service providers for location Filter and prioritize information Reformat content for device

Figure 6.30. MDC GET Card

The GET card (see Figure 6.30) for the above scenario is derived and the goal diagram is drawn. In this scenario, the change in the environment is a change in

geographic location and the services available in the new location.

Figure 6.31. MDC Goal Diagram

Figure 6.32. MDC Fuzzy Cognitive Goal Net

We derive the goal diagram, agent identification diagram and also FCGN

diagram for all the agents identified. For the sake of conciseness, we have only shown one of the FCGN diagrams for the Choose Provider goal of the Middle Agent. The FCGN diagrams for User Agent, Location Agent, and Service Agents must also be defined.

Figure 6.33. MDC Agent Identification

6.3.6 Implementation and Evaluation

In this subsection, we will examine the implementation and testing of the Mobile Digital Concierge System. The MDC system has agent based components that interact together with other agents forming a multiagent system (MAS). However, several components of the MDC are implemented using non-agent based approaches. Non-agent components include the GPS location information provider, the Web Mapping Service (WMS), the Map and Hotspot database, the Information Service database, the Events database, and the Promotions database.

In the MDC system, there are multiple data sources from non-agent based components. To improve interoperability, these non-agent based components export service oriented interfaces. Agents in the system connect with these non-agent components through the service oriented interfaces. Between agents in the multiagent system, the rich semantics of agent communication language (ALC) help the different agents in the system to cooperate to fulfil the system objectives.

The partitioning of the system for deployment into different physical servers is dependant upon the infrastructure. For example, all agents related to the organization C (see Figure 6.34) for users can be placed on agent server 2 (AS2). The geographical distribution of service providers also has an impact on the distribution of agents and agent servers. The Map database and Information database are located in the same data centre; hence, their service agents are co-located on the same agent server 1 (AS1).

The MDC system is distributed to improve reliability by removing unique points of failure. It is also able to continue to function even when there is a partial failure of some service providers. To enable self-management or autonomic characteristics of the MDC system, we require a distributed autonomic system as well.

Figure 6.34. Block Diagram for the Mobile Digital Concierge

Agent capabilities that are specifically for autonomic behaviour in the MDC (see Table 6.5) must be distributed. The structure for the network of autonomic agents may be hierarchical, distributed MAPE elements, or peer-to-peer. For MDC, a structured P2P network is used to support the interconnection of the autonomic agents. Service providers and middle agents in the MDC do not show a very high rate of joining and leaving the P2P network. Hence, the structured P2P network is able to meet our requirements.

A prototype of the MDC was created. The implementation platform for the agent servers is JADE [Bellifemine et al., 2000]. The Tomcat web server was used as the web container. The JXTA [JXTA, 2007] platform was used to create the structured P2P network and provide the connectivity between autonomic agents.

One common problem is structured P2P is that the routing is not locality aware.

The logical structure of the P2P network may be incongruent with the underlying IP network. For example, two nodes located thousands of miles apart may be adjacent to each other on the logical overlay network formed by the P2P routing algorithm. This creates a problem with the MDC system because of the geographic distribution of the peers. In addition, parts of the MDC system are based on conventional client-server network architecture. For example, the information database and events database are relational databases connected through the client-server network.

Figure 6.35. System Diagram for the Mobile Digital Concierge

In the MDC, the service agents must be connected to both the conventional client-server network because they act as gateways for other agents to access the services of the databases. The service agents are connected to the service-oriented interfaces through the conventional client-server network. The JXTA P2P platform is used to interconnect the peer autonomic agents.

Load balancing increases the self-protection and self-healing of the MDC system. Load balancing can be accomplished with agent based mechanisms. Agents are used to discover where loads are at a maximum and where loads are at a minimum. The agents can then instruct the server which are lightly loaded to download some of

the jobs from the maximally loaded servers. The overall effect results in a more uniform re-distribution of computational loads in the MDC system.

Simulation Results

As in the case of ADE, we used the MASON simulator to exercise the autonomic behaviour of the FCGN Agents in the MDC case. As before, JXTA is used to connect the agents in the simulator to simulated effectors and simulated sensors. The following screenshot shows the simulator running. The acronyms used are: UA is a User Agent; MA is a Middle Agent; LA is a Location Agent; SA is a Service Agent.

Figure 6.36. MDC Simulation

The environment that the simulation is trying to simulate is shown in the diagram below. We have made an assumption that the location agent and some service agents are very dependent on network environment changes and the user mobility. In the actual prototype the simulated sensors and effectors are replaced by real ones.

Figure 6.37. MDC Simulation Model

6.3.7 Discussion

FCGN agents and GOFASS were used by 2 test groups of students. The ADE group had 5 students. The MDC group had 5 students. All the students were aged between 18 and 21 years old. There ADE group had one female member and four male members. The MDC group had 2 female members and 3 male members. The groups were formed voluntarily and are a representative mix of students from the School of Media and Info-Communications Technology, Singapore Polytechnic. All students are studying for their Diploma in IT and are in their 3rd and final year of study. The students are familiar with UML and Object-Oriented (OO) Programming. Agent-Oriented (AO) Techniques were new to them and introduced during the course of their project work.

Upon completion of the project and their formal assessment, the students were asked to give feedback about the experiences in the project with object-oriented techniques and agent-oriented techniques. Data was collected through a survey (see Appendix B) for questions.

Table 6.6. Normalized Ratings

OO-ADE	AO-ADE	OO-MDC	AO-MDC
0.576	0.568	0.488	0.672
0.456	0.520	0.344	0.512
0.472	0.640	0.400	0.592
0.640	0.616	0.560	0.616
0.464	0.608	0.576	0.848

Figure 6.38. Graph comparison of ratings

A rating of > 0.5 indicates adequacy of the approach for the completion of the project tasks. Overall, AO is given a higher rating compared to OO by the students. Examining the responses given by the students to individual questions (Q1-Q25) also indicate where there is greatest perceived advantages or disadvantages of the AO techniques.

From Figure 6.39, we can see that although AO techniques are preferred overall, there are some areas where OO and AO have not significant perceived differences over one another. Only for survey question 5 (about Non-functional requirements), are OO techniques perceived to have an advantage. This could be due to the fact that the

students did not focus on non-functional requirements (NFRs) or soft-goals in their project and hence may not have adequately identified methods of dealing with them.

Figure 6.39. Graph comparison of ratings (per question)

6.4 Comparison of Evaluations with Related Work

The evaluation process was primarily qualitative and was carried out a user-oriented viewpoint rather than using formal analysis, or using statistical results from simulations. The methodology requires feedback from practical usage by software developers in order to refine its usability and accessibility to non-agent researchers. This method of evaluation is not unique to our methodology [Dam and Winikoff, 2003, Sturm and Shehory, 2003]. It has particular merits in our case as we are developing a methodology to bridge the gap in Agent-Oriented Software Engineering (AOSE) between analysis and implementation.

Similar evaluation has been performed on other AOSE methodologies. Dam and Winikoff [Dam and Winikoff, 2003] apply the Feature-based Analysis, Survey, and Case Study types of evaluation while Sturm et al. [Sturm and Shehory, 2003] apply

Feature-based Analysis, Survey, and Structured Analysis. Sturm et al. based their work on the experience of a 15 students course in which each methodology was used by a group of 2 or 3 students. In the work of Dam and Winikoff, the survey is based on the evaluation done by the authors of the methodologies complemented by an experiment with 5 students (one per each methodology to be evaluated). The survey results from the above studies is not relevant from a statistical point of view; but it still provides interesting information on the experience of the students, and gives some indication of the difficulties novice users may encounter with AOSE and how they can be overcome.

One possible area of further work is there needs to be more empirical results gathered from implementation of various MAS. The users using our GOFASS methodology will be surveyed to evaluate our method. Another possible area of future work is to propose better mathematical evaluation models for the FCGN and FCGN MAS. These will help to improve our methodology.

6.5 Summary

FCGN agents when correctly configured and linked together can provide autonomic services in a variety of application domain. The main challenge faced is during the analysis phase, when suitable goals and necessary goals must be identified.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

This chapter summarizes the research work done and its main contributions. It also contains details for the planned future work.

7.1 Conclusions

Services play an increasingly important role in software applications today. There are increasing demands to build/compose software as a collection of services. Such service oriented software raises some challenges when designed using current object-oriented methodologies in situations where there is a need to trade-off competing goals, where there are complex business workflows, or when service execution is highly dynamic. Multi-Agent Systems (MAS) fits well into complex services, but current Agent-Oriented Software Engineering (AOSE) methodologies do not address the aforementioned challenges of services. AOSE for services that operate in open, dynamic and complex environments is a problem we have addressed in this thesis.

Related research suggests that services in complex and dynamic environments need to rely on agent-mediated functions such as trust negotiation, autonomic behaviours, service matching, dynamic composition, and goal-directed behaviours. Conventional approaches to services are relative static, require human management, and do not support dynamic composition, autonomous negotiation or autonomic behaviours. However, agent-oriented software engineering (AOSE) is relatively immature. In this thesis, we addressed the need for AOSE to bridge abstract agent

designs with concrete system implementations. We also addressed the problem of AOSE methodologies being inaccessible to current software developers. Lastly, we addressed the lack of a complete AOSE methodology to engineer autonomic services.

This thesis describes an AOSE methodology that is particularly suited for building service oriented software. It has the advantages of being accessible and pragmatic. The thesis presents a novel Goal-Oriented and FCM for Agent-mediated Autonomic Services (GOFASS) methodology used to specify agents for the agent-mediated autonomic services architecture. The GOFASS methodology provides guidance from early requirements gathering through to agent design and implementation. GOFASS provides a complete set of guidelines for each phase of the development of complex services. GOFASS also provides sets of models, diagrams and notations that fit into the Unified Modelling Language (UML) 2.0 superstructure as a UML 2.0 Profile. The advantage of leveraging on UML is that it increases the accessibility of GOFASS to current software developers through the use of a familiar notation. To bridge the gap between abstract design and concrete implementation, we implemented a framework for agent-mediated services with autonomic behaviour. In our framework, we proposed a new type of fuzzy cognitive goal net (FCGN) agent suited to autonomic services. This new type of agent shows fast fuzzy inference together with goal-directed behaviour. This thesis describes the design of the framework and how GOFASS utilizes FCGN agent capabilities to design-in and implement required autonomic behaviours.

This thesis is, to our best knowledge, the first AOSE targeting autonomic service oriented software. It describes a complete AOSE for modelling, designing and implementing service oriented software for complex services. Our case studies and prototypes have shown the proposed methodology is practical. Using GOFASS, two prototypes were created to evaluate the effectiveness of the FCGN agent framework in

real world. The results show that our AOSE methodology is well suited for the practical implementation of complex services in open dynamic environments. The contributions, observations and highlights from the research worked are described in more detail in the subsequent sub-sections.

7.1.1 GOFASS provides AOSE for services

The GOFASS methodology addresses the entire system and service development life-cycle from early requirements gathering, to specification, to architectural and detailed design, and finally to the actual implementation. In each phase, GOFASS provide guidelines for processes, models, diagrams and notations to be used to define complex services. A UML 2.0 Profile was is used for our model notations.

GOFASS uses Goal-Net to capture the required behaviour of services. Goal-Net allows for decomposition of complex behaviour into sub Goal-Nets. We propose a new agent model that is a hybrid of the Goal-Net model and fuzzy cognitive networks (FCGN). This new model exhibits goal and behaviour autonomy while retaining the fast action selection required for dynamic open situations. The GOFASS methodology uses the FCGN framework for translating the design into implementation.

The FCGN agent framework extends previous research done on the Goal-Net model through the addition of fuzzy goals and also fuzzy relationships between the goals. This leads to the ability to model goal autonomous agents with more a more compact goal-net. Fuzzy rules lead to more compact representation because of the ability to interpolate between imprecise values. This is one of the key advantages of fuzzy based systems.

7.1.2 GOFASS leads towards autonomic services

GOFASS provides a set of heuristics for mapping requirements to autonomic services and agent identification. The autonomic services are also mapped to high-variability sets of goals and possible system configurations to realize these goals.

FCGN agents use a FCM based inference mechanism. The FCM inference does not depend on symbolic manipulation. FCM inference requires simple mathematical computations. Because of the lightweight inference, FCGN are able to converge rapidly in a dynamic environment. This is definitely an advantage for autonomic system, services and applications which execute within the pervasive grid environment. Autonomic systems rely on a feedback control loop that exists between the sensors in the managed element and the autonomic manager's actions. In a control system, a time delay is the time it takes since the moment we make a change in the control signal until a reaction is seen in the output variable. As a general principle, time delays in the effect of the control variable or time delays in sensing the feedback signal both cause greater instability in the system. If the time delays are too large, there is a tendency to for the control mechanism to overcompensate and the system becomes unstable (outputs become unbounded). Fast inference and short delays help in maintaining range of stable control actions that may be executed by the autonomic manager.

7.1.3 GOFASS leads towards practical implementations

The GOFASS methodology was used to implement services for two case-study applications for mobile data exchange and for digital concierge services. The GOFASS methodology was used to identify agents and autonomic services in the two application

domains. The detailed design in both instances was practically implemented using FCGN agents.

One of the recurrent issues in large-scale systems is one of scalability. For the mobile data exchange service, this was a critical concern as there were potentially tens to hundreds of thousands of potential mobile users. In the prototype toolkit, we experimented with different techniques of increasing scalability. One of the methods that gave positive results is to use a pool of worker threads to execute the agent instances rather than spawning a separate thread for each agent.

J2ME or Java 2 Micro-edition is the java execution platform found on many mobile devices such as PDAs and smart-phones. The prototype was implemented using only a subset of the full java API and is compatible with the Java APIs supported by J2ME. Two main problems encountered were 1) the lack of floating point support on some J2ME devices and 2) the lack of XML parsing in the current APIs implemented in smart-phones. The lack of floating number support for FCMs could be overcome by using libraries for emulating floating point operations by using java integers to implement fixed point arithmetic. Alternatively, newer mobile devices which support the CLDC 1.1 J2ME configuration have support for floating point numbers built-in. A similar situation exists for XML parsing. New devices that will become available will support XML parsing with built-in java API libraries since these new devices must be able to connect to web-services. As an alternative, we can replacement of the APIs with third-party open-source XML parsers available for J2ME such as kXML2 (<http://kxml.sourceforge.net/>)

7.2 Future work

In brief, I plan to extend my research in the following areas in the future. I plan the use of GOFASS and FCGN agents to Ubiquitous Learning (U-Learning) applications. And, I plan to enhance the GOFASS methodology for the modelling and development of digital ecosystems. Further work on emergent behaviours in the MAS and additional evaluation techniques are also potential area for exploration. Each of these planned objectives will be elaborated on in the subsequent sections.

7.2.1 U-Learning Agent Application

In the 21st century, the learner faces many challenges brought about by globalization and technology advances. The 21st century learner may need to learn while being gainfully employed. There are multiple demands for the learner's attention and oftentimes there are only short pockets of time when the learner can gain access to traditional modes of instruction. But, the 21st century learner is also more participatory and has access to a large body of information resources because of the internet. Ubiquitous learning (U-learning) was developed to allow learners to learn anywhere, anytime, on any device. It was conceptualized as a hybrid of electronic learning (E-learning) and mobile learning (M-learning). However, U-learning faces several limitations. To enable effective learning for the 21st century learner, U-learning needs to address the needs for trusted peer-to-peer participation between communities of learners. The U-learning system must allow trust relationships to be formed to facilitate the sharing of information without compromising the ownership of intellectual property.

The environment for U-learning is a dynamic and open one with many

heterogeneous devices interacting with each other and with services. This would provide an ideal environment to test the effectiveness of GOFASS in developing autonomic services for U-learning. FCGN agents may then be used to implement the autonomic U-learning services.

7.2.2 GOFASS for Digital Ecosystems

Digital Ecosystem (DE) is an emerging paradigm for economic and technological innovation. It consists of a self-organizing digital infrastructure, aimed at creating a digital environment for networked organizations (or agents) supporting the cooperation, knowledge sharing and development of open and adaptive technologies and evolutionary domain knowledge rich environments. DE captures the essence of the ecological community in nature, where biological organisms form a dynamic and interrelated complex ecosystem, in analogy with economic organisms (such as business entities) or digital organisms (such as software applications). Digital species, just like biological ones, ‘create and conserve resources that humans find valuable’. In this proposal, we extend the concepts of DE to address specific concerns in a ubiquitous learning environment such as device diversity (analogous to specie diversity), multiple conflicting goals, and requirements for trust relationships and negotiations.

The DE paradigm is increasingly regarded as the main driver behind “business model innovation in the Digital Economy”. It transcends traditional rigorously defined collaborative environments, such as centralized (client-server) or distributed (peer-to-peer) models into agents-based, loosely coupled, domain-specific and demand driven interactive communities which offer cost-effective digital services and value-creating activities that attract agents to participate and benefit from it. DE technologies provide

transparent, open ICT support to the ecosystem's intelligence and knowledge based development, moving away from isolated business to business cooperation toward systemic collaboration, competition and synergy with partners, enabling virtual economic districts and enhancing their competitiveness over the global market.

Currently, there is no definite methodology for the development of services for the digital ecosystems. Digital ecosystem services are autonomous and autonomic. Multiagent systems (MAS) have been proposed as a means to implement digital ecosystems. We believe GOFASS can be enhanced to operate within the peer-to-peer service-oriented environment of digital ecosystems effectively while providing necessary heuristics and mappings for creating services with autonomic characteristics.

7.2.3 Emergent Behaviours

An emergent behaviour or emergent property can appear when a number of simple entities (agents) operate in an environment, forming more complex behaviours as a collective. Emergent behaviour is the patterns and properties in a complex system which cannot be predicted through analyzing behaviour of individual components in this complex system. The number of interactions between components of a system increases combinatorially with the number of components, thus potentially allowing for new types of behaviour to emerge. However, having merely a large number of interactions alone cannot guarantee emergent behaviour as many of these interactions are independent or cancel the effects of each other. In some cases, a large number of interactions result in a larger level of background noise which masks out any small-scale emergent behaviour.

Multi-agent simulations and analytic modelling can be used to study emergent behaviour in real systems. Many attempts to design algorithms for emergent behaviours or distributed problem solving devices have been inspired by the emergent

behaviour of social insect colonies and other animal societies [Bonabeau et. al. 1999]. Among these are (1) Ant Colony Optimization (ACO), (2) Particle Swarm Optimization (PSO), and (3) Stochastic Diffusion Search (SDS). Some of the common factors underlying these algorithms include: (1) Positive feedback (amplification) – simple behavioural "rules of thumb" that promote the creation of structures: including recruitment and reinforcement (2) Negative feedback – negative feedback counterbalances positive feedback and helps to stabilize the collective pattern: it may take the form of saturation, exhaustion, or competition, (3) Randomness – amplification of fluctuations (random walks, errors, random task-switching, and so on) is often crucial, since it enables the discovery of new solutions, and fluctuations can act as seeds from which structures nucleate and grow; and (4) Stigmergy – Stigmergy is a mechanism of spontaneous, indirect coordination between agents or actions, where the trace left in the environment by an action stimulates the performance of a subsequent action, by the same or a different agent.

Within the MAS-based Grid, there are many intelligent interacting components within the system. Certainly, organizational rules and interaction leads to positive feedback (amplification). Resource constraints lead to the negative feedback. The presence of multiple agents leads to random fluctuations within the MAS-based Grid. Underlying Stigmergy is the presence of state information stored in the environment of the Grid agents. Currently, Stigmergy-based interaction is not used in most common Grid algorithms and mechanisms. One area to pursue for emergent behaviour in MAS-based Grids is the introduction of Stigmergy into the interaction of the agents.

The characterization of emergent behaviour in a MAS-based grid is an interesting area of research but it has not been covered within the scope of this thesis. This is a very promising area of future work to be pursued for the fuzzy cognitive goal net and the GOFASS methodology.

7.2.4 Other Evaluation Techniques

Table 7.1. Other Evaluation Techniques

Characteristic to evaluate	Analytic Modelling	MAS Simulations
Execution-time scalability	□	••
Effectiveness	□	•
Accuracy/expressiveness	•	•
Robustness	□	••

In addition to the user evaluation framework and case studies presented, we can evaluate the methodology using Analytic/formal models and through the use of Multi-agent simulations. With respect to evaluating the methodology for the desired characteristics, these two techniques can have different contributions (• indicates technique contributes some information; •• indicates it contributes significantly; □ indicates little or no contribution).

Firm guarantees could be obtained if the system is modelled formally and the required macroscopic behaviour is proofed analytically. However, constructing a formal model and correctness proof of a complex distributed interacting computing system is infeasible. Wegner [Wegner 1997] proves this based on the fact that computing systems using interaction are more powerful problem solving engines than mere algorithms. In general, today's systems are much too complex to be formally modelled in the required granularity [Parunak et al. 1998]. The alternative to formal proof is to use an empirical method to verify the system behaviour. System behaviours are typically quantified with measurable variables which we define as macroscopic variables. To date, research in complex system (such as self-organising emergent systems) has mainly analysed their solutions empirically by performing a large number

of simulation experiments and measuring the macroscopic variables to obtain statistical results [De Wolf 2005, Kevrekidis 2004].

Agent based modelling and simulations can be used to study macroscopic behaviour in real systems. MAS are more suitable for generating complex behaviour than other computing infrastructure. We first have to identify the macroscopic variables to be measured for the characteristics of Execution-time scalability, Effectiveness, Accuracy/expressiveness and Robustness. A large number of simulation experiments, under varying external conditions, for varying completeness and values of agent knowledge, have to be conducted and the macroscopic variables measured to obtain statistical results.

Formal models are less suited for verification, but are still useful to unambiguously specify the macroscopic behaviour. We can use formal models to acquire more understanding on how the FCGN MAS work. A more rigorous and formal specification of the information types and how they are incorporated into or relate to other information and actions helps us to evaluate the expressiveness of the FCGN MAS created using our methodology. A more detailed analytic/formal model of the FCGN and FCGN MAS is a possible area for future work. Execution time analysis requires a timed version of the FCGN model with annotations of timing information within the model (this is similar to analysis using Timed Petri Nets versus Petri Nets). A timed version of the FCGN model is one area of possible future work. Since the system is too complex for a complete analytic model [Wegner 1997], formal models have limited contributions to make in terms of evaluating effectiveness or robustness of the system.

An Agent Based Approach Towards Autonomic Services

REFERENCES

- [3APL, 2007] 3APL (triple-a-p-l) An Abstract Agent Programming Language [Online]. Available: <http://www.cs.uu.nl/3apl/>
- [3GPP, 2007] 3GPP TS 22.228; 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Service requirements for the Internet Protocol (IP) multimedia core network subsystem; Stage 1.
- [ABLE, 2007] Agent Building and Learning Environment (ABLE). [Online]. Available: <http://www.alphaworks.ibm.com/tech/able>
- [Agarwal et al., 2003] M. Agarwal, V. Bhatt, H. Liu, V. Putty, C. Schmidt, G. Zhang, L. Zhen and M. Parashar, "AutoMate: Enabling Autonomic Grid Applications," CAIP Technical Report # TR-269.
- [AGLETS, 2004] Aglets Java mobile agent platform and library, version 2.0.2. [Online]. Available: <http://aglets.sourceforge.net/>
- [A-GLOBE, 2007] A-globe. A-globe is an agent platform designed for testing experimental scenarios. [Online]. Available: <http://agents.felk.cvut.cz/aglobe/>
- [Agre and Chapman, 1988] P. E. Agre and D. Chapman, "Pengi: An implementation of a theory of activity," *Artificial Intelligence*, 1987, pp. 268-272.
- [Alencar et al 2000] F. Alencar. Jaelson Castro. G. Cysneiros. John Mylopoulos. "From Early Requirements Modeled by the i* Technique to Later Requirements Modeled in Precise UML." In Anais do III Workshop em Engenharia de Requisitos. Rio de Janeiro, Brazil, 2000.
- [Andrews et al., 2003] T. Andrews et al., "Business Process Execution Language for Web Services. [BPEL4WS.]," May 2003. Available: [http://xml.coverpages.org/BPELv11 - May052003Final.pdf](http://xml.coverpages.org/BPELv11-May052003Final.pdf)

An Agent Based Approach Towards Autonomic Services

- [Anton, 1996] A. I. Anton, "Goal-based Requirements Analysis," in Proceedings of the *Second IEEE International Conference on Requirements Engineering*, 15-18 April 1996, pp. 136-144.
- [Anton, 1997] A. I. Anton, "Goal Identification and Refinement in the Specification of Software-Based Information Systems," Ph.D. thesis, Department of Computer Science, Georgia Institute of Technology, June 1997.
- [Arsanjani, 2004] A. Arsanjani, "Service-oriented Modeling and Architecture," IBM developerworks, November 2004. [Online]. Available: <http://www-106.ibm.com/developerworks/library/ws-soa-design1/>
- [Axelrod, 1976] R. Axelrod, *Structure of Decision: the cognitive maps of political elites*, Princeton University press, New Jersey, 1976.
- [Bass et al., 2001] L. Bass, P. Clements and R. Kazman, *Software Architecture in Practice*, Addison-Wesley Professional, 1997.
- [Bauer et al., 2001] B. Bauer, F. Bergenti, P. Massonet and J. Odell, "Agents and the UML: A Unified Notation for Agents and Multi-agent Systems?" in *Int. Workshop on Agent-Oriented Software Engineering (AOSE)*, 2001, pp. 148-150.
- [BEE-GENT, 2005] Bonding and Encapsulation Enhancement aGENT (Bee-gent). [Online]. Available: <http://www.toshiba.co.jp/rdc/beegent/index.htm>
- [Bellifemine et al., 2000] F. Bellifemine, A. Poggi, and G. Rimassa. Jade programmers guide, June 2000. Available: <http://sharon.cselt.it/projects/jade>.
- [Bertolini et al., 2003] D. Bertolini et al. (2003). "Designing Peer-to-Peer Applications: An Agent-Oriented Approach," In *Agent Technologies, Infrastructures, Tools, and Applications for E-Services: NODE 2002 Agent-Related Workshops*, Erfurt, Germany, October 7-10, 2002. Revised Papers: 92-106
- [Bigus et al., 2002] J. P. Bigus, D. A. Schlosnagle, J. R. Pilgrim, W. N. Mills III, and Y. Diao, "ABLE: A toolkit for building multiagent autonomic systems," *Artificial Intelligence*, Vol. 41, No. 3, 2002, p. 350

An Agent Based Approach Towards Autonomic Services

- [Bonabeau et. al. 1999] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Santa Fe Institute Studies on the Sciences of Complexity, Oxford University Press, USA, September 1999.
- [Bonifacio et al., 2002] M. Bonifacio, P. Bouquet, R. Ferrario, D. Ponte, "Towards a model of goal autonomous agents," in Proceedings of the *Fourth International Bi-Conference Workshop on Agent-Oriented Information Systems AOIS 2002 at AAMAS'02*, Bologna, Italy, July 16, 2002.
- [BPELWS, 2007] Business Process Execution Language for Web Services (BPELWS) version 1.1. [Online]. Available:
<http://www.ibm.com/developerworks/library/specification/ws-bpel/>
- [Bratman et al., 1988] M. E. Bratman, D. J. Isreal, and M. E. Pollack, "Plans and resource-bounded practical reasoning," *Computational Intelligence*, vol. 4(2), pp. 349-355, 1988. Available: <http://citeseer.ist.psu.edu/bratman88plans.html>
- [Bratman, 1987] M. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA, USA, 1987.
- [Braubach et al., 2004] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf, "Goal Representation for BDI Agent Systems," in Proceedings of the *Second Workshop on Programming Multiagent Systems: Languages, frameworks, techniques, and tools (ProMAS04)*, 2004.
- [Bresciani et. al., 2001] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, "Modeling Early Requirements in Tropos: A Transformation Based Approach," in *2nd International Workshop on Agent-Oriented Software Engineering (AOSE)*, 2001, pp. 151-168.
Available: <http://citeseer.ist.psu.edu/bresciani01modeling.html>
- [Breu,1991] Breu, R., "Algebraic specification techniques in object-oriented programming environments." In Lecture notes in computer science 562, New York: Springer-Verlag, 1991
- [Brooks, 1986] R. A. Brooks, "A Robust Layered Control System For A Mobile Robot," *IEEE Journal Of Robotics And Automation*, RA-2, April, 1986, pp. 14-23.

An Agent Based Approach Towards Autonomic Services

- [Brooks, 1991] R. A. Brooks, "Intelligence Without Reason," Technical Report. UMI Order Number: AIM-1293., Massachusetts Institute of Technology, 1991.
- [Busetta et al., 2000] P. Busetta, N. Howden, R. Ronnquist, and A. Hodgson. "Structuring BDI agents in functional clusters," in *ATAL '99: 6th International Workshop on Intelligent Agents VI, Agent Theories, Architectures, and Languages (ATAL)*, pages 277–289, London, UK, 2000. Springer-Verlag.
- [Busetta et al., 2000] Busetta, P., et al, "JACK Intelligent Agents – Components for Intelligent Agents in Java," Technical Report tr9901. Available: <http://www.agent-software.com>
- [Caire et al., 2001] Caire, G, et al., "Agent Oriented Analysis Using MESSAGE/UML," *Agent Oriented Software Engineering II*, M. Wooldridge, G. Wei, and P. Ciancarini (eds.), Volume 2222 of LNCS, New York: Springer, 2001, pp. 119-135
- [Castelfranchi, 1994] C. Castelfranchi, "Guarantees for Autonomy in Cognitive Agent Architecture", *ECAI Work-shop on Agent Theories, Architectures, and Languages*, 1994, pp. 56-70.
- [Castro et al., 2002] J. Castro, M. Kolp, and J. Mylopoulos, "Towards Requirements-Driven Information Systems Engineering: The TROPOS Project," *Information Systems*, Vol 27, No 6, 2002, pp. 365-389
- [Cernuzzi and Rossi, 2002] L. Cernuzzi and G. Rossi. "On the evaluation of agent oriented modeling methods," in *Proceedings of Agent Oriented Methodology Workshop*, Seattle, November 2002.
- [Cheng et al., 2003] Cheng, X.; Dantong Ouyang; Zhang, C., "A logic framework with algebraic extension," In *Proceedings of the 25th International Conference on Information Technology Interfaces 2003, ITI 2003*, pp.633-638, 16-19 June 2003.
- [Cossentino and Potts, 2002] M. Cossentino and C. Potts, "A Case Tool Supported Methodology for the Design of Multi-Agent Systems," *The 2002 International Conference on Software Engineering Research and Practice (SERP'02)*, 2002.
- [COUGAAR, 2007] Cougaar – Cognitive Agent Architecture. Cougaar is an open-source,

An Agent Based Approach Towards Autonomic Services

Java-based architecture that supports distributed, agent-based applications. [Online]. Available: <http://www.cougaar.org/>

[Dam and Winikoff, 2003] Dam, KH and Winikoff, M. "Comparing Agent-oriented Methodologies," *5th International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*, Springer LNCS 3030, 2003, pp. 78-93.

[Demarco, 1978] T. DeMarco, *Structured Analysis and System Specification*, Yourdon Press, N.Y., 1978.

[Dennett, 1987] D. C. Dennett, *The Intentional Stance*, The MIT Press, 1987.

[Dessales et al., 2007] Dessalles, J-L., Müller J.-P. & Phan D., "Emergence in multi-agent systems: conceptual and methodological issues." In: F. Amblard & D. Phan (Eds), *Agent-based modelling and simulation in the social and human sciences*. Oxford: The Bardwell-Press, 327-355, 2007.

[DeWolf and Holvoet, 2005] T. DeWolf & T. Holvoet. "Towards a Full Life-cycle Methodology for Engineering Decentralised Multi-Agent Systems," in *Proceedings of the Fourth International Workshop on Agent-Oriented Methodologies, OOPSLA, 2005*. [Online]. Available: <http://www.cs.kuleuven.be/~tomdw/publications/pdfs/2005oopslaworkshop.pdf>

[Dickerson and Kosko, 1994] J.A.Dickerson and B. Kosko, "Virtual Worlds as Fuzzy Cognitive Maps", *Presence*, 3(2):173-179, MIT Press, 1994

[Ferguson, 1992] I. A. Ferguson, "Touring Machines: Autonomous Agents with Attitudes," *Computer*, vol. 25, issue. 5, pp. 51-55, May, 1992.

[Fielding, 2000] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," (PhD Thesis), UC Irvine, Information and Computer Science, 2000.

[FIPA-OS, 2003] FIPA-OS, version 2.2.0. FIPA-OS is a component-based toolkit enabling rapid development of FIPA compliant agents. [Online]. Available: <http://fipa-os.sourceforge.net/>

An Agent Based Approach Towards Autonomic Services

- [Foster et al., 2004] Ian Foster, Nicholas R. Jennings and Carl Kesselman, "Brain Meets Brawn: Why Grid and Agents Need Each Other," in *Proceedings of 3rd Int. Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, New York, USA, 2004.
- [Fox and Patterson, 2003] Armando Fox and David Patterson, "Self-Repairing Computers," *Scientific American*, May 12, 2003.
- [Franklin and Graesser, 1997] S. Franklin and A. Graesser, (1997). "Is it an agent, or just a program?: A taxonomy for autonomous agents," in *Proceedings of Third International Workshop on Agent Theories, Architectures, and Languages*, 1997.
- [Futasugi et al., 1985] Futasugi, K., Goguen, J. A., Kouannaud, J. P., and Meseguer, J., "Principles of OBJ2." In *Proceedings 1985 ACM Symposium on Principles of Programming Languages*, NY, pp.52-66, 1985
- [Ganek and Corbi, 2003] A. G. Ganek and T. A. Corbi, "The Dawning of the Autonomic Computing Era," *IBM Systems Journal*, vol. 42, No. 1, 5-18, 2003.
- [Goguen et al., 1976] Goguen, J., Thatche, J., And Wagner, E., "An initial algebraic approach to the specification, correctness, and implementation of abstract data types." In *Research Report of IBM TJ Watson Research Center*, Armonk, NY: IBM, 1976.
- [Hayes-Roth, 1984] B. Hayes-Roth, "Bb1: an Architecture for Blackboard Systems that Control, Explain, and Learn about their Own Behavior," *Technical Report. UMI Order Number: CS-TR-84-1034.*, Stanford University, 1984.
- [Helsinger et al., 2004] A. Helsinger, M. Thome and T. Wright, "Cougaar: a scalable, distributed multi-agent architecture," *Proc. of IEEE International Conference on Systems, Man and Cybernetics 2004*, Oct, 10-13, 2004, vol. 2, pp. 1910–1917.
- [Horling and Lesser, 2005] Horling, Bryan; and Lesser, Victor. A Survey of Multi-Agent Organizational Paradigms. *The Knowledge Engineering Review*, Volume 19, Number 4, Cambridge University Press, pp. 281-316. 2005. An earlier version is available as UMass Computer Science Technical Report 04-45.

An Agent Based Approach Towards Autonomic Services

- [Horn, 2001] P. Horn, "Autonomic Computing: IBM's Perspective on the State of Information Technology," IBM Corporation, October 15, 2001.
- [Iglesias et al., 1998] Carlos Argel Iglesias, Mercedes Garrijo, José Centeno-González, "A Survey of Agent-Oriented Methodologies," in *Proceedings of Fifth International Workshop on Agent Theories, Architectures, and Language (ATAL-98)*, 1998, pp. 317-330.
- [Ingrand et al., 1992] Francois F. Ingrand, Michael P. Georgeff, Anand S. Rao, "An architecture for Real-Time Reasoning and System Control," *IEEE Expert: Intelligent Systems and Their Applications*, vol. 07, no. 6, pp. 34-44, Dec., 1992.
- [Jabber, 2007] Jabber Software Foundation. [Online] Available: <http://www.jabber.org>
- [JACK, 2007] JACK Agent Platform Software. JACK is a Java-based cross-platform environment for building, running and integrating multi-agent systems. It is built on a BDI (Beliefs/Desires/Intentions) foundation. [Online]. Available: <http://www.agent-software.com/>
- [JADE, 2007] Java Agent Development Framework (JADE), version 3.5. [Online]. Available: <http://jade.tilab.com/>
- [Jaffar et al., 1986] Jaffar, J., Lassez, J. L., And Maher, M. J., "Some issues and trend in the semantics of logic programming." In *Research Report of IBM TJ Watson Research Center*, Armonk, NY: IBM, 1986.
- [JASON, 2007] Jason – a Java-based interpreter for an extended version of AgentSpeak. [Online]. Available: <http://jason.sourceforge.net>
- [Jennings et al., 1998] N. R. Jennings, K. Sycara, and M. Wooldridge, "A Roadmap of Agent Research and Development," *Autonomous Agents and Multi-Agent Systems*, Vol 1, Issue 1, pp. 7-38, 1998.
- [Jennings, 2000] N. R. Jennings, "On Agent-Based Software Engineering," *Artificial Intelligence*, Vol. 117, No. 2, pp. 277-296, 2000.

An Agent Based Approach Towards Autonomic Services

- [Jennings, 2001] N. R. Jennings, "An Agent-Based Approach for Building Complex Software Systems," *Communications of the ACM*, Vol. 44, No. 4, April 2001.
- [JSON, 2007] JavaScript Object Notation (JSON). [Online]. Available: <http://json.org/>
- [JSR-271, 2007] JSR 271 Mobile Information Device Profile 3 (MIDP3) [Online]. Available: <http://jcp.org/en/jsr/detail?id=271>
- [JUNG, 2007] Java Universal Network/Graph Framework (JUNG), version 2.0 [Online]. Available: <http://jung.sourceforge.net/>
- [JXTA, 2007] JXTA (Juxtapose) is an open source peer-to-peer platform, version 2.5. [Online]. Available: <https://jxta.dev.java.net/>
- [Kavi, 2004] K. Kavi, "A Case for Agent-Oriented Software Engineering," [Online]. Available: <http://www.csrl.unt.edu/~kavi/Research/AOSE.pdf>
- [Kendall, 2000] Elizabeth A. Kendall, "Role Modeling for Agent System Analysis, Design, and Implementation," *IEEE Concurrency*, April-June 2000, pp. 34-41.
- [Kephart and Chess, 2003] Jeffrey O. Kephart, David M. Chess, "The Vision of Autonomic Computing," *IEEE Computer*, 36(1): 41-50 2003.
- [Kevrekidis 2004] I. G. Kevrekidis. *Equation-Free Modeling for Complex Systems*, 2004. [Online]. Available: <http://www.nae.edu/nae/NAEFOE.nsf/weblinks/JHAY-65GKQW>
- [Khare and Taylor, 2004] R. Khare and R. N. Taylor, "Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems," in *Proceedings of the 26th International Conference on Software Engineering*, Washington, DC, May 23-28, 2004, pp. 428-437.
- [Kinny and Georgeff, 1997] David Kinny, Michael P. Georgeff, "Modeling and Design of Multi-Agent Systems," in *Proceedings of the 4th International Workshop on Agent Theories, Architectures, and Language (ATAL-97)*, 1997, pp. 1-20.

An Agent Based Approach Towards Autonomic Services

- [Kinny et al., 1996] D. Kinny, M. Georgeff, and A. Rao, "A Methodology and Modeling Technique for Systems of BDI Agents," in W. Van de Velde and J. W. Perram (eds.), *Agents Breaking Away: Proceedings of the Seventh European Workshop on Modeling Autonomous Agents in a Multi-Agent World*, Lecture Notes in Artificial Intelligence, Vol. 1038, Springer-Verlag: Berlin, Germany, 1996, pp. 56-71.
- [Kolp et al., 2001] M. Kolp, P. Giorgini and J. Mylopoulos, "A Goal-Based Organizational Perspective on Multi-Agent Architectures," in Pre-proceedings of the *Eighth International Workshop on Agent Theories, Architectures, and Languages (ATAL-2001)*, 2001, pp. 146-158.
- [Kosko, 1986] B. Kosko, "Fuzzy Cognitive Maps," *Int. Journal of Man-Machine Studies*, Vol. 24, pp. 65-75, 1986.
- [Laird et al., 1987] John E. Laird, Allen Newell, and Paul S. Rosenbloom, "SOAR: An architecture for general intelligence," *Artificial Intelligence*, vol. 33(1), pp. 1--64, 1987.
- [Laird et al., 1994] J. E. Laird and P. S. Rosenbloom, "The Evolution of the Soar Cognitive Architecture," In D. M. Steier and T. M. Mitchell, (Eds.), *Mind Matters: A Tribute to Allen Newell*, pp. 1-50. Erlbaum, Mahwah, NJ, 1996. [Online]. Available: <http://citeseer.ist.psu.edu/laird94evolution.html>
- [Lamsweerde, 2001] A. V. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in Proceedings of the *5th IEEE International Symposium on Requirements Engineering*, Toronto, Canada, August 27-31, 2001, pp. 249-262.
- [Lenz et al., 1998] M Lenz, H D Burkhard, B Bartsch-Sporl and S Weiss , "Case-Based Reasoning Technology: From Foundations to Applications", *Lecture Notes in Artificial Intelligence*, vol. 1400, Springer Verlag, 1998.
- [Lin et al., 2007] C. Lin, K. M. Kavi, F. T. Sheldon, K. M. Daley and R. K. Abercrombie, "A Methodology to Evaluate Agent Oriented Software Engineering Techniques," in Proceedings of the *40th Annual Hawaii international Conference on System Sciences (HICSS 2007)*, Waikoloa, HI, Jan. 3-6, 2007, pp. 60-60.

An Agent Based Approach Towards Autonomic Services

- [Lind, 2001] Jürgen Lind, *Iterative Software Engineering for Multiagent Systems - The MASSIVE Method*, Volume 1994 of Lecture Notes in Computer Science, Springer Verlag, Heidelberg, May, 2001
- [Liu and Liu, 2002] Z. Liu and L. Liu, "An Agent and Goal-Oriented Approach for Virtual Enterprise Modelling: A Case Study", *Engineering Societies in the Agents World III: Third International Workshop (ESAW 2002)*, Madrid, Spain, Sep. 16-17, 2002.
- [Liu and Yu 2001] Lin Liu. Eric Yu. "From Requirements to Architectural Design: Using Goals and Scenarios." In the First International Workshop From Software Requirements to Architectures (STRAW 01) at ICSE 2001. Toronto, Canada, 14 May 2001.
- [Luck et al., 2004] M. Luck, R. Asri and M. d'Inverno, *Agent Based Software Development*, Artech House, 2004.
- [Macfadzean and Barber, 1995] R.H. Macfadzean and K.S. Barber, "An approach for decision making and control in geographically distributed systems," in Proc. *IEEE International Conference on Systems, Man and Cybernetics*, 22-25 Oct., 1995, vol. 4, pp. 3816-3821.
- [Macfadzean and Barber, 1996] R. H. Macfadzean and K. S. Barber, "Reasoning About Autonomy in Multi-Agents Systems," in Proc. *the International Multidisciplinary Conference of Intelligent Systems: A Semiotic Perspective*, Gaithersburg, MD, 1996, pp. 73-79.
- [Online]. Available: <http://citeseer.ist.psu.edu/166961.html>
- [MacNamee et al., 2003] B. MacNamee, S. Dobbyn, P. Cunningham and C. O'Sullivan, "Simulating Virtual Humans Across Diverse Situations," in *IVA '03*, Kloster Irsee, Germany, 2003.
- [Maes, 2003] P. Maes, "The agent network architecture (ANA)," *SIGART Bull.*, vol. 2, issue 4, pp. 115-120, 1991.

An Agent Based Approach Towards Autonomic Services

- [Maffre et al., 2001] E. Maffre, J. Tisseau and M. Parenthoën, "Virtual Agents Self-Perception in Virtual Story Telling," in *ICVS'01 Proceedings*, pp. 155-158, Springer LNCS 2197, 2001.
- [MASON, 2007] MASON Multiagent Simulation Toolkit, version 12.
[Online]. Available: <http://www.cs.gmu.edu/~eclab/projects/mason/>
- [Mellor and Johnson, 1997] S. Mellor and R. Johnson, "Why explore Object Methods, Patterns, and Architectures," *IEEE Software*, vol 14, no. 1, pp. 27-20, 1997.
- [Menzies and Richardson, 2006] T. Menzies and J. Richardson, "Making Sense of Requirements, Sooner," *Computer*, vol. 39, no. 10, pp. 112-114, Oct. 2006.
- [Miao et al., 1998] C. Miao, Q. Yang, H. Fang and A. Goh, "Fuzzy Cognitive Agents for Personalized Recommendation," in *Proc. the 3rd international Conference on Web information Systems Engineering(WISE)*, Singapore, Dec. 12-14, 2002, pp. 362-372.
- [Müller et al, 1995] Jörg P. Müller and Markus Pischel, "The Agent Architecture InteRRaP: Concept and Application," Technical Report RR-93-26, DFKI, Saarbrücken, 1993.
[Online]. Available: <http://citeseer.ist.psu.edu/muller93agent.html>
- [Newman et al., 2001] Harvey B. Newman, Iosif C. Legrand, Julian J. Bunn, "A Distributed Agent-based Architecture for Dynamic Services", in *Proc. Computing In High Energy And Nuclear Physics (CHEP 2001)*, Beijing, Sept, 2001. Available: <http://citeseer.ist.psu.edu/newman01distributed.html>
- [Norman et al., 2003] D. A. Norman, A. Ortony, and D. M. Russell, "Affect and machine design: Lessons for the development of autonomous machines," *IBM Systems Journal*, vol. 42, no. 1, pp. 38-44, 2003.
- [Nwana and Ndumu, 1999] H. Nwana and D. Ndumu, "A Perspective on Software Agents Research," *Knowledge Engineering Review*, Vol. 14, No. 2, pp. 1-18, 1999.

An Agent Based Approach Towards Autonomic Services

- [O'Malley and DeLoach, 2002] S. A. O'Malley and S. DeLoach, "Determining When to Use an Agent-Oriented Software Engineering Paradigm," in Revised Papers and invited Contributions From the *Second international Workshop on Agent-Oriented Software Engineering II*, M. Wooldridge, G. Weiß, and P. Ciancarini, Eds. Lecture Notes In Computer Science, vol. 2222. Springer-Verlag, London, 2002, pp. 188-205.
- [Odell et al., 2001] J. Odell, H. Van Dyke Parunak, and B. Bauer, "Representing Agent Interaction Protocols in UML," *Agent-Oriented Software Engineering*, Paolo Ciancarini and Michael Wooldridge (eds.), Springer-Verlag, Berlin, 2001, pp. 121–140.
- [Odell, 1999] J. Odell, "Objects and Agents: Is There Room for Both?" *Distributed Computing*, November 1999.
- [Odell, 2002] J. Odell, "Objects and Agents Compared", *Journal of Object Technology*, Vol. 1, No. 1, pp. 41-53, 2002.
- [Ohtani et al., 2000] S. Ohtani, M. Ishido, and K. Shimohara, "Multi-Agent Based Neural Network as a Dynamical Brain Model." in Proceeding of *5th International Symposium on Artificial Life and Robotics (AROB'00)*, Vol.1, pp. 200-203, 2000.
- [Ominici, 2001] A. Ominici, "SODA: Societies and Infrastructures in the Analysis and Design of Agent-Based Systems," *Agent Oriented Software Engineering*, Volume 1957 of LNCS, P. Ciancarini and M. Wooldridge (eds.), New York: Springer, 2001, pp. 185-193.
- [OSOA, 2007] Open Service Oriented Architecture Collaboration (OSOA). [Online]. Available: <http://www.osoa.org>
- [Padgham and Winikoff, 2002] Lin Padgham and Michael Winikoff, "Prometheus: A Pragmatic Methodology for Engineering Intelligent Agents", in Proceedings of the *workshop on Agent-oriented methodologies at OOPSLA 2002*.
- [Parenthoën et al., 2001] M. Parenthoën, P. Reignier, J. Tisseau, "Putting Fuzzy Cognitive Maps to Work in Virtual Worlds," in Proceedings of the *10th IEEE Conference on Fuzzy Systems*, Melbourne, Australia, Dec 2-5th, 2001, vol.1, pp. 252-255.

An Agent Based Approach Towards Autonomic Services

- [Park et al., 2000] K. Park, J. Kim and S. Park, "Goal Based Agent-Oriented Software Modeling," in Proceedings of the *7th Asia-Pacific Software Engineering Conference (APSEC'00)*, Singapore, Dec. 5-8, 2000.
- [Parunak et al. 1998] H. V. D. Parunak, R. Savit & R. L. Riolo. "Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users' Guide." In Lecture Notes in Computer Science, vol. 1534, pages 10–25, 1998. [Online]. Available: <http://www.newvectors.net/staff/parunakv>
- [Parunak et al., 1997] Parunak, H. V. "Go to the Ant: Engineering Principles from Natural Agent Systems." *Annals of Operations Research*, 75:69-101, 1997.
- [Patterson et. al., 2002] David Patterson, et al, "Recovery Oriented Computing (ROC): Motivation, Definition, Techniques, and Case Studies"; Computer Science Technical Report UCB//CSD-02-1175, U.C. Berkeley March 15, 2002
- [Pattnaik et al., 2003] Pratap Pattnaik, Kattamuri Ekanadham, and Joejon Jann, "Autonomic computing and Grid", Thomas J. Watson Research Center, Yorktown Heights, New York.
- [Petrie and Bussler, 2003] C. Petrie and C. Bussler, "Service Agents and Virtual Enterprises: A Survey," *Internet Computing*, July/August 2003, 7 4.
Available: <http://citeseer.ist.psu.edu/petrie03service.html>
- [Poellabauer et al., 2003] C. Poellabauer et al., "Service Morphing: Integrated System- and Application-Level Service Adaptation in Autonomic Systems," in Proceedings of the *5th Annual International Workshop on Active Middleware Services (AMS)*, June 2003.
Available: <http://citeseer.ist.psu.edu/poellabauer03service.html>
- [Rao and Georgeff, 1992] A. Rao and M. Georgeff, "An Abstract Architecture for Rational Agents," in Proceedings of the *Third International Conference on Principles of Knowledge Representation and Reasoning (KR'92)*. Morgan Kaufmann Publishers, 1992.

An Agent Based Approach Towards Autonomic Services

- [Rao and Georgeff, 1995] A. S. Rao and M. P. Georgeff, "BDI agents: From Theory to Practice", Tech. Rep. 56, Australian Artificial Intelligence Institute, Melbourne, Australia, 1995.
- [RETSINA, 2005] Reusable Environment for Task Structured Intelligent Network Agents (RETSINA).
[Online]. Available: http://www.ri.cmu.edu/projects/project_76.html
- [Rosenschein and Zlotkin, 1994] J. S. Rosenschein and G. Zlotkin, *Rules of Encounter: Designing Conventions for Automated Negotiation among Computers*, MIT Press, Cambridge, Massachusetts, USA, 1994.
- [Rumbaugh, 1991] J. Rumbaugh et al., *Object-Oriented Modeling and Design*, Prentice Hall, New York, 1991.
- [Sabas et al, 2002] A. Sabas, M. Badri and S. Delisle, "A Multidimensional Framework for the Evaluation of Multiagent System Methodologies," *6th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI-2002)*, 2002, pp. 211-216.
- [SCA, 2007] Service Component Architecture (SCA). [Online]. Available: <http://www.osoa.org/display/Main/Service+Component+Architecture+Home>
- [Schreiber et al., 1994] A. T. Schreiber et al., "CommonKADS: A Comprehensive Methodology for KBS Development," *IEEE Expert*, Vol. 9, No. 3, 1994, pp. 28-37.
- [SDO, 2007] Service Data Objects (SDO). [Online]. Available: <http://www.osoa.org/display/Main/Service+Data+Objects+Home>
- [Shen and Norrie, 1999] W. Shen and D. H. Norrie, "Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey," *International Journal of Knowledge and Information Systems*, Vol. 1, pp. 129-156, 1999.
- [Shen et al., 2004] Z. Shen, C.Y. Miao, R. Gay, X. Tao, "Goal Oriented Modeling for Intelligent Software Agents," in *Proceedings of 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'04)*, 2004, pp. 540-543.

An Agent Based Approach Towards Autonomic Services

- [Shen et al., 2004a] Z. Shen, C.Y. Miao, R. Gay, "Goal Autonomous Agent Architecture," in *Proc. The 28th Annual International Computer Software and Applications Conference (COMPSAC 2004)*, 2004, vol.2, pp. 45-46.
- [Shen et al., 2005] Z Shen, D. Li, C.Y. Miao, and R. Gay, "Goal-oriented Methodology for Agent System Development," in *Proceedings of 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology (IAT'05)*, 2005, pp. 95- 101
- [Shen et al., 2006] Jun Shen; Jun Yan; Yun Yang, "SwinDeW-S: extending P2P workflow systems for adaptive composite Web services," *Australian Software Engineering Conference*, 18-21 April, 2006, 9 pp.
- [Shen, 2005] Z. Shen, "Goal-Oriented Modeling for Intelligent Agents and Their Applications," PhD Thesis, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, 2005.
- [Siau and Rossi, 1998] K. Siau and M. Rossi, "Evaluation of Information Modeling Methods – A Review", in *Proc. 31 Annual Hawaii International Conference on System Science*, January, 1998, pp. 314-322.
- [Simjava2, 2002] Simjava 2, a discrete event, process oriented simulation package written in java. URL – <http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/>
- [Singh and Huhns, 2005] Munindar P. Singh and Michael N. Huhns, *Service-Oriented Computing : Semantics, Processes, Agents*, John Wiley & Sons, 2005.
- [SIP, 2007] SIP: Session Initiation Protocol; IETF RFC 3261
- [SOAP, 2007] Simple Object Access Protocol (SOAP). [Online]. Available: <http://www.w3.org/TR/soap/>
- [Sturm and Shehory, 2003] Arnon Sturm and Onn Shehory. "A Framework for Evaluating Agent-Oriented Methodologies," *Fifth International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2003)*, Melbourne, Australia, 14 July, 2003, pp. 94-109.

An Agent Based Approach Towards Autonomic Services

- [Sturm et al., 2003b] Sturm, A., Dori, D., and Shehory, O. 2003. "Single-model method for specifying multi-agent systems," in Proceedings of the *Second international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS '03)*, Melbourne, Australia, July 14 - 18, 2003, pp.121-128.
- [Sudeikat et al., 2004] Jan Sudeikat et al., "Evaluation of Agent – Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform" in: P. Giorgini and J. P. Müller and J. Odell (Hrsg.): *Agent-Oriented Software Engineering V, Fifth International Workshop AOSE 2004*, Springer Verlag, pp. 126-141
- [Sycara, 1998] K. P. Sycara, "Multi-Agent Systems", *AI Magazine*, Vol. 19, pp. 79-92, 1998.
- [Thangarajah et al., 2001] Thangarajah J, Padgham L and Harland J, "Representation and reasoning for goals in BDI agents," *Australian Computer Science Conference*, Jan 2002.
- [Tuscany, 2007] Apache Tuscany Project. Tuscany is based on specifications defined by the Open SOA Collaboration. [Online]. Available: <http://cwiki.apache.org/TUSCANY/>
- [Tveit, 2001] Tveit A. "A survey of agent-oriented software engineering," in Proc. the *First NTNU CSGS Conference*, May, 2001.
- [Tyrell, 1993] Tyrell, T., "Computational mechanisms for action selection," PhD thesis, University of Edinburgh, 1993.
- [UDDI, 2007] Universal Description, Discovery, and Integration [Online]. Available: http://uddi.org/pubs/uddi_v3.htm
- [Vere and Bickmore, 1990] Steven Vere and Timothy Bickmore, "A Basic Agent," *Computational Intelligence*, vol. 6(1), pp. 41-60, 1990.
- [Vincent et al, 2000] Vincent, Regis, Horling, Bryan, Lesser, Victor. Experiences in Simulating Multi-Agent Systems Using TAEMS. The Fourth International Conference on MultiAgent Systems (ICMAS 2000), AAAI. July 2000.

An Agent Based Approach Towards Autonomic Services

- [WAP, 2007] Wireless Application Protocol (WAP) specifications.
[Online]. Available: <http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html>
- [Wegner 1997] P. Wegner. "Why Interaction is More Powerful than Algorithms."
Communications of the ACM, vol. 40, no. 5, pp.80-91, May 1997.
- [Winikoff et al., 2001] M. Winikoff, L. Padgham, and J. Harland, "Simplifying the Development of Intelligent Agents", in Proc. the *14th Australian Joint Conference on Artificial Intelligence (AI'01)*, pp. 557-568, 2001.
- [Wood and DeLoach, 2001] M. Wood and S. A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology", *Agent-Oriented Software Engineering*, Lecture Notes in Computer Science, Vol. 1957, P. Ciancarini and M. Wooldridge, Eds., Springer Verlag, Berlin, January 2001.
- [Wood, 2000] M. F. Wood, "Multiagent Systems Engineering: A Methodology for Analysis and Design of Multiagent Systems," M.Sc. thesis, AFIT/GCS/ENG/ 00M-26, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, 2000.
- [Wooldridge and Jennings, 1995] M. Wooldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering Review*, vol. 10 (2), pp. 115-152, 1995.
- [Wooldridge et al., 2000] M. Wooldridge, N. R. Jennings and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *International Journal of Autonomous Agents and Multi-Agent Systems*, Vol. 3, No. 3, 2000, pp. 285-312.
- [WSBPEL, 2007] OASIS Web Services Business Process Execution Language Version 2.0 (WSBPEL 2.0) specifications. [Online]. Available:
<http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf>
- [WSDL, 2007] Web Services Description Language (WSDL) 1.1. [Online]. Available:
<http://www.w3.org/TR/wsdl>
- [XHTML, 2006] XHTML 1.1 MP Specification (XHTML-MP).
[Online]. Available: http://www.openmobilealliance.org/release_program/

An Agent Based Approach Towards Autonomic Services

docs/Browsing/V2_2-20061020-A/OMA-WAP-XHTMLMP-V1_1-20061020-A.pdf

[XML, 2007] Extensible Markup Language (XML). [Online]. Available:

<http://www.w3.org/TR/xml/>

[Xuan and Lesser, 2002] Xuan, Ping; Lesser, Victor. Multi-Agent Policies: From Centralized Ones to Decentralized Ones. Proceedings of the 1st International Joint Conference on Autonomous Agents and Multiagent Systems, Volume Part 3, ACM Press, pp. 1098-1105. 2002.

[Yan et al., 2006] Jun Yan; Yun Yang; G.K. Raikundalia, "SwinDeW -a p2p-based decentralized workflow management system," in Proc. *IEEE Transactions on Systems, Man and Cybernetics*, Sept. 2006, Part A, vol. 36, no. 5, pp. 922-935.

[Yu, 2001] E. Yu, "Agent Orientation as a Modelling Paradigm", *Wirtschaftsinformatik*, 43(2), pp. 123-132, April 2001.

[Zambonelli et al., 2003] F. Zambonelli, N. R. Jennings, M. J. Wooldridge, "Developing Multiagent Systems: the Gaia Methodology," *ACM Transactions on Software Engineering and Methodology*, vol. 12(3), pp. 317-370, September 2003.
Available: <http://citeseer.ist.psu.edu/zambonelli03developing.html>

[Zhao et al., 2001] B. Y. Zhao, J. D. Kubiawicz and A.D. Joseph. "Tapestry: An infrastructure for fault tolerant wide area location and routing." Technical report CSD-01-1141, University of California at Berkeley, 2001.

[Zimmerman et al., 2004] O. Zimmerman, P. Krogh and C. Gee, "Elements of Service Oriented Analysis and Design," *IBM developerworks*, June, 2004. [Online]. Available: <http://www-128.ibm.com/developerworks/library/ws-soad1/>

APPENDIX A AUTHOR'S PUBLICATIONS

Conference Papers

[Leong and Miao, 2008] Peter Leong and Chunyan Miao, "Ubiquitous Digital E-learning Ecosystem", Proceedings of *2nd IEEE Int. Conf. on Digital Ecosystems Technology (DEST 2008)*, Phitsanulok, Thailand, 26-28 Feb, 2008.

[Leong et. al., 2007] Peter Leong, Senthil Kumar, Valencia Lee, "Peer-to-peer Autonomic Location Based Services", Proceedings of the *Int. Conf. on Mobile Technology, Applications, and Systems (Mobility Conference 2007)*, Singapore, 12-14 Sep, 2007

[Leong et al., 2007b] Peter Leong, Chunyan Miao, et. al., "Agent Mediated Peer-to-Peer Mobile Service-Oriented Architecture", Proceedings of *1st IEEE Int. Conf. on Digital Ecosystems Technology (DEST 2007)*, Cairns, Australia, 21-23 Feb, 2007, pp. 414-419.

[Leong et al., 2006] Peter Leong, Chunyan Miao, Bu-Sung Lee, "Large Scale Agent Mediated Autonomic Service Orchestration in Grid Environment", Proceedings of *4th IEEE Int. Conf. on Industrial Informatics (INDIN 2006)*, Singapore, 16-18 Aug, 2006, pp. 469-473.

[Leong et al., 2006b] Peter Leong, Chunyan Miao, Bu-Sung Lee, "Agent Oriented Software Engineering for Grid Computing", Proceedings of the *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, Singapore, 17-19 May, 2006, 7 pp.

[Leong et al., 2005] Peter Leong, Chunyan Miao, Bu-Sung Lee, "Agent Mediated Autonomic Service Orchestration in Grid Environment", Proceedings of *3rd IEEE Int. Conf. on Industrial Informatics (INDIN 2005)*, Perth, Australia, 10-12 Aug, 2005, pp. 148-153.

[Leong and Miao, 2005b] Peter Leong, Chunyan Miao, "Fuzzy Cognitive Agents in Shared Virtual Worlds", Proceedings of the *2005 International Conference on Cyberworlds (CW 2005)*, Singapore, 23 - 25 November, 2005, 5 pp.

Journal Papers

[Leong et al., 2007c] Peter Leong, Chunyan Miao, Bu-Sung Lee, "Survey of Agent Oriented Software Engineering for Service-Oriented Computing ", *International Journal of Web Engineering and Technology*, to be published.

Presentations

[Leong, 2006c] Peter Leong, "Singapore: an intelligent and ubiquitous nation", presented at the International Conference on u-City 2006, Seoul, South Korea, 30 Nov, 2006

[Leong, 2006d] Peter Leong, "Singapore: updates on RFID/NFC, WSN, and Wireless Monitoring projects", presented at the Telematics and USN division of ETRI, Daejeon, South Korea, 1 Dec, 2006

APPENDIX B SURVEY QUESTIONS & DATA SAMPLES

Comparison of Object Oriented (OO) and Agent Oriented (AO) Software Development

Case Study: _____

	Criteria	OO	AO
(1)	Simplify mapping of system goals to system components.	Use Cases	Use Cases+ Goal Tree
		Score (0-4):	Score (0-4):
(2)	Cross-cutting concerns such as exception conditions, changes in system environments, system re-configurations are handled.		
		Score (0-4):	Score (0-4):
(3)	Enable business processes to smoothly extend in a cross-enterprise environment, including solutions to facilitate business coalition in a flexible and dynamic manner.		
		Score (0-4):	Score (0-4):
(4)	Supports dynamic virtual organizations where users are frequently added or removed from the systems. Services and resources are also dynamically added or removed from the virtual organization.		
		Score (0-4):	Score (0-4):
(5)	Non-functional requirements extensibility, usability, performance, security, availability and flexibility are captured adequately by the system model. Non-functional requirements are requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. (These are also referred to as softgoals in Requirements Engineering)		
		Score (0-4):	Score (0-4):

An Agent Based Approach Towards Autonomic Services**APPENDIX B**

(6)	Facilitates self-configuration during deployments (plug-and-play characteristics)		
		Score (0-4):	Score (0-4):
(7)	Facilitates software evolution when there are changes in the external environment of system.		
		Score (0-4):	Score (0-4):
(8)	Exhibits self-management or autonomic characteristics in the system (self-configuration, self-healing, self-optimization, self-protection, self-adaptation)		
		Score (0-4):	Score (0-4):
(9)	Supports software re-configuration to support high-variability goals in the system. This provides a form of self-adaptation and self-protection (autonomic characteristics).		
		Score (0-4):	Score (0-4):
(10)	Support for long-lived transactions		
		Score (0-4):	Score (0-4):
(11)	Support for long-lived work-flow or business processes		
		Score (0-4):	Score (0-4):
(12)	Loosely coupled system components which facilitate integration/interoperability.		
		Score (0-4):	Score (0-4):
(13)	Scalability of implementation (i.e. system is able to handle large number of users through the addition of additional resources. There is no system limiting)		
		Score (0-4):	Score (0-4):
(14)	Support the provision of functionalities as services through a service-oriented architecture (SOA)		
		Score (0-4):	Score (0-4):

An Agent Based Approach Towards Autonomic Services

APPENDIX B

(15)	Supports composition of services to create more complex services from individual ones. Facilitates middleware layer for orchestration / choreography of business processes.		
		Score (0-4):	Score (0-4):
(16)	Provides for execution on distributed and virtualized infrastructure and resources.		
		Score (0-4):	Score (0-4):
(17)	Support for asynchronous communications where users may be intermittently connected and disconnected (such as mobile users).		
		Score (0-4):	Score (0-4):
(18)	Support for mobility of user services (users are able to roam between different network addresses and devices and still have access to services)		
		Score (0-4):	Score (0-4):
(19)	Facilitates implementation that makes efficient use of modern hardware resources such as multi-core CPUs.		
		Score (0-4):	Score (0-4):
(20)	Facilitates the translation of software design models into working implementation.		
		Score (0-4):	Score (0-4):
(21)	Reduces the time for implementation through reuse of system components.		
		Score (0-4):	Score (0-4):
(22)	Provides software architecture, design and implementation patterns as guides for system designer and implementer.		
		Score (0-4):	Score (0-4):
(23)	Active threads of control/locus of control in system components facilitate partitioning of system (Active objects versus passive objects).		
		Score (0-4):	Score (0-4):

An Agent Based Approach Towards Autonomic Services

APPENDIX B

(24)	Facilitates the integration of legacy software components into new systems.		
		Score (0-4):	Score (0-4):
(25)	Supports interoperability and implementation between different implementation platforms and system architectures.		
		Score (0-4):	Score (0-4):
	TOTAL SCORES		

Initials: _____

Date: _____

Name: _____

MDC Data Sample**Comparison of Object Oriented (OO) and Agent Oriented (AO) Software Development**Case Study: Mobile Digital Tourist Assistant

	Criteria	OO	AO
(1)	Simplify mapping of system goals to system components.	Use Cases	Use Cases+ Goal Tree
		Score (0-4): <u>3</u>	Score (0-4): <u>2</u>
(2)	Cross-cutting concerns such as exception conditions, changes in system environments, system re-configurations are handled.		
		Score (0-4): <u>3</u>	Score (0-4): <u>2</u>
(3)	Enable business processes to smoothly extend in a cross-enterprise environment, including solutions to facilitate business coalition in a flexible and dynamic manner.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(4)	Supports dynamic virtual organizations where users are frequently added or removed from the systems. Services and resources are also dynamically added or removed from the virtual organization.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(5)	Non-functional requirements extensibility, usability, performance, security, availability and flexibility are captured adequately by the system model. Non-functional requirements are requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. (These are also referred to as softgoals in Requirements Engineering)		
		Score (0-4): <u>3</u>	Score (0-4): <u>2</u>
(6)	Facilitates self-configuration during deployments (plug-and-play characteristics)		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(7)	Facilitates software evolution when there are changes in the external environment of system.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(8)	Exhibits self-management or autonomic characteristics in the system (self-configuration, self-healing, self-optimization, self-protection, self-adaptation)		
		Score (0-4): <u>1</u>	Score (0-4): <u>3</u>
(9)	Supports software re-configuration to support high-variability goals in the system. This provides a form of self-adaptation and self-protection (autonomic characteristics).		
		Score (0-4): <u>1</u>	Score (0-4): <u>3</u>
(10)	Support for long-lived transactions		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(11)	Support for long-lived work-flow or business processes		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(12)	Loosely coupled system components which facilitate integration/interoperability.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(13)	Scalability of implementation (i.e. system is able to handle large number of users through the addition of additional resources. There is no system limiting)		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(14)	Support the provision of functionalities as services through a service-oriented architecture (SOA)		

An Agent Based Approach Towards Autonomic Services

APPENDIX B

		Score (0-4): <u>2</u>	Score (0-4): <u>2</u>
(15)	Supports composition of services to create more complex services from individual ones. Facilitates middleware layer for orchestration / choreography of business processes.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(16)	Provides for execution on distributed and virtualized infrastructure and resources.		
		Score (0-4): <u>1</u>	Score (0-4): <u>2</u>
(17)	Support for asynchronous communications where users may be intermittently connected and disconnected (such as mobile users).		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(18)	Support for mobility of user services (users are able to roam between different network addresses and devices and still have access to services)		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(19)	Facilitates implementation that makes efficient use of modern hardware resources such as multi-core CPUs.		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(20)	Facilitates the translation of software design models into working implementation.		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(21)	Reduces the time for implementation through re-use of system components.		
		Score (0-4): <u>1 2</u>	Score (0-4): <u>2</u>
(22)	Provides software architecture, design and implementation patterns as guides for system designer and implementer.		
		Score (0-4): <u>2</u>	Score (0-4): <u>2</u>
(23)	Active threads of control/locus of control in system components facilitate partitioning of system (Active objects versus passive objects).		
		Score (0-4): <u>2</u>	Score (0-4): <u>3</u>
(24)	Facilitates the integration of legacy software components into new systems.		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(25)	Supports interoperability and implementation between different implementation platforms and system architectures.		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
TOTAL SCORES		50	74

Initials: VB VALName: ValenciaDate: 15/01/07

ADE Data Sample**Comparison of Object Oriented (OO) and Agent Oriented (AO) Software Development**Case Study: Mobile Workflow and Data Exchange

	Criteria	OO	AO
(1)	Simplify mapping of system goals to system components.	Use Cases	Use Cases+ Goal Tree
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(2)	Cross-cutting concerns such as exception conditions, changes in system environments, system re-configurations are handled.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(3)	Enable business processes to smoothly extend in a cross-enterprise environment, including solutions to facilitate business coalition in a flexible and dynamic manner.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(4)	Supports dynamic virtual organizations where users are frequently added or removed from the systems. Services and resources are also dynamically added or removed from the virtual organization.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(5)	Non-functional requirements extensibility, usability, performance, security, availability and flexibility are captured adequately by the system model. Non-functional requirements are requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors. (These are also referred to as softgoals in Requirements Engineering)		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(6)	Facilitates self-configuration during deployments (plug-and-play characteristics)		
		Score (0-4): <u>4</u>	Score (0-4): <u>0</u>
(7)	Facilitates software evolution when there are changes in the external environment of system.		
		Score (0-4): <u>3</u>	Score (0-4): <u>4</u>
(8)	Exhibits self-management or autonomic characteristics in the system (self-configuration, self-healing, self-optimization, self-protection, self-adaptation)		
		Score (0-4): <u>4</u>	Score (0-4): <u>4</u>
(9)	Supports software re-configuration to support high-variability goals in the system. This provides a form of self-adaptation and self-protection (autonomic characteristics).		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(10)	Support for long-lived transactions		
		Score (0-4): <u>0</u>	Score (0-4): <u>4</u>
(11)	Support for long-lived work-flow or business processes		
		Score (0-4): <u>0</u>	Score (0-4): <u>4</u>
(12)	Loosely coupled system components which facilitate integration/interoperability.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(13)	Scalability of implementation (i.e. system is able to handle large number of users through the addition of additional resources. There is no system limiting)		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(14)	Support the provision of functionalities as services through a service-oriented architecture (SOA)		

An Agent Based Approach Towards Autonomic Services

APPENDIX B

		Score (0-4): <u>0</u>	Score (0-4): <u>4</u>
(15)	Supports composition of services to create more complex services from individual ones. Facilitates middleware layer for orchestration / choreography of business processes.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(16)	Provides for execution on distributed and virtualized infrastructure and resources.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(17)	Support for asynchronous communications where users may be intermittently connected and disconnected (such as mobile users).		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(18)	Support for mobility of user services (users are able to roam between different network addresses and devices and still have access to services)		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(19)	Facilitates implementation that makes efficient use of modern hardware resources such as multi-core CPUs.		
		Score (0-4): <u>3 0</u>	Score (0-4): <u>4</u>
(20)	Facilitates the translation of software design models into working implementation.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(21)	Reduces the time for implementation through re-use of system components.		
		Score (0-4): <u>3</u>	Score (0-4): <u>3</u>
(22)	Provides software architecture, design and implementation patterns as guides for system designer and implementer.		
		Score (0-4): <u>3</u>	Score (0-4): <u>2</u>
(23)	Active threads of control/locus of control in system components facilitate partitioning of system (Active objects versus passive objects).		
		Score (0-4): <u>0</u>	Score (0-4): <u>4</u>
(24)	Facilitates the integration of legacy software components into new systems.		
		Score (0-4): <u>2</u>	Score (0-4): <u>4</u>
(25)	Supports interoperability and implementation between different implementation platforms and system architectures.		
		Score (0-4): <u>1</u>	Score (0-4): <u>4</u>
TOTAL SCORES		<u>59</u>	<u>80</u>

Initials: Lim
 Name: Lim Beng Kiat

Date: 16/1/2007

APPENDIX C MATHEMATICAL BACKGROUND

Definitions

$N = \{ 0, 1, \dots \}$ the natural numbers

$N^+ = \{ 1, 2, \dots \}$ positive integers

$Z = \{ \dots, -1, 0, 1, \dots \}$ the integers

Glossary of Terms

Arc: A directed edge of a net which may connect a goal state to a transition or a transition to a goal state. Normally, it is represented by an arrow.

Input Arc (of a transition): An arc directed from a goal state to the transition.

Output Arc (of a transition): An arc directed from the transition to a goal state.

Arc annotation: An expression that may involve constants, variables and operators used to annotate an arc of a net.

Token: A data item associated with a state and chosen from the state's type.

Simple Token: A valueless token, normally represented by a black dot.

Enabling Tokens: The multiset of values obtained when an input arc annotation is evaluated for a particular binding to variables.

Marking (of a net): The set of the markings for all goal states of the net.

Marking of a state: A multiset of tokens 'residing in' the goal state.

Multiset: A collection of items where repetition of items is allowed.

State: A node of a net, taken from the state kind. A state is typed (has types of data associated with it).

Input State (of a transition): A goal state connected to the transition by an input arc.

Output State (of a transition): A goal state connected to the transition by an output

State Type: A non-empty set of data items associated with a state. (This set can describe an arbitrarily complex data structure.)

Operator: A symbol representing the name of a function.

Sort: A symbol representing the name of a set.

Signature/Many-sorted signature: A mathematical structure comprising a set of sorts and a set of operators.

Transition: A node of a net, taken from the transition kind.

Transition Mode: A pair comprising the transition and a mode. A transition mode is an assignment of values to the *transition's variables* that satisfies the transition condition.

Mode: A value taken from the transition's type.

Enabling (a transition): A transition is enabled with respect to a *net marking*. A transition is also enabled in a particular *transition mode*. Enabled transitions can occur (or fire). When a transition occurs, tokens are removed from its input states, and tokens are added to its output states.

Transition Condition: A Boolean expression (one that evaluates to true or false) associated with a transition.

Transition Occurrence (Transition Rule): If a transition is enabled in a mode, it may occur in that mode. On the occurrence of the transition, the following actions occur indivisibly:

1. For each input state of the transition: the enabling tokens of the input arc with respect to that mode are subtracted from the input state's marking, and
2. For each output state of the transition: the multiset of tokens of the evaluated output arc expression is added to the marking of the output state.

NOTE: A state may be both an input state and an output state of the same transition.

Transition Variables: All the variables that occur in the expressions associated with the transition. These are the transition condition, and the annotations of arcs surrounding the transition.

Type: A set.

Net Graph: A directed graph comprising a set of nodes of two different kinds, called states and transitions, and their interconnection by directed edges, called arcs, such that only states can be connected to transitions, and transitions to states, but never transitions to transitions, nor states to states.

Multisets

Basis set: The set of objects used to create a multiset.

A **multiset**, B , (also known as a bag) over a non-empty *basis* set, A , is a function

$$B : A \rightarrow N$$

which associates a multiplicity, possibly zero, with each of the basis elements. The multiplicity of $a \in A$ in B , is given by $B(a)$. A set is a special case of a multiset, where the multiplicity of each of the basis elements is either zero or one. The set of multisets over A is denoted by μA .

A multiset may be represented as a symbolic sum of basis elements scaled by their multiplicities (sometimes known as co-efficients).

$$B = \sum_{a \in A} B(a) \cdot a$$

Multiset cardinality is defined in the following way. The cardinality $|B|$ of a multiset B , is the sum of the multiplicities of each of the members of the multiset.

$$|B| = \sum_{a \in A} B(a)$$

Many Sorted Algebra

The concept of many sorted algebra, originating from the theory of abstract algebra in modern mathematics, aptly serves as a novel data model language (Goguen et al. 1976, Futasugi et al. 1985, Jaffar et al. 1986). The many sorted algebra has the features of being object-oriented, declarative, operational, well defined, and implementation-independent (Breu 1991). Contrasting with traditional algebra, which allows one data type only, many sorted algebra can handle multiple data types and possess many preferable properties.

Since algebra preserves rigid syntactic and semantic structures in mathematical forms, some computer scientists have extended it to many sorted algebra as their theoretical foundation of programming languages (Goguen et al. 1976, Futasugi et al. 1985, Jaffar et al. 1986), abstract data types, object-oriented methods (Breu 1991), system development, and others.

A typical algebraic system, also called algebra for abbreviation, consists of three components: carrier H (a family of sets of elements), signature Σ_H (a family of function symbols associated with the carrier H), and a family of axioms A . Therefore, an algebra can be denoted as a tuple $\langle H, \Sigma_H, A \rangle$. For simplicity, we might use the names of the carriers or the major function names in the signatures to stand for the algebras and simply denote Σ for Σ_H if this will not cause any confusion. For example, the algebra N of natural numbers has a simple structure $\langle N, \Sigma, A \rangle$, where the carrier N only contains the set of natural numbers $\{0, 1, 2, \dots\}$, the signature Σ could be the set of addition and multiplication functions $\{+, \times\}$, and the set of axioms A conserves the reflective, associative, distributive and commutative relations.

In addition to the natural number system, many well-known systems, such as the real number system REAL and propositional logic BOOL, are also shown as algebraic systems. A conventional algebraic system consists of only one sort (data type), e.g. the real numbers, in the set of elements. However, this is not the case for the systems in the world of computer programming languages. Almost every program simultaneously uses data types of integers, real numbers, characters, and other user-defined data types through various constructs, such as array and record. To accommodate this circumstance, the theory of many sorted algebra is developed by extending the single sorted algebra.

A many sorted Σ -algebra, H , is a tuple $\langle H_s, \Sigma, A \rangle$, where H_s is the *carrier* of the algebra, Σ is a set of operators (functions) $\{f_1, f_2, \dots\}$ called *signature*, and A is a set of *mandatory properties* that the algebra should follow. The carrier H_s is a set of sorts S . Each operator, say f , is a total function with domain $H_{s_1} \times H_{s_2} \times \dots \times H_{s_n}$ and range H_s , and denoted in a form of $f : H_{s_1} \times H_{s_2} \times \dots \times H_{s_n} \rightarrow H_s$. Thus, the carrier H_s is closed under each of the operators f_i in Σ . Mandatory properties could be axioms, constraints, equivalent equations, and logical clauses.

For every signature Σ there is a particularly important Σ -algebra called the *term algebra* for Σ . Term algebra is purely a formal one. Its carrier consists of sequences of symbols or strings, called *terms*, which are constructed using the function symbols in Σ . In other words, the carrier of the term algebra with respect to the signature Σ is a set of terms T_Σ which satisfies

- if $f \in \Sigma$ has arity 0 then the symbol f is in T_Σ .
- if $f \in \Sigma$ has arity $n > 0$ then the string of the form $f(t_1, \dots, t_n)$ is in T_Σ , whenever $t_1, \dots, t_n$ are strings in T_Σ .

Comparing the relation between Σ -algebra and its term algebra, one will find that term algebras are syntactic in nature and Σ -algebra semantics. In other words, every term could be viewed as a possible operational procedure in a sequence of functions and its corresponding meaning is the result after evaluation. This mechanism matches the common sense that different function expressions could be mapped to the same value.

Signatures

A many-sorted (or S -sorted) signature, Sig , is a pair:

$$Sig = (S, O)$$

Where

- S is a set of sorts (the **names** of sets, e.g., Z for the integers); and
- O is a set of operators (the **names** of functions) together with their *arity* in which specifies the names of the domain and co-domain of each of the operators

Boolean Signature

The term *Boolean Signature* is used to mean a many-sorted signature where one of the sorts is *Bool*

Variables

Let V be a set of sorted variables, called an *S -sorted set of variables*.

A variable in V of sort $s \in S$ would be denoted by v_s . For example, if $\text{integer} \in S$, then an integer variable would be v_{int} . V can be partitioned into sorts where V_s is denotes the set of variables of sort s (i.e. $v_a \in V_s$ iff $a = s$)

Terms built from a Signature and Variables

Terms of sort $s \in S$ and may be built from a signature $Sig = (S, O)$ and an S -sorted set of variables V . Denote the set of terms of sort $s \in S$ by $TERM(O \cup V)_s$.

Carrier: A set of a many-sorted algebra. For every sort, $s \in S$ there is a corresponding set, H_s , known as a *carrier*.

Many-sorted Algebras

A many-sorted Algebra, H , is a pair

$$H = (S_H, O_H)$$

Where

- S_H is the set of carriers, with for all $s \in S$, $H_s \neq \emptyset$
- O_H the set of corresponding functions
 $O_H : H_{s1} \times H_{s2} \dots \times H_{sn} \rightarrow H_s$