Patterns and Protocols for Agent-Oriented Software Development

Ayodele Oluyomi

Submitted in total fulfillment of the requirements of the degree of Doctor of Philosophy

November, 2006

Department of Computer Science and Software Engineering
Faculty of Engineering
The University of Melbourne
AUSTRALIA
Patterns and Protocols for Agent-Oriented Software Development

Ayodele Oluyomi

Abstract

Agent-oriented software engineering is faced with challenges that impact on the adoption of agent technology by the wider software engineering community. This is generally due to lack of adequate comprehension of the concepts of agent technology.

This thesis is based on the premise that the comprehension of the concepts of and the adoption of agent technology can be improved. Two approaches are explored: the first approach is the analysis and structuring of the interactions in multiagent systems; the second approach is sharing of experiences of what works and what does not in agent-oriented software engineering using software patterns. While analysis of interactions in multiagent systems improves the understanding of the behaviour of multiagent systems, sharing multiagent system development experience improves the understanding of the concepts of agent technology as well as the challenges that face the engineering of multiagent systems. It is therefore believed that interaction analysis and experience sharing can enhance the comprehension of agent technology and hence, the adoption of the technology by the wider community of software practitioners.

This thesis addresses the challenges facing agent-oriented software engineering by presenting a dedicated approach for developing agent interaction protocols to guide the interactions in a multiagent system; and a comprehensive framework for classifying, analyzing and describing agent-oriented patterns for the purpose of sharing multiagent systems development experiences.

Department of Computer Science and Software Engineering
Faculty of Engineering
The University of Melbourne
AUSTRALIA
Declaration

This is to certify that:

i. the thesis comprises only my original work towards the PhD except where indicated in the Preface

ii. due acknowledgement has been made in the text to all other material used; and

iii. this thesis is less than 100,000 words in length exclusive of tables, maps, bibliographies, and appendices

Ayodele Oluyomi

Preface

Work previously published by me that is referenced in this thesis includes [94, 95, 96, 93]. The work on interaction protocol [93] won the best paper award at PRIMA 2004. [94] relates to the Two Way classification scheme, [95] and [96] relate to pattern description templates.
Acknowledgements

To God, the giver of life and grace and strength and wisdom

To my supervisors, Leon Sterling and Shanika Karunasekera for all your support and advice and encouragement both towards the research and other aspects of life

To my darling wife, Adeola, for your love, thoughtfulness, support, prayers and sacrifice in so many ways in achieving this

To my parents, for your love and support in all ways over the years, for your prayers and encouragement towards realizing this

To my siblings, my family and friends for your prayers and good wishes and support at different stages

I appreciate you!
Contents

1. Introduction ............................................................................................ 1
   1.1 Motivation ......................................................................................... 4
   1.2 Aim ................................................................................................. 5
   1.3 Scope ............................................................................................... 6
   1.4 Thesis Contribution .......................................................................... 7
   1.5 Structure of Thesis ........................................................................... 8

2. Literature Survey .................................................................................... 11
   2.1 Agent Technology ............................................................................. 11
       2.1.1. Agents and Multiagent Systems ............................................... 12
       2.1.2. Agent-Oriented Software Engineering ................................... 14
       2.1.3. Issues with Agent-Oriented Software Engineering ................. 15
       2.1.4. Addressing Agent-Oriented Software Engineering Issues ...... 16
   2.2 Interaction Analysis in Multiagent Systems ....................................... 19
       2.2.1. Interactions among Agents ...................................................... 20
       2.2.2. Issues with Interactions among Agents .................................. 23
   2.3 Overview of Software Patterns ......................................................... 26
       2.3.1. Software Patterns ................................................................... 27
       2.3.2. Advantages of Software Patterns ........................................... 29
       2.3.3. Agent-Oriented Software Patterns ....................................... 30
       2.3.4. Misconceptions about Agent-Oriented Patterns .................... 31
       2.3.5. Classification of Agent-Oriented Patterns ................................ 34
       2.3.6. Description of Agent-Oriented Patterns ................................... 42
   2.4 Chapter Summary ............................................................................. 50

3. Developing Agent Interaction Protocols ................................................. 51
   3.1 Introduction ....................................................................................... 51
   3.2 Agent Interaction Protocol Structure ................................................. 54
       3.2.1. Agent Interaction Protocol Components ................................... 54
           3.2.1.1. Purpose ............................................................................ 56
           3.2.1.2. Players ............................................................................ 56
           3.2.1.3. Performatives ................................................................. 57
           3.2.1.4. Message Structure .......................................................... 58
           3.2.1.5. Message Exchange Rules ............................................... 59
           3.2.2. Organization of the Agent Interaction Protocol Components .... 59
   3.3 Agent Interaction Protocol Property Suite ......................................... 61
       3.3.1. Agent Interaction Protocol Property Suite ................................ 61
       3.3.2. Relationship between Protocol Structure and Protocol Property Suite ... 63
   3.4 Dedicated Approach for Developing Agent Interaction Protocols ....... 64
6.3.4. Variable Elements of the Template Structure................................................. 205
6.4 Two Way Classification Scheme Pattern Templates........................................... 206
  6.4.1. Template for Agent-Oriented Analysis Level Patterns................................. 207
    6.4.1.1. Agent-Oriented Analysis-Organizational Patterns.............................. 208
    6.4.1.2. Agent-Oriented Analysis-Interactional Patterns.............................. 210
  6.4.2. Template for MAS Architecture Level Patterns.......................................... 212
    6.4.2.1. MAS Architecture-Definitional Patterns...................................... 213
    6.4.2.2. MAS Architecture-Structural Patterns...................................... 215
    6.4.2.3. MAS Architecture-Interactional Patterns...................................... 217
  6.4.3. Template for Agent Internal Architecture Level Patterns.......................... 219
    6.4.3.1. Agent Internal Architecture-Structural Patterns.......................... 221
    6.4.3.2. Agent Internal Architecture-Interactional Patterns.......................... 223
    6.4.3.3. Agent Internal Architecture-Strategic Patterns.......................... 225
6.5 Examples of Pattern Descriptions with Pattern Templates.................................. 227
  6.5.1. Description of Agent-Oriented Analysis Patterns...................................... 227
    6.5.1.1. Agent-Oriented Analysis-Organizational Pattern........................ 228
    6.5.1.2. Agent-Oriented Analysis-Interactional Pattern............................ 230
  6.5.2. Description of Multiagent System Architecture Patterns.......................... 232
    6.5.2.1. Multiagent System Architecture-Definitional Pattern.................... 232
    6.5.2.2. Multiagent System Architecture-Structural Pattern...................... 234
    6.5.2.3. Multiagent System Architecture-Interactional Pattern.................... 236
  6.5.3. Description of Agent Internal Architecture Patterns.................................. 239
    6.5.3.1. Agent Internal Architecture-Structural Pattern............................ 239
    6.5.3.2. Agent Internal Architecture-Interactional Pattern.......................... 242
    6.5.3.3. Agent Internal Architecture-Strategic Pattern............................. 245
6.6 Chapter Summary............................................................................................. 248

7. Evaluation and Conclusions.................................................................................. 251
  7.1 Evaluation....................................................................................................... 251
  7.2 Conclusions..................................................................................................... 254
    7.2.1. Interaction Analysis............................................................................. 254
    7.2.2. Experience Sharing through Software Patterns.................................... 255
List of Figures

Figure 3.1: Agent Interaction Protocol Structure ................................................................. 59
Figure 3.2: Protocol Structure and Protocol Property Suite ................................................... 64
Figure 3.3: Proposed Agent Interaction Protocol Engineering Process ................................. 67
Figure 3.4: The Process and The Product ............................................................................. 69
Figure 4.1: General Phases of Software Development .......................................................... 79
Figure 4.2: ROADMAP Models ............................................................................................ 80
Figure 4.3: The Prometheus Methodology ........................................................................... 82
Figure 4.4: 7 Levels of Abstractions in Multiagent System Development ............................... 85
Figure 4.5: 4 Levels of Abstractions in Multiagent System Development ............................... 86
Figure 4.6: Levels of Abstractions of the Two Way Classification Scheme ......................... 87
Figure 4.7: Definition of Agent-Agent Relationships ............................................................ 91
Figure 4.8: Process Agent Architecture ................................................................................. 104
Figure 4.9: SCIC Agent Architecture .................................................................................... 105
Figure 5.1: The Attribute-based Analysis Process ................................................................. 129
Figure 5.2: The Structure-in-5 pattern .................................................................................. 134
Figure 5.3: The Joint Venture pattern .................................................................................... 136
Figure 5.4: The Pyramid pattern .......................................................................................... 137
Figure 5.5: Agent as Delegate Pattern .................................................................................. 142
Figure 5.6: Star-Shaped Movement Pattern ........................................................................... 144
Figure 5.7: Branching Pattern .............................................................................................. 145
Figure 5.8: Itinerary Pattern .................................................................................................. 146
Figure 5.9: The FIPA Contract Net Protocol .......................................................................... 148
Figure 5.10: The Embassy Pattern ......................................................................................... 150
Figure 5.11: The Monitor Pattern .......................................................................................... 151
Figure 5.12: The Wrapper Pattern ......................................................................................... 152
Figure 5.13: Basic Negotiating Agent Architecture ............................................................... 157
Figure 5.14: InteRRaP Agent Architecture ............................................................................ 159
Figure 5.15 Sense-and-Infer Pattern .................................................................................... 161
Figure 5.16: Ecological Recogniser Pattern .......................................................................... 162
Figure 5.17: The BDI architecture .......................................................................................... 170
Figure 5.18: The Sentinel Agent Pattern ................................................................................. 172
Figure 5.19: Structure of the Synchronizer Pattern ............................................................... 174
Figure 5.20: Collaborations in the Synchronizer Pattern ....................................................... 175
Figure 5.21: Structure of the Environment Mediated Pattern ............................................... 176
Figure 5.22: Collaborations in the Environment Mediated Pattern ....................................... 176
Figure 5.23: Structure of the Behaviour-Based Pattern ........................................................ 178
Figure 5.24: Structure of the Master-Slave Pattern ............................................................... 179
Figure 5.25: Meeting Pattern ................................................................................................ 180
Figure 6.1: The Structure-in-5 pattern .................................................................................... 230
Figure 6.2: Agent as Delegate Pattern ................................................................................... 234
Figure 6.3: The FIPA Contract Net Protocol .......................................................... 238
Figure 6.4: The InteRRaP Agent Architecture ...................................................... 242
Figure 6.5: Ecological Recogniser Pattern ............................................................ 244
Figure 6.6: The BDI Architecture ........................................................................ 246
List of Tables

Table 3.1: Elements of the Purpose Component.......................................................... 57
Table 3.2: Elements of the Players Component.......................................................... 58
Table 3.3: Elements of the Protocol Property Suite.................................................. 62
Table 3.4: PAP Purpose Component......................................................................... 70
Table 3.5: PAP Players Component......................................................................... 71
Table 3.6: PAP Protocol Property Suite component.............................................. 71
Table 4.1: The Two Way Classification Scheme Structure...................................... 77
Table 4.2: Vertical Dimension of the Two Way Classification Scheme............... 95
Table 4.3: Horizontal Dimension at the Agent-Oriented Analysis level..................... 96
Table 4.4: Horizontal Dimension at the Multiagent System Architecture level...... 97
Table 4.5: Horizontal Dimension at the Agent Internal Architecture level........... 98
Table 4.6: Horizontal Dimension at the Multiagent System Realization level........ 99
Table 4.7: The Two Way Classification Scheme..................................................... 101
Table 4.8: Listing of the Level and Category Attributes........................................ 108
Table 5.1: Level Attributes Table........................................................................... 124
Table 5.2: Category Attributes Table – Agent-Oriented Analysis Level............... 125
Table 5.3: Category Attributes Table – Multiagent System Architecture Level.... 125
Table 5.4: Category Attributes Table – Agent Internal Architecture Level........... 125
Table 5.5: Ninety Seven Existing Agent-Oriented Patterns................................. 131
Table 5.6: Classification of 28 Agent-Oriented Patterns........................................ 133
Table 5.7: Identification of Patterns’ Level Attributes........................................... 187
Table 5.8: Patterns’ Category Attributes – Agent-Oriented Analysis Level.......... 189
Table 5.9: Patterns’ Category Attributes – Multiagent System Architecture Level 190
Table 5.10: Pattern’s Category Attributes – Agent Internal Architecture Level.... 191
Table 6.1: Pattern Template Structure showing Fixed and Variable Segments.... 199
Table 6.2: Fixed Elements of the Template Structure............................................ 200
Table 6.3: Pattern Template Structure................................................................. 204
Table 6.4: Pattern Templates – Agent-Oriented Analysis level............................ 207
Table 6.5: Pattern Templates – Multiagent System Architecture level................. 212
Table 6.6: Pattern Templates – Agent Internal Architecture level....................... 220
Chapter 1

Introduction

Agent technology is a software development technology for analyzing and conceptualizing software applications as a system of one or more autonomous entities called software agents. While there are many definitions of a software agent [33, 79], a software agent can be described as a software entity that is situated in an environment and is capable of autonomous, proactive, reactive and social behaviours for achieving a set of goals [133]. A software agent encompasses the main concepts of agent technology which include autonomy, proactive behaviour, reactivity, social behaviour, goals, and environment. Other concepts of agent technology include adaptability, intelligence, learning, and intention recognition. A multiagent system (MAS) is a system of interacting software agents.

Success stories exist in the use of agent technology for developing software applications for domains such as process control [54], manufacturing [36, 99], air traffic control [78], information management [82], business process management [51], and e-commerce [15]. These domains require software systems with ability to handle tasks that involve an appreciable degree of decision making with minimal human intervention; capture the changes in the system’s usage and environment and adjust the system’s behaviour according to these captured changes; and software systems which comprise of components that interact in a fashion that is similar to real life organizations or human level interactions.

In the manufacturing industry for instance, changes in customer requirements, globalization and stakeholder expectations are driving major reforms in production planning, scheduling and control mechanisms [120]. Traditional approaches to
production planning and manufacturing organization configuration are inflexible, prone to plan fragility and increased response overheads. Software systems that support manufacturing organizations and control production processes need to respond to the need for reforms in the industry. This indicates that software systems for manufacturing organizations need to have flexible (or adaptive) planning/scheduling capabilities with decentralized control. Software systems that will achieve these objectives require attributes such as reasoning capabilities, context sensitive interactions, and decision making. Existing software technologies have some drawbacks in developing efficient software products for these types of domains. Agent technology features concepts and constructs that are well suited for developing software systems for these domains [50].

However, the adoption of agent technology by the wider software engineering community has suffered some setbacks [134]. Reasons for setbacks to the adoption of agent technology include the diversity of agent-oriented software engineering methodologies [123] and the lack of maturity of some of these various methodologies [80, 20]. It is believed that the variety of and lack of maturity of some agent-oriented software methodologies negatively impact the adequate understanding of the concepts, principles, applicability as well as engineering process demands of agent technology by the wider community of software developers. The proliferation of agent-oriented methodologies affect the comprehension of the correlation of agent technology concepts with phases of software engineering and how to transfer these concepts to programming abstractions [119]. Consequently, software developers either avoid the use of the technology [123] or they develop multiagent systems in non efficient ways resulting in further setbacks to the adoption of the technology.

This thesis is based on the premise that the comprehension of the concepts of and the adoption of agent technology can be improved by the following two approaches. The first approach is the analysis and structuring of the interactions in multiagent systems. The second approach is sharing experiences of what works and what does not in agent-oriented software engineering using software patterns.
While analysis of interactions in multiagent systems improves the understanding of the behaviour of multiagent systems, sharing multiagent system development experiences improves the understanding of the concepts of agent technology as well as the challenges that face the engineering of multiagent systems. It is therefore believed that interaction analysis and experience sharing can enhance the comprehension of agent technology and hence, the adoption of the technology by the wider community of software practitioners.

It has been recognized that autonomy and interaction are the two most critical features of an agent [92, 37, 102]. If the interactions in a multiagent system are understood, the intended behaviour of the system can be better comprehended, designed and realized. A strategy for the evolution of the system can be established by the analysis and design of interactions within the system. The relationships among different roles in an organization are determined by the types of interaction that exist among these roles. Put in another sense, an organizational model is defined by the interactions within the members of the organization. An organizational model defines an organization’s relevant actors or roles, their goals and dependencies [66].

To illustrate the importance of interaction to the structure of an organization, consider two organizational models Structure-in-5 and Pyramid [65]. These organizational models which are by the same authors are chosen because they adequately describe the relationships among the entities in an organization. A review of these models shows that their structures are similar as they both have three levels of control (see section 5.4.1.1 for further description of these models). The differences in their structure and their operations can be analyzed in terms of the interactions in these models. Whereas the interaction in Structure-in-5 can be described as collaborative, the interaction in Pyramid can be described as instructional. Therefore, if the type of interaction in Structure-in-5 is changed, it can evolve into the Pyramid model and vice versa. Hence, the behaviour of the multiagent system that should represent these organizations can be analyzed by the interactions in the organization.
Software patterns provide a universal way to communicate important experience in software development. A software pattern can be defined as a general solution to a recurring problem in context. Agent-oriented software patterns provide a platform for sharing multiagent system development experiences. Experience sharing provides a medium for the analysis and structuring of communications among software practitioners (researchers and developers) about the concepts of agent technology, communications about the behaviour of a society of agents (multiagent systems), and communications about what works and what does not work in the process of engineering multiagent systems. If there are structured and universally comprehensible communications about the concepts and behaviour of agents as well as what works in engineering multiagent systems, the potential of agent technology will be more widely appreciated and embraced. In other words, the appreciation and adoption of agent technology is significantly affected by the effectiveness of the communications about the concepts, behaviour and the multiagent systems engineering experiences among software practitioners.

1.1 Motivation

Based on the importance and central nature of interaction to the dynamics of multiagent systems, it follows that multiagent systems could be analyzed and developed by analyzing and defining the interactions within the system. Therefore, interaction analysis presents an important approach for enhancing the comprehension of multiagent systems concepts and behaviour. The principles of software patterns are universal. Hence when properly structured and written, the features of a software pattern are usually understood by a wide range of software experts both academics and industry practitioners. This way, the presentation of the concepts and principles of agent technology cuts across a range of researchers and practitioners. In other words, an agent technology concept described as a pattern can be understood by software practitioners apart from the members of the research group or organization that describes the concept.

This research is motivated by the following:
• The recognition of the impact of the peculiarities of individual domains of application on the agent interaction protocol being designed. Based on this recognition, a comprehensive analysis of the domain of application of multiagent systems is necessary for the development of appropriate agent interaction protocols.

• The need to have a structured definition for agent interaction protocols. Apart from the concepts of micro and macro protocols, nested protocols and concatenation of protocols, it is necessary to appropriately conceptualize the definition of agent interaction protocols. This will facilitate reusability as it will make it easy to customize a protocol where there is a clear structure to its definition.

• The need to reflect the notions of agent technology in agent-oriented patterns. If the notions of agent technology are appropriately reflected in agent-oriented patterns, software practitioners engaging in communications about multiagent systems using patterns will be better able to comprehend the potentials and properties of agent technology as well as the issues in engineering multiagent systems.

1.2 Aim

The aim of this thesis is to present a pragmatic approach for analyzing and structuring interactions in multiagent systems as well as for organizing and communicating agent-oriented patterns. It is believed that achieving this aim will improve the adoption of agent technology by the wider software engineering community.

This thesis addresses the following issues in order to achieve its aim:

• how can the behaviour of a multiagent system be analyzed in accordance with the characteristics and demands of the domain that the multiagent system is to be applied;
how can agent interaction protocols be developed to realize inter agent interactions that achieve the behaviour of a multiagent system in accordance with the peculiarities of the domain of application of this multiagent system;

how should agent-oriented patterns be classified according to the concepts they convey and the levels of abstractions they relate to;

how can agent-oriented patterns be analyzed to determine the level of abstraction and agent-oriented software engineering activity they relate to;

how can agent-oriented patterns be described such that the patterns effectively reflect the notions of agent technology which will facilitate the comprehension of these notions by non agent software practitioners

This thesis addresses the issues listed by presenting a dedicated approach for developing agent interaction protocols to guide the interactions in a multiagent system; and a comprehensive framework for classifying, analyzing and describing agent-oriented patterns for the purpose of sharing multiagent systems development experiences.

1.3 Scope

The direction of this thesis in achieving its aim is the design of a comprehensive process for the analysis and structuring of the interactions in multiagent systems and a comprehensive framework for sharing multiagent system development experiences.

This thesis presents an approach for defining the parties to, components of and properties of interactions in a multiagent system. However, it does not define a particular language for the actual design of agent interaction protocols. Examples of agent interaction protocol design languages which already exist and how they fit into the approach are given. This research is also not about specifying agent communication languages or implementing interactions in different programming paradigms.
For the purpose of describing, analyzing and classifying agent-oriented patterns, levels of abstractions of agent-oriented software development is designed in this work. These levels of abstractions are based on general phases of software development as well as two agent-oriented software methodologies (ROADMAP and Prometheus). Even though these levels of abstractions are relatively general, the levels of abstractions defined in this work cannot be precisely mapped to those of all the existing agent-oriented software engineering methodologies due to the variety and diversity of these methodologies.

Due to the fact that multiagent systems can currently be implemented using different programming paradigms, analysis and classification of implementation patterns in this work are not as comprehensive as the patterns belonging to higher levels of abstractions since such patterns would have been classified according to the individual programming paradigms. Agent-oriented pattern adaptation, integration and composition are also beyond the scope of this work. However, the framework for analyzing and classifying agent-oriented patterns presents a platform on which work on agent-oriented pattern adaptation, integration and composition can be built.

1.4 Thesis Contribution

This thesis makes two main contributions to the field of agent technology in the areas of inter agent interactions and agent-oriented patterns as follows:

The first main contribution, which is in the area of inter agent interactions, is a dedicated approach for analyzing multiagent system interactions and developing agent interaction protocols. This dedicated approach is based on a structured definition of agent interaction protocols, that is, a model of the components of an agent interaction protocol and the relationship among these components. This structured definition of agent interaction protocols is also defined by this thesis;
The second main contribution, which is in the area of agent-oriented patterns, is the development of a comprehensive framework for classifying, analyzing and describing agent-oriented patterns. This main contribution encompasses the following:

- A Two Way classification scheme for classifying agent-oriented patterns. Each category of the classification scheme is defined by its horizontal and vertical dimensions.

- The definition of the characteristic properties (attributes) of each level and category of the Two Way classification scheme;

- A process for analyzing agent-oriented patterns to determine which of the categories of the Two Way classification scheme they belong to, using the level and category attributes of the Two Way classification scheme;

- A template structure and templates for describing agent-oriented patterns. There are different templates for the different categories of the Two Way classification scheme.

1.5 Structure of Thesis

Chapter Two is a literature review which discusses the importance of interaction in understanding the behaviour of social organizations and the importance of experience sharing as a tool for bridging the gap between different software engineering communities. It discusses agent interaction protocols as a tool for governing the interactions among software agents in a multiagent system as well as software patterns as a universal platform of communication that is applicable even when sharing experience about unfamiliar concepts. Different agent-oriented software engineering methodologies are examined to understand how interaction analysis is applied in these methodologies and to understand the concepts of agent
technology that should be reflected in sharing multiagent systems development experiences.

Chapter Three describes the dedicated approach for developing agent interaction protocols. The analysis of the interactions in an organization is a means of understanding the characteristics of a domain of application. Interaction analysis can therefore be used to model the behaviour of a multiagent system developed for a particular domain. In the development of multiagent systems, agent interaction protocols are used to realize the mechanism of the interactions that define the behaviour of the system. This Chapter explores, and applies software engineering principles that are applicable, to the development of agent interaction protocols for multiagent systems. This Chapter presents a structured definition of an agent interaction protocol, specifying the components of a protocol and how these components are related. It then describes a process for developing agent interaction protocols to realize the components and the relationship between them such that they are customized to the domain of application in question.

Agent-oriented patterns should reflect the notions of agent technology for these patterns to realize their potential in enabling communications about the concepts of agent technology and the issues associated with engineering multiagent systems. Chapter Four sets the framework for incorporating the concepts of agent technology into the categorization and description of agent-oriented patterns by presenting an agent-oriented pattern classification scheme, the Two Way classification scheme. Two of the existing agent-oriented software engineering methodologies are analyzed in the light of the general phases of software development. This analysis forms a basis for defining the levels of abstraction of the classification scheme. The classification scheme is intended and designed to be relatively general and not locked in to a particular methodology. Also, in this Chapter, the properties of the levels and categories of the classification scheme are defined. These properties represent the attributes that are expected of the patterns that fall into these levels and categories.
With the classification framework presented in Chapter Four, a process is necessary for analyzing existing patterns in order to determine what category of the classification they belong to as well as for analyzing an engineering problem in order to determine what category of patterns to search for the right pattern. Chapter Five describes a process for analyzing agent-oriented patterns. The level and category attributes defined in Chapter Four form the basis for this analysis process. Attribute tables and how to use these attribute tables to analyze agent-oriented patterns are described. Twenty eight agent-oriented patterns, chosen from ninety seven patterns studied, are analyzed and categorized in this Chapter. This analysis process also allows for a deeper insight into the capability and applicability of some of these patterns.

The format for describing agent-oriented patterns, that is, the description template is almost as important as the content of the pattern to be described. The template provides the placeholders for the information that will be included in the pattern description, as well as the arrangement of this information. That is, what is captured by a pattern description as well as how it is captured is determined by the pattern template. In Chapter Six, an agent-oriented pattern template structure is designed based on the classification framework presented in Chapter Four. The template structure is designed to capture the notions of agent technology that is relevant at the respective levels of abstractions in agent-oriented development while avoiding redundancy. Chapter Seven presents an evaluation and the conclusion of this thesis.
Chapter 2

Literature Survey

Based on the review of existing literature, this Chapter describes the concepts, applications, benefits and engineering challenges of agent technology. This chapter focuses on the two main contributions of this thesis, showing how interaction analysis and agent-oriented patterns enhance the comprehension of agent technology. It then discusses the issues with interaction analysis and agent-oriented patterns in engineering agent based systems and how these issues affect the comprehension and therefore adoption of agent technology by the wider community of software developers.

2.1 Agent Technology

Software agent technology is a software development paradigm for conceptualizing, analyzing and developing software systems that have high level attributes. These attributes include independence, rational planning, intelligence, context level interaction, adaptability, and reactivity. This type of software systems are needed for application domains with non trivial decision making processes, application domains requiring appropriate and timely reactions to environmental changes in order to achieve particular goals, application domains featuring roles that require independence and that are capable of context level interactions. Examples of such domains include process control [54], patient monitoring in medical applications [44], e-commerce [134], and intelligent manufacturing [120]. The characteristic features of a domain of application determine the applicability of Agent technology to such a domain.
2.1.1. Agents and Multiagent Systems

There are a number of definitions for a software agent. One of the reasons for so many definitions is that different attributes of an agent are essential to different domains. Researchers tend to define a software agent to reflect the attributes emphasized by their domain of interest, for example information retrieval and electronic markets. However, there is general agreement as to the features of an agent. An adaptation of the definition of an agent according to [133] which will be adopted in this thesis, is

An agent is a computing entity, situated in an environment and is capable of autonomous, proactive, reactive, and social behaviours for achieving a set of goals.

From this definition, the concepts of agent technology include autonomy, proactive behaviour, reactivity, social behaviour, goals, and environment. Other concepts of agent technology include adaptability, intelligence, learning, and intention recognition.

The concept of the environment in the definition of an agent represents the agent’s sphere of operation and interaction. An agent’s environment is the receptor of the agent’s actions and the agent takes input from its environment. Examples of an agent’s environment include other agents, the internet, physical world, user, or any combination of these [132].

Autonomy refers to the ability of an agent to make decisions and take actions with minimal human intervention. Autonomy in an agent implies control over the agent’s actions and internal state. Autonomy is a key feature of an agent as it determines the degree to which human intervention will be required in the operations of the agent.
Social behaviour refers to the agent’s interaction with other agents in its environment. Interaction among agents is high level and context sensitive. Interaction is carried out using some Agent Communication Language (ACL) that is governed by some Agent Interaction Protocol (AIP).

Proactive behaviour refers to the ability of the agent to devise plan of actions that are directed towards achieving the goals of the agent. By this attribute the agent undertakes goal directed actions that impact on its environment rather than merely waiting react to input from the environment.

Reactive behaviour refers to the ability of the agent to react to changes in the environment. In the pursuit of the agent’s goals, it senses changes in the environment that can impact on the realization of its objectives and adequately responds to such changes in a good time.

Goal refers to the set of objectives that the agent sets out or is created to achieve. The goal of an agent might be to maximize some gains, ensure safe running of a system, or assist a human in specific tasks. The goal concept is critical to the definition of an agent as it determines what the agent plans to do, how it interacts and what it reacts to in the environment, and how it reacts to the environment.

Considering the many definitions of an agent, it is not surprising that there are different definitions of a Multiagent System. According to [53] a Multiagent System is a system that is made up of more than one autonomous component (an agent). A type of multiagent system is a software representation of an organization where the various roles for achieving the goals of the organization are modeled by software agents and the dynamics of the organization are modeled by the interactions among the agents in the system, for instance intelligent manufacturing. In this case, the multiagent system is an implementation of a virtual organization. Another type of multiagent system is a software system where software agents from different sources interact in the same environment to achieve their individual goals for instance online auction or e-commerce.
The interactions among the agents in a multiagent system determine if the system is a virtual organization or a virtual marketplace for instance. The interactions among the agents in a virtual organization will be collaboration or cooperation while the interactions among the agents in virtual marketplace will be competition.

A multiagent system could either be a closed system or an open system. The agents that participate in closed multiagent systems are all predefined and remain constant. For instance, the agents collaborating in an intelligent assembly line process interact in closed multiagent systems. The agents that make up open multiagent systems are not predefined and are from different sources. A multiagent system for on-line auction is an example of an open multiagent system.

2.1.2. Agent-Oriented Software Engineering

The research activities in agent technology demonstrate that the attributes and capabilities of software agents can be realized in software systems. The challenge therefore is to take this technology and its potentials beyond the walls of research laboratories and produce industrial scale multiagent systems that are reliable and that deliver the promises of the technology. The research community therefore embarked on developing Agent-Oriented Software Engineering (AOSE) methodologies for building multiagent systems. Some of the agent-oriented software engineering methodologies are AAII [63, 62], MAS CommonKADS [47], DESIRE [9], MaSE [131, 22], Gaia [137, 135, 138], ROADMAP [58, 57, 73], Tropos [10], ADEPT [52, 49] and Prometheus [97]. The agent-oriented software engineering methodologies were and are still being developed simultaneously by different agent research groups. These methodologies differ in a variety of ways. The methodologies differ in the stages of software development that they focus on. For instance whereas ROADMAP, Gaia and Prometheus focus on the analysis and design stages of development, Tropos and MaSE attempt to cover all the stages of implementation from analysis to implementation [20, 73].
2.1.3. Issues with Agent-Oriented Software Engineering

In an evaluation of three agent-oriented software engineering methodologies MaSE, Prometheus and Tropos by [20], it was noted that the implementation phase is not well supported by these methodologies, quality assurance, guideline estimation are not supported at all by any of these three methodologies. This evaluation also observed that the extent of “real” use of these three methodologies apart from student projects and demonstrations is not clear. The agent-based system modeling techniques Agent-Oriented Methodology (AOM)/GAIA, ADEPT and DESIRE were evaluated in [119]. This evaluation suggests that these modeling techniques will benefit from enhancements in the areas such as implementation of agent-based systems, testing, connecting agent-based models with stages of software development. These evaluations both noted that there are not many instances of large scale commercially viable multiagent systems applications. The conclusions of these evaluations are a reflection of the fact that the take-up of agent technology as a software development paradigm is minimal as noted by [80].

There are factors responsible for the slow take off of agent technology in the industry. One of these factors is lack of adequate and universal understanding of the concepts of agent technology as a result of immaturity of agent technology research [81]. The immaturity of the technology results in differences in the expression of agent technology’s concepts and terminologies which introduces differences in the meanings of particular concepts. For this reason, the knowledge of this technology is not prevalent among commercial software developers [81].

Another factor is lack of connection of research work done in conceptual modeling of multiagent systems and implementation of such systems [117]. There are different documented examples of multiagent systems development that end at the analysis and design stages and there are different examples of multiagent systems development for another set of problems that only describe the implementation of the system. There are not many documented examples of complete multiagent systems development from requirements gathering to deployed systems.
Although the development of true agent-oriented programming languages has progressed significantly [7], their use in the industry is still limited [81]. Agent-oriented programming languages with constructs that define and represent the concepts of agent technology such as autonomy, are still in the pipeline. Currently, most multiagent systems are implemented by other programming paradigms such as Object Oriented (OO) programming. This therefore requires translating the multiagent systems design into some OO design. As a result, some software practitioners argue that translating multiagent systems design to OO design is inefficient as OO is not suited for implementing the full capabilities of agent technology [131, 75]. In some other cases, the actual gains of using agent technology to solve a particular problem may be lost in the process of translating to other programming paradigms.

Another factor is lack of adequate understanding of problem domains to which agent technology is applicable [134]. As a result, multiagent systems are used to build software systems where there are no justifications or no gains for using the technology.

These factors are responsible for at least the following two consequences. The first is lack of adequate understanding of how to analyze multiagent systems development issues in order to improve the development process i.e. apply well defined processes that allows for making good and timely design decisions. The second is lack of appropriate approaches for effectively communicating the potentials, concepts and applicability of agent technology to the wider software engineering community.

2.1.4. Addressing Agent-Oriented Software Engineering Issues

The agent community has taken different approaches to address the issues with agent-oriented software engineering methodologies and improve the adoption of agent technology.
Standards definition is one of the approaches invested in by the agent community to provide the necessary comprehension and guide for multiagent systems development. Research groups working on standardization of multiagent systems development include the Foundation for Intelligent Physical Agents (FIPA), and Knowledge-able Agent-oriented System (KAoS). Different agent-oriented software engineering methodologies and implementations of agent based systems have used the specifications of these standards. For example, the FIPA Agent Communication Language specifications and interaction protocols are used in [60] to implement the SIMBA architecture. Also, FIPA standards were extended in the simplified application of the Netbill electronic commerce [72]. These standards specifications attempt a relatively universal definition for some of the concepts of agent technology and have contributed to the universal comprehension of some of these concepts.

Different research groups embark on the design of modeling notations, techniques, toolkits, agent programming languages and development environments for modeling and implementing multiagent systems in order to address some of the challenges of agent-oriented software engineering. Some of these software development aids are for modeling only, for example Agent UML [6], while are intended to be for modeling and implementation [124]. There are well over one hundred of these tools\(^1\) developed by different agent research groups with different perceptions on the concepts of agent technology. Some are for implementation only, while some others are intended to assist agent based software development from requirements gathering to implementation. Some of them are based on existing methodologies. Examples of these include REBEL [73], Prometheus Design Tool (PDT) [97], INGENIAS-IDE\(^2\), and agentTool [23]. A good number of the others are developed independent of existing methodologies. A good number of these programming languages and development environments are either fully implemented in Java e.g. JADE\(^3\), JATLite [56] or are extensions to Java e.g. JACK. The extent of use and application of these tools to real life agent based systems is

\(^1\) [http://www.agentlink.org/resources](http://www.agentlink.org/resources)
\(^2\) [ingenias.sourceforge.net](http://ingenias.sourceforge.net)
\(^3\) [jade.cselt.it](http://jade.cselt.it)
not well documented in literature. While these efforts address a valid need for realizing agent based systems, the sheer number and diversity in conceptual framework of these tools pose challenges as to which tool is most suitable and well developed to use for effective implementation of agent based systems.

Another initiative towards enhancing the comprehension of agent technology and its potentials and addressing the challenges of agent-oriented software engineering is the Trading Agent Competition (TAC) [129, 130]. Trading Agent Competition is a forum for different agent technology research groups to demonstrate the potentials and capabilities of agent based systems developed to solve problems in particular domains. Two popular domains of TAC application include Scheduling and Supply Chain Management [129]. TAC provides a motivation for actual implementation of agent based systems that will put the capabilities of agent technology to the test. It allows for comparison of the results of the works of different research groups in modeling and developing agent based systems. Also, non agent researchers can experience the demonstration of the operation of the agent based systems competing. The emphasis of TAC is on the end result i.e. agent based systems and their performances. The software engineering processes and practices that yielded these agent based systems are not shared as part of the competition. The viability or efficiency of engineering processes and application of concepts of agent technology in building the different TAC systems are therefore not transparent to others apart from the researchers presenting a particular system.

Another approach to addressing the challenges facing agent-oriented software engineering is a process for evaluating and comparing agent-oriented software engineering methodologies. In [123], Sturm and Shehory proposed a framework for evaluating and comparing agent-oriented software engineering methodologies. The proposed framework can be used for identifying the advantages and weaknesses of agent-oriented software engineering methodologies for the purpose of improving these methodologies and ultimately enhancing the adoption of agent technology.
Specification of standards, software engineering tools, and the Trading Agent Competition are some of the initiatives by the agent technology community to address the challenges of agent-oriented software engineering, improve the comprehension of the concepts of agent technology and the attributes of agent based systems and therefore enhance the adoption of agent technology. Another approach to addressing the challenges of agent-oriented software engineering is interaction analysis.

Complexity in a system results where there are a large number of components in the system that are engaged in many interactions [48]. Furthermore, the interactions among the agents in a system represent the social process of the system and each interaction the agents engage in implies an organizational context of the system [48]. This organizational context determines the relationships among the agents in the system. It can therefore be concluded that the complexity of a system can be assessed by the nature and volume of interactions that take place among the components of the system. It also therefore follows that the behaviour of the system and its components and the relationships among the components of the system can be assessed and modeled by the analysis of the interactions among the components of the system.

The argument of this thesis is if the analysis, structuring and modeling of interactions in multiagent systems and the sharing of multiagent system development experiences are improved, the comprehension of the concepts of agent technology towards boosting the adoption of the technology can be significantly improved. The following two sections 2.2 and 2.3 discuss the issues with interactions in multiagent systems and sharing of multiagent system development experiences through software patterns in turn.

2.2 Interaction Analysis in Multiagent Systems

In a work titled ‘Social Interactions and Economic Behaviour’ [139], the author investigated why social interactions should affect economic behaviour. The author
describes different models, called interactions-based models, for studying the behaviours of societies. These interactions-based models were used to explain certain behaviours in the society that defied conventional economic models. The author then concludes that the behaviours of the members of a society are affected by their social context, that is, the interactions they engage in. For instance, the social interactions of the patients in a nursing facility can be used to assess and analyze the progress of these patients [16]. The analyses of the patient’s social interactions define their behaviour which in turn provides indications of their state of health among other things.

Generally, the interactions among the components of a system provide a basis to analyze and understand the behaviour of the system as well as the intentions of the entities in the system. Interaction analysis is used in different disciplines from economics [139] to medicine [16] to analyze, understand and model the behaviour of entities within a system. Therefore, analyzing multiagent systems by the interactions within such systems provides a relatively universal platform for reasoning about the concepts and applicability of agent technology and hence improves the appreciation of the technology. Also, interaction analysis has the potential to better deal with agent-oriented software engineering challenges relating to modeling the behaviour of the system and recognizing the intentions of the agents within the system.

This section discusses the contribution of interaction analysis to addressing the challenges facing agent-oriented software engineering.

2.2.1. Interactions among Agents

According to [136], interaction is arguably the most important single characteristic of complex software which constitutes multiagent systems. In [26], interaction is identified as an essential component of the dynamics of the multiagent systems. An essential feature of software agents and multiagent systems is social behaviour suggesting that multiagent systems can be viewed as a society of agents [26].
Regardless of the complexities or sophistication or simplicity of the individual agents in any multiagent system, a common characteristic of such systems is interaction amongst the different agents within the system.

Interaction is central to the characteristics and behaviour of an agent since every dealings/contact of the agent with its environment is one form of interaction or another. Whether an agent collaborates, cooperates, competes, negotiates, or argues, it is interacting and these are the activities an agent will engage in to achieve its goals. A multiagent system is conceptualized as a system of autonomous agents, each possessing the ability to interact with other agents to achieve its goals and/or the goals of the multiagent system.

Interactions in multiagent systems are at the knowledge level. Knowledge level interactions refer to interactions that factor in the goals of the agents and/or the goals of the multiagent system in which the agent operates; factors in which other agent(s) should be involved in the interaction and when the interaction should take place. The other dimension to the interactions in multiagent systems is context sensitivity. Due to the fact that agents have partial perception of and control over their environment [48], the environment changes in ways that cannot be predetermined by the agent. Therefore it is necessary for the agents to be capable of making adjustments to the interactions they engage in, in a manner that is consistent with the current state of their goals and the environment. The contextual nature of interactions in multiagent systems and the demand this places on the agents in the system is also a reflection of how the interactions in multiagent systems can impact on the adaptability concept of agent technology. That is, the agents in multiagent systems will exhibit adaptability by changing the way in which they interact as the system’s environmental context changes. Also, due to the knowledge level nature of the interactions in multiagent systems, interactions among agents can be used to analyze as well as model the relationships among the agents in the system.
Interactions in multiagent systems are usually represented by agent interaction protocols [45]. An agent interaction protocol is “a methodological way to conceptualize, model, and implement conversations in agent-based systems” [84]. Conversation policies, that is, agent interaction protocols, are the implementation of the principles that guide the conduct of the interactions (or conversations) between the agents in a multiagent system for those interactions to be meaningful [30].

Agent Interaction Protocols are the concrete definition and means of implementing the interactions in multiagent systems. Agent Interaction Protocols give context and direction to the interactions in multiagent systems. Agent Interaction Protocols constitute a crucial aspect of the development, the implementation and the operation of multiagent systems because interactions drive the overall behaviour of the multiagent systems and influence the decision processes of the agents in the system [93]. Agent interaction protocols are a different component of multiagent systems. Unlike the individual agents which take up specific roles [137, 58], agent interaction protocols have no function without at least two agents, and two agents cannot interact effectively without an agent interaction protocol. Hence an agent interaction protocol is defined in the light of a minimum of two agents engaged in an interaction. So where agents require attributes that will ensure the achievement of their goals, agent interaction protocols require attributes that will ensure that two or more agents with similar or divergent goals interact effectively in order to achieve their respective objectives. Examples of these properties include rule consistency, rule simplicity, inclusiveness, and architecture independence. These attributes define the identity of the agent interaction protocols and determine their success in achieving interaction in the domain of application. The nature and structure of interaction, and therefore the structure and properties of the agent interaction protocols that will achieve a specific interaction, are dependent on the characteristics of the application domain of the multiagent system in which the interaction takes place.
2.2.2. Issues with Interactions among Agents

A review of the agent-oriented software engineering methodologies reveals that most of them do not have clear processes for developing agent interaction protocols that will be used in the multiagent systems to be developed [136, 58]. The existing body of research work on agent interaction protocol is largely focused on areas such as agent interaction protocol specification methods [91, 103]; analysis of interaction types, for example negotiation, argumentation, and persuasion, and their underlying philosophies [71, 5, 85]; agent interaction protocol concatenation and extension issues [89]; languages and tools for representing agent interaction protocols [101, 102]; and implementing agent interaction protocols [68]. The design of multiagent system interactions in [22] is focused on the implementation of interaction in the individual agents within the multiagent system. It does not explicitly describe the interaction protocols that guide the interactions in the context of the domain of application. [125] describes Conversation, Message, Messaging and Protocols as different interaction concepts. These concepts express important aspects of the interactions in a multiagent system. However, the software development stage at which each concept is defined or designed is not discussed. Also, the components of a complete interaction protocol are not specified. The separation of individual agent’s internal interaction processing from the guiding principles among the agents in the multiagent system is not clear. In as much as these research efforts all have their significant contributions to the agent world, they seem to be going in their individual directions with no convergence of efforts and their results. Only a few of these works such as [45, 70, 69] are focused on defining a process for the development of agent interaction protocols. Furthermore, a good number of the existing agent interaction protocols have weaknesses. A common major weakness is that many of the existing agent interaction protocols do not have important properties such as termination and rule-consistency, which may limit their suitability for application to particular domains [85].

It is important to note that different agent interaction protocol properties or different combinations of these properties are required for different domains of
multiagent systems applications. This is because the aspiration of the agent community to make the interactions in multiagent systems as close as possible to human interactions, emphasizes the context sensitivity of inter-agent interactions and therefore the agent interaction protocols that will guide these interactions [13]. The contexts of the domain of multiagent systems applications should therefore determine the structure and properties of agent interaction protocols necessary to achieve effective and efficient interactions within the multiagent systems.

Most of the multiagent systems development methodologies that consider interaction identify the interaction needs of the system and then implement an existing agent interaction protocol. For instance, a multiagent system that requires a Negotiation type interaction may implement the FIPA Contract Net Protocol [122]. Although this suggests the software engineering concept of reuse, it does not always achieve desired results but features the following issues: the agent interaction protocol chosen may be too generic for the intended application making it inefficient; the agent interaction protocol chosen may not be comprehensive enough for the intended application; the agent interaction protocol chosen may not have the desired properties such as safety, confidentiality, or timing constraints, to ensure rich context based interaction that are well suited for the intended application. Also, it is well accepted that inadequately planned reuse is counter productive.

Based on the analysis of existing work on agent interaction protocols as presented here, the issues impacting of the analysis and implementation of interactions in multiagent systems can be summarized as the following:

1. Agent interaction protocols developed for general use may not be well suited for use in particular application domains. Where an agent interaction protocol that is defined for general use is used to model the interactions in particular multiagent systems, the agent interaction protocol may not have the necessary features that will ensure optimal performance in representing the interactions in the domain of application of the multiagent systems.
2. The properties of agent interaction protocols are not explicitly described and are not separated from the rules guiding the sequencing of the messages in the interaction. The properties that the agent interaction protocols should have in order to adequately represent the interactions in a domain are not explicitly defined. It is necessary to explicitly define these properties as they provide input into the design of the rules for sequencing of messages, the message format as well as performatives of the agent interaction protocols for the multiagent systems implementations in the domain.

3. A proper understanding of the components of a protocol is lacking, for instance describing the same message structure only with different parameters as separate protocols. Some agent interaction protocol designs define two agent interaction protocols by only changing the parameters of one. This is a case of defining two instances of the application of an agent interaction protocol as two separate types of agent interaction protocols. When an agent interaction protocol defined in this manner is applied to implement interaction in multiagent systems it will produce inadequacies in the performance of the interaction and therefore the behaviour of the multiagent systems.

4. A proper understanding of a protocol structure is lacking, for instance, the difference between an agent interaction protocol and a component of the interaction protocol, for instance performative, is unclear. One of the major factors responsible for the inadequacies in the design of agent interaction protocols and therefore implementation of interactions in multiagent systems is the lack of a structured definition of an agent interaction protocol. The definitions of the components of an agent interaction protocol and how they are related is lacking. Hence there is no way to assess completeness, appropriateness or applicability of the agent interaction protocols that are designed.

The significance and peculiarity of agent interaction protocols in multiagent systems as described above, demands a dedicated approach to the development of agent interaction protocols when building multiagent systems to be able to tackle the issues noted.
This thesis presents a dedicated approach for developing agent interaction protocols in order to address these issues with the conceptualization, modeling and implementation of interactions in engineering multiagent systems. Chapter Three of this thesis describes this dedicated approach for developing agent interaction protocols. The Chapter presents a structured definition of an agent interaction protocol, specifying the components of a protocol and how these components are related. It then describes a process for developing agent interaction protocols to realize the components and the relationship between them such that they are customized to a domain of application for which a multiagent system is being developed.

2.3 Overview of Software Patterns

Interaction analysis presented in the preceding section, discusses the agent-oriented software engineering challenges posed by the social behaviour attribute of agent technology. Other concepts of agent technology such as autonomy, proactive behaviour, reactivity, and adaptability also pose their own challenges to engineering agent based systems. Sharing experiences of what works and what does not in software engineering helps to communicate software engineering problems, context and solutions among unrelated software practitioners. Sharing software engineering experiences in a structured way can be achieved by the use of software patterns. Software patterns provide a medium for sharing software engineering experiences by describing solutions to specific problems in certain contexts. Software patterns are an expression of issues such as what works, lessons learnt and mistakes to avoid. Software patterns have recorded remarkable success in their application to the Object Oriented paradigm.

Considering the non traditional nature of the issues in agent-oriented software engineering, a structured way to share experiences of what works, lessons learnt and mistakes to avoid in the development of agent based systems is therefore of great value to the appreciation of agent-oriented software engineering challenges and comprehension of the software engineering techniques for addressing such
challenges. It is believed that agent-oriented software patterns have the potential to enhance the comprehension of the concepts of agent technology and improve the adoption of the technology by the larger community of software practitioners.

Therefore, the other approach taken in this thesis to address agent-oriented software engineering challenges posed by the other concepts of agent technology is sharing agent based software development experiences using agent-oriented patterns.

2.3.1. Software Patterns

There exists a range of different definitions for (software) patterns. These include:

- Design patterns capture the static and dynamic structures of solution that occur repeatedly when producing applications in a particular context [118];

- A pattern is a written document that describes a general solution to a design problem that recurs repeatedly in many projects [46];

- A pattern is a way of doing something or a way of pursuing an intent [86];

- Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience [100];

- A pattern is a solution to a problem in a context [17];

- Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice [1];
A close look at these definitions and the body of work on patterns reveal the following concepts of (software) patterns. These concepts set a basis for reasoning about software patterns; they set a basis for evaluating and analyzing software patterns:

The first is the concept of a problem. A pattern is written with a particular software engineering problem in focus. A design problem that recurs in different software development projects, but in similar circumstances, that is context, becomes a candidate for a pattern. As such it is necessary to be able to define the problem in context and also identify the design constraints that give definition to the problem. These design constraints are known as forces. This first concept therefore emphasizes that the right application of a pattern by software developers starts by understanding the particular design problem that the pattern addresses.

A general solution is the second fundamental concept of design patterns. A solution makes a good pattern if it is general in approach and description. Also, a general solution suggests that the solution should be adaptable to different situations. The solution should not be about the fine level of details used in a particular project; rather it should be the conceptual view of the overall solution to the problem in context. The purpose of the generality of the solution is adaptive reusability. The whole essence is for software developers to be able to clearly identify similar problems in their projects and appropriately adapt and apply the solution that the pattern proffers. A solution that stands by itself, being only applicable to a very specific problem (or project) is no more than an artifact.

A third concept is universal communication. A general solution to a design problem becomes a pattern when it can be communicated to other software developers, in a language they can readily identify with. The universal understanding of the description of the problem and its general solution requires a universal agreement on the meaning of the key elements of such descriptions.
From the definitions and the analysis of patterns, the components of a good pattern are problem, context, forces, solution and resulting context. Fundamental to the definition of software patterns therefore, is the connection between the context of the recurring problem, the constraints defined by the context which are the forces and how the solution resolves the forces to generate a resulting context. Also, the generality (adaptive reusability) of the solution is important to the definition of software patterns.

2.3.2. Advantages of Software Patterns

Software patterns are valuable to software engineering in a number of ways. In answering general questions about software design patterns, [18] discusses the following as some of the advantages of software patterns.

Software patterns help software developers to find tested and tried solutions to recurring software development problems. Certain software engineering problems recur in a number of projects. With software patterns, software developers do not have to reinvent the wheel in solving these kinds of problems. There is the added advantage that pattern solutions are solutions that have been tested. Therefore, a developer that is applying a pattern is assured the solution will work, provided it is appropriately adapted and applied.

Software patterns facilitate discussions about designs and design options among software developers. Software patterns are named representations of design options. As such, the deliberation of software developers about the design options used for a project can be made easier where some of these design options are known patterns. When a software developer mentions a software pattern as a design option that is used, the other developers with whom the pattern is familiar can immediately appreciate the structure and features of the design option being deliberated upon. This can save valuable time in communications among software developers.
Software patterns provide the structure for evaluating design trade offs. A software pattern documents the system of forces to a particular problem, how the solution resolves this system of forces and the resulting context. The resulting context is the documentation of the forces that are resolved and those not resolved by the pattern’s solution. Also, the resulting context documents any other forces that may be introduced by the application of the pattern’s solution. Developers therefore, are able to assess the trade offs that will result from a particular solution before the solution is applied. This constitutes significant savings in cost and efforts in the development process.

Software patterns provide a medium for archiving experiences, expertise and best practices. The experiences of software engineering veterans can be archived when documented as software patterns. These are expertise, bodies of knowledge and best practices that may otherwise be lost with the retirement of such experienced hands. Capturing these experiences as patterns also provides a platform for some of the software engineering practices to be improved upon. Such improvement can be achieved when developers in the future look for ways of enhancing a solution such that it resolves more forces than were resolved in the original pattern.

2.3.3. Agent-Oriented Software Patterns

Agent-oriented software patterns (agent-oriented patterns henceforth) by their structure depict the connection between software development problems and the concepts of the technology as applied in solving such problems. They present examples that bring the concepts to light. Patterns don’t just describe processes or solutions, the problem that necessitates these solutions or processes are also described. The contextual characteristics of the particular problem that justifies the solution are also described. This discussion of what necessitates or justifies a particular solution in software development enhances the comprehension of the concepts used in the solution.
The principles of software patterns are universal. Hence when properly structured and written, the features of a software pattern are usually understood by a wide range of software experts both academics and industry practitioners. The application of software patterns to agent technology allows the presentation of the concepts and principles of agent technology to cut across a range of researchers and practitioners.

The contribution of the Trading Agent Competition, which is discussed in section 2.1.4, to the adoption of agent technology would be of higher significance if the experiences of the development of the various agent based systems are documented and communicated. Software patterns provide the medium to realize the communication of such experiences in a manner that achieve universal comprehension of the issues, concepts and solutions that are be described.

2.3.4. Misconceptions about Agent-Oriented Patterns

In appreciation of the potential of agent-oriented patterns towards enabling the take up of agent technology, efforts were made by agent technology software practitioners to document different agent-oriented software engineering experiences as agent-oriented patterns. Examining some of the agent-oriented patterns in the light of the concepts of software patterns described in section 2.3.1 above indicate that there are certain misconceptions about agent-oriented patterns. These misconceptions relate to what makes a pattern in comparison with other concepts in software engineering; differences between adaptation of patterns and pattern variants; Object Oriented versus agent-oriented patterns; what constitutes a pattern problem and/or solution; and whether patterns facilitate reuse or are reused. The following are the misconceptions noted.

- A software pattern is often perceived to be synonymous with the following concepts of software engineering:
  - Frameworks: In software development, a Framework is a partial implementation of the core of a software system that is meant to be
extended in order to build more robust systems. Typically, a framework may include support programs, code libraries and a scripting language amongst other software to help develop and glue together the different components of your project.

- Components: A software component is a loosely defined term for a software technology for encapsulating software functionality. Five criteria for defining a software component: Multiple-use; Non-context-specific; Connectible with other components; Encapsulated i.e. non-investigable through its interfaces; A unit of independent deployment and versioning.

- Processes: A software development process is a sequence of steps that practitioners and managers take to create software.

- Configuration (e.g. the Sentinel Agent Behaviour Pattern): The way in which a system, whether it is hardware and/or software, is set up. Generally, a configuration is the arrangement - or the process of making the arrangement - of the parts that make up a whole.

- Method/Methodology: In software engineering and project management, a methodology is a codified set of recommended practices, sometimes accompanied by training materials, formal educational programs, worksheets, and diagramming tools.

- Model: In science and technology, a model (abstract) is understood as an abstract or theoretical representation of a phenomenon

- Pattern variants and adaptation. Differences in context and forces could result in variants to a particular pattern. However, it is necessary to draw the line between the generation of another pattern as a variant and need for adaptation. The distinction between a variant of a pattern and a case of pattern adaptation is essential because an adaptation of a pattern to a specific project may not change the fundamental structure of the solution as it may be a one-off thing that is peculiar to that particular project. In such a case, there is no new solution and therefore there is no new pattern. A pattern variant results if there is a significant change to the solution as a result of major differences in the context to the same problem.
- Classifying OO patterns as agent-oriented patterns because agents are viewed as extensions to objects. This perception of agents only as extensions to objects results in reasoning about agent technology and agent based systems as well as agent-oriented patterns in terms of OO constructs and artifacts.

- All solutions to all problems are patterns. There is an element of recurrence in the definition of a pattern. Also, a pattern solution should resolve a system of forces. See section 2.3.1 above. Therefore for a problem to qualify as a pattern, it should be a recurring problem. A one-off problem that is peculiar to a particular project should not constitute a pattern. On the other hand, not all solutions to a recurring problem qualify as a pattern. A pattern solution should be a tried and tested solution which resolves the system of forces of the recurring problem in question.

- Patterns are “plug and play”. Patterns are not reused; they enhance reuse of the quality and concept of the solution that they describe. According to [1], a pattern is said to describe the core of the solution in such a way that it can be adapted and used several times over without doing it the same way any two times. This emphasizes the fact that patterns are not meant to be taken and just plugged in to a design. Patterns are intended to describe the important concept of a solution such that the concept can be adapted and applied variously in similar but different situations.

The discussion of these misconceptions provides an insight into the cause of some of the issues that impact on the quality and the realization of the potentials of agent-oriented patterns. This thesis focuses on the issues that relate to the classification and description of agent-oriented patterns. In sections 2.3.5 and 2.3.6 below, the specifics of these issues in the context of the existing body of work in agent-oriented patterns as they relate to classification and description of agent-oriented patterns are discussed. In section 2.3.5, the issues relating to agent-oriented pattern classification in the context of the existing body of work relating to
agent-oriented pattern classification are discussed. The issues relating to agent-oriented pattern description are discussed in section 2.3.6 in the context of the existing body of work relating to agent-oriented pattern description.

2.3.5. Classification of Agent-Oriented Patterns

Pattern users require some categorization [11] to ease the process of finding appropriate patterns for the problem they have to solve [40, 117]. Patterns should be categorized according to some criteria which the potential pattern users can relate with in order to be able to make informed search for patterns that will address their concerns in developing agent based systems. The other dimension to the issue of categorization of patterns is that pattern writers need to place the patterns they write in categories. The categories in which pattern writers place their patterns should be put into consideration the fact that potential pattern users, who are not party to the writing of the patterns, will have to search for patterns that are applicable to their problems. Classification of agent-oriented patterns is therefore essential for the following reasons.

- A classification scheme gives a definition to the pattern. For instance, a pattern in the structural class of the Gang of Four (GoF) patterns [34] automatically has an implicit definition by the category it falls into. This implicit definition allows pattern users to know what to expect from the pattern and the circumstances in which the pattern may be applicable.

- A classification scheme is necessary for agent-oriented pattern catalog. It is essential to build a repository of agent-oriented patterns and a classification scheme is the first step in that direction.

- A classification scheme can be used for the assessment of the quality and usability of existing agent-oriented patterns. The classification scheme specifies the attributes of the patterns that should belong to the individual categories in it and these attributes form the basis of the assessment of patterns.
A classification scheme provides a guide for design of agent-oriented pattern templates. The basis for different pattern templates is defined by the attributes of the classifications. Basing the template designs on the classifications removes the risk of ad hoc template designs.

There are a number of classification schemes by different agent-oriented pattern writers. The description and analysis of some of the existing agent pattern classification schemes are presented as follows.

The motivation of the work in [40] is the need to develop a sound software engineering methodology for multiagent systems due to the increasing sophistication and variety of such systems. In pursuing this goal, the authors present a catalog (collection) of collaboration patterns (architectural design patterns) for multiagent systems. They are of the opinion that such patterns represent a unified documented experience base which is valuable in maintaining and reusing implemented agent-oriented systems. The authors recognize that selecting the appropriate pattern for a development problem requires the establishment of a comprehensive catalog of agent patterns. This establishment first of all involves defining a proper catalogue structure or classification scheme. Then agent-oriented patterns can be collected and categorized using the classification scheme, in a fashion that facilitates the selection of appropriate patterns. The authors propose four types of coordination patterns (called architectural styles). These are hierarchical, federated, peer-to-peer and agent-pair. They present five examples of federated patterns which are Broker, Embassy, Mediator, Monitor, and Wrapper. These categories represent the grouping of a particular type of patterns, and are not intended to be comprehensive enough to represent the feasible spectrum of agent-oriented patterns.

Sauvage’s works on agent-oriented patterns [116, 114, 115] highlight the gap between the modeling of the concepts in agent based systems and actual implementation of these concepts in multiagent systems for real world problems. It recognizes that most of the existing work on agent patterns are significantly about
OO design patterns and then emphasizes that the difference between objects and agents makes OO design patterns inappropriate for agent-oriented development. It presents a diagrammatic representation of agent patterns categorization in the form of OO inheritance. The categories are Metapatterns, Metaphoric patterns, Antipatterns, inheriting from Pattern. These categories do not represent any conceptual idea. Metapatterns, according to the description and example given, are patterns that present general overviews of some of the concepts of agent technology e.g. organization, protocols. They do not address specific agent-oriented software engineering problems and therefore describe no particular software engineering qualities in their solutions. Metaphoric patterns are so grouped based on the source of the inspiration of the solution. In this work, antipatterns are patterns that describe how to correct common mistakes. This is a different definition to some other definition of antipatterns as patterns that represent ways to not solve a particular problem so as to streamline a designer’s effort in looking for a solution. That is, antipatterns are equally good solutions in their own rights; however, they are not applicable to the context in question. Architectural patterns are also described even though they are not presented in the diagram of the agent pattern categories. The underlying basis for the classification of agent-oriented patterns in this work is neither the problems that the patterns solve nor the solutions they proffer. As such any pattern could fall into any of the categories. Also, the categories do not relate to levels of abstractions in agent-oriented software engineering. There is no information as to the extent of the actual usage of this classification in the work.

The work of [117] proposes a classification along two dimensions for agent implementation patterns. These dimensions are Context and Realization. Context specifies the part of the multiagent systems where the pattern can be applied. These parts, according to the authors, are Environment, Agent and Multi-agent. “Realization describes the kind of problem the pattern is trying to solve.” The authors present two kinds of problems, these are System and Concept. This classification was designed for patterns at the implementation phase of agent based software development. It was not intended to be comprehensive enough to
represent the feasible spectrum of agent-oriented patterns. The classification is based on the authors’ experience in developing agent based systems using Object Oriented programming and was used to classify the patterns they recognized in their experience. The applicability of the classification to other patterns and to other programming paradigms for implementing agent based systems is not documented. The authors also stated that the classification is a starting point for further discussions.

Aridor and Lange, in their work on implementing mobile agents, identified and described a number of agent-oriented patterns [2]. The work includes a classification for the patterns identified. The pattern classification categorizes agent-oriented implementation patterns into three categories which are Traveling, Task and Interaction patterns. No criteria are presented for the classification and the patterns are applicable only at the implementation phase of the development of mobile agents using Object Oriented programming.

Tahara et al in [126] proposed a method for the development of agent based systems using agent patterns. This method, which is based on their Plangent agent platform, defines a layered architecture for the development of agent based systems. The architecture has three layers. The first layer which is the Macroarchitecture layer represents the outline of the system configuration and agent behaviour. It is not based on the agent platform. The second layer which is the Microarchitecture layer represents the details of the system configuration and agent behaviour. It is based on the agent platform (Plangent, Aglet). The third layer is the Object Oriented level. It represents the code for the implementation of the two layers above. Classification of agent patterns according to Tahara et al is based on this layered architecture. Therefore, there are Macroarchitecture agent patterns, Microarchitecture agent patterns and Object Oriented design patterns. At the Macroarchitecture level, they further classify patterns into Mobility and Coordination Protocol patterns. The Aridor and Lange patterns [2] are classified under the Microarchitecture category of patterns. This classification of agent-oriented patterns has a structure according to the layers of the architecture of the
Plangent agent platform developed by the authors. Though the focus of this classification is on the development of mobile agents and it assumes Object Oriented implementation of the agent based systems, it recognizes the need to classify agent-oriented patterns according to the levels of abstractions in agent-oriented software engineering.

Heinze identified and documented a number of agent patterns based on his experience design and implementation of agent-oriented systems [41]. In his work, Heinze presented a set of agent-oriented patterns for the purpose of modeling intention recognition in individual intelligent agents according to the contexts and forces of the agents in the multiagent systems. Heinze acknowledges that the OO classifications of software patterns are not adequate for agent-oriented patterns giving reasons such as differences in possible implementation languages for agent systems and increase in the levels of abstractions in agent based development.

In his work on patterns for agent-oriented software engineering, [77] proposed a catalogue structure for agent-oriented patterns. The proposed catalogue structure is based on the view oriented approach, an approach for developing agent-oriented systems presented also by Lind [76]. Views represent different perspectives from which an agent-oriented system can be modeled. According to Lind, the views in the view oriented approach represent the decomposition of a system and have proven to be effective developing agent based systems. Therefore, he proposes the views as a starting point to the definition of the categorizations of agent-oriented patterns. These views are Interaction, Role, Architecture, Society, System, Task, and Environment. These views are not directly related to the levels of abstractions in agent-oriented software engineering. However, they are based on the concepts of agent technology. Therefore, using these views as the categorizations of agent-oriented patterns emphasizes the importance of reflecting the notions of agent technology in agent-oriented pattern classification [41].

The different works reviewed above present various perspectives for classification of agent-oriented patterns. The various perspectives in these works emphasize the
authors’ recognition of the importance of classification of agent-oriented patterns. Close studies of the various perspectives to classification of agent-oriented patterns presented by the different authors indicate certain inconsistencies and inadequacies which may be summarized as follows.

Firstly, the authors that provide the criteria for the design of their classification of agent-oriented patterns have different criteria for the classification of a similar category of patterns. For instance, where a group of authors define macro and micro architecture categories of agent-oriented patterns based on the architecture of their agent platform [126], another author defines an architecture category of agent-oriented patterns based on perspectives of modeling agent based systems [77]. On the other hand, some authors do not provide any criteria or rationale for the design of their classification schemes.

Secondly, a good number of the classifications are not comprehensive. That is, they are not designed to capture all the different phases and process of agent-oriented software engineering. Some of the classifications are designed according to the levels of abstractions while other classifications are based on the notions of agent technology but not in the context of agent-oriented software engineering processes. Some classifications focus on either only the implementation phase of development or only on some aspect of the design phase. Other classifications are based on implementation of only a particular application of agent technology, for example, classification of patterns for implementation of mobile agents. In other words, a number of existing classifications neither reflect the different levels of abstractions in agent based system development nor the concepts of agent technology that are relevant at the different levels of abstractions.

These inconsistencies and inadequacies affect the application of agent-oriented patterns in the following ways:

- It becomes difficult for non agent software practitioners working with agent patterns to understand the different aspects of agent based systems.
development. Where agent-oriented pattern classifications are not designed according to the levels of abstractions and processes in agent-oriented software engineering, non agent software practitioners wishing to work with these patterns may not be able to recognize what stage of agent-oriented software engineering the patterns relate to. This is particularly so because different stages of agent-oriented software engineering feature different concepts of agent technology.

- Potential pattern users do not have adequate criteria to search for suitable patterns to solve their problems. A very important essence of agent-oriented pattern classification is to facilitate the process of searching for agent-oriented patterns. If however, pattern classifications do not facilitate this process, they would not be achieving one major reason for designing them in the first place. The inconsistencies in the classification criteria of most of the existing classification schemes as well as absence of classification criteria altogether for some other classification schemes make it very difficult to define a systematic process for searching for patterns classified according to the classification schemes.

- Different pattern writers either do not classify the patterns they write or they design their own classifications. Due to the fact that existing classification schemes either do not have classification criteria or have incompatible criteria and classification schemes are not comprehensive, pattern writers have difficulties in determining what categories to place the patterns they write. Therefore, pattern writers design their own classification scheme to categorize their own patterns, further complicating the situation.

- It is difficult to extend or improve on the design of existing classification schemes. The classification of agent-oriented patterns may require amendments and extensions as the scope of experiences in agent technology increases. With the absence of classification criteria or inconsistencies in the criteria for the design of most of the existing agent-oriented pattern
classification schemes, improving or extending agent-oriented pattern classifications would be almost impossible.

- Combination of agent-oriented patterns written by different authors, into composite patterns or pattern languages is nearly impracticable. This is due to the fact that the classification criteria of agent-oriented pattern classification schemes are inconsistent and the levels of abstractions of agent-oriented software engineering that the classification schemes capture are not defined. These factors make it difficult to understand the attributes of the patterns and place them in the appropriate context of agent-oriented software engineering. For instance, attempts may be made to combine two patterns that belong to different levels of abstractions as though they are at the same level because the authors’ classifications do not reflect the levels of abstractions at which the patterns are applicable.

It is believed that these issues may impact negatively on the ability of agent-oriented patterns to achieve their potential in enhancing the comprehension and adoption of agent technology by the larger community of software practitioners. In other words, these issues may limit the potential of agent-oriented patterns to address some of the software engineering challenges posed by the concepts of agent technology.

The considerations that are critical to the design of a comprehensive agent-oriented pattern classification scheme can be deduced from the summary of the inconsistencies and inadequacies presented above and the impact of these on the potential of agent-oriented patterns. Some of the different works in agent-oriented patterns identify and emphasize or apply different aspects of these considerations in their work. In this light, [41] acknowledges that Object Oriented classifications are not adequate for classifying agent-oriented patterns. The need for classification of agent-oriented patterns along the levels of abstractions in agent-oriented software engineering is suggested in [126] while [77] recognizes the importance of reflecting the notions of agent technology in the classification of agent-oriented
patterns. Another major consideration is the definition of consistent criteria for designing the categories in the classification scheme. These considerations should all be incorporated in the design of a comprehensive classification scheme for agent-oriented patterns.

To this end, this thesis presents a Two Way Classification Scheme for classifying agent-oriented patterns. The Two Way classification scheme is designed to reflect the levels of abstractions in agent-oriented software engineering as well as the notions of agent technology that are relevant to the different levels of abstractions, in the definition of the categories. The Two Way classification scheme and how it addresses some of the issues relating to agent-oriented pattern classifications are presented in Chapter Four of this thesis.

2.3.6. Description of Agent-Oriented Patterns

The quality of a pattern is a function of how it is described. The value in a pattern can only be communicated if the pattern is described in a way that facilitates the expression of its values. The values of a pattern can be expressed if its description contains the features of a good pattern which are problem, context, forces, solution and resulting context (see section 2.3.1) and realizes certain qualities that define a good pattern. According to Corfman [19] these qualities of a good pattern include the following.

- A pattern should be clear and complete. That is, a pattern should stand on its own. A pattern should not require a reference to another material for its essence to become clear. A pattern should contain within itself, necessary information to convey the problem, context and solution.

- There should be a clear understanding of the solution. The solution of a pattern should be comprehensible by a broad spectrum of software practitioners. This suggests that the solution should be presented using description formats that will facilitate clear comprehension by interested software practitioners.
- There should be a clear understanding of the trade-offs that the pattern makes. A pattern should reflect the impact of applying its solution to the context of the system. The pattern should clearly state if the application of its solution introduces other problems to the system. The pattern should make clear which constraints are addressed, which constraints are not addressed, and which constraints are introduced or emphasized by the application of its solution.

- There should be a clear understanding of when it is or when it is not appropriate to apply the pattern. The applicability of a pattern is as important as the solution it describes since the relevance of the solution is determined by the context in which the problem recurs. The pattern should therefore explicitly express the context in which it becomes applicable.

Regardless of the efforts put into identifying a pattern and the wealth of experience of the pattern writer, the pattern form or pattern template used in describing the pattern determines if the qualities of a good pattern will be expressed in the pattern. Therefore, the pattern template should be designed to allow for adequate expression of these qualities since the quality of a pattern’s description is determined by the pattern template used in describing the pattern. The expression of the qualities of a good pattern in pattern descriptions is quite significant to agent-oriented patterns because of the non traditional nature of the concepts of agent technology that agent-oriented patterns are intended to communicate. That is, agent-oriented patterns should be clear and complete and expressed with minimal complexities; present clear description of the concepts in the solution to facilitate comprehension by non agent software practitioners; clear presentation of the applicability of the patterns such that software practitioners can understand the contexts to which the concepts and solutions of agent technology are applicable. It can therefore be concluded that the pattern template makes significant contribution towards the overall quality of agent-oriented patterns and the delivery of the potential of agent-oriented patterns to the adoption of agent technology.
According to [111] there is a widespread agreement on the importance of pattern template to software patterns, however, there is not as much agreement as to what should constitute a pattern template. There are several pattern templates, used, adapted or designed for describing agent-oriented patterns. The description and analysis of some of these pattern templates are presented as follows.

Chacon et al [14] present two patterns without using any template. The two patterns were described by continuous blocks of text. This requires a potential user of the patterns they describe to figure out the problem, the solution and the applicability of these patterns. This type of description may result in subjective interpretation of the essence and the components of the patterns presented. The template used for describing these patterns cannot be assessed for expressiveness as this is a no element pattern template.

Tahara et al [126] describes the summaries of a set of agent-oriented patterns with a three element template. These template elements are Purpose, Applicability and Forces. Tahara et al states that patterns are in general described with templates having more than ten elements. This statement suggests the following. One, it suggests that a single description template is good enough for all categories of agent-oriented patterns. Two, it suggests that not much importance is given to pattern description template as there is no reference to the particular template, the features that this template has or the precise number of these features. The authors used three template features to describe the summaries of the agent-oriented patterns they presented. These are Purpose, Applicability and Forces. The authors did not present the pattern summaries in a uniform structure with the use of these three template features. For some patterns presented, the descriptions were structured according to each of these three features while other patterns were described as continuous blocks of text without reference to these three features. These three features address some of the important aspects of a pattern, but the solution and resulting context of the patterns described are not included in these features and these features do not reflect the concepts of agent technology.
Sauvage uses a pattern template with eight elements to describe the patterns presented in [116]. He also presented three categories of patterns which are Metapatterns, Metaphoric and AntiPatterns. The elements of the pattern template presented here are Name, Synopsis, Forces, Examples, Solution, Implementation, Examination, and Associated Patterns. Forces is replaced with Concepts and Implementation is removed from the template elements listed, for describing patterns that fall into the Metapatterns category. An element called Origin is added to the elements listed for describing the patterns that fall into the Metaphoric category. Dys-solution and Symptoms are added for describing the patterns that fall into the AntiPatterns category. This difference in template elements for the different categories is recognition of the fact that the same pattern template cannot be used for all categories of agent-oriented patterns. The use of the Concepts element ensures that the concepts of agent technology are captured in the patterns described using this template, it is however restricted to the description of the patterns at the Metapatterns category only. By studying the template elements and the actual patterns described using these elements, it is noted that the problem that the patterns are addressing are not explicitly stated. Also, the context and the forces are not explicitly described.

Weiss [128] presents a pattern language describing the individual patterns with a template of five elements similar to the basic Alexandrian form. The elements are Context, Problem, Forces, Solution, and Resulting Context. This template is valuable in that it defines the major components of a pattern. However, it does not specify elements for specifically capturing the concepts of agent technology in the description of agent-oriented patterns. Also, this work assumes that the same template is adequate for describing all categories of agent-oriented patterns.

Aridor and Lange [2] use an abridged GoF template [34] having six elements for describing their agent mobility patterns. The elements used include Intent, Motivation, Applicability, Participants, Collaboration and Consequences. Schelfthout et al [117] present their agent implementation patterns using an adapted version of the GoF template having seven elements. The adapted template has
elements that include Intent and Motivation (as one element), Applicability, Structure and Participants (as one element), Collaborations, Consequences, Known uses, and Related Patterns. Hayden et al [40] described patterns for multiagent coordination using the GoF template. Aridor and Lange’s abridged GoF form may be appropriate for their patterns since they describe OO concepts for implementing agent systems. However, it is not certain that the adapted GoF template is adequate for describing the Schelfthout et al’s patterns as these patterns are intended to convey certain concepts of agent technology at the implementation phase of multiagent systems development. It is also not certain that the adapted GoF template is adequate for describing the Hayden et al’s patterns as these patterns are intended for conceptual modeling of multiagent systems where the emphasis is on concepts of agent technology. It is worthy of note that the fact that Aridor and Lange and Schelfhout et al either abridge or adapt the original GoF template before using it suggests that they acknowledge that one template may not be adequate for describing all types and categories of patterns. Also, from the examination of some of the patterns described by these authors, it is observed that there may be ambiguity in the meaning of the template element Intention. While Hayden et al used it as a synopsis of the Solution, Schelfthout et al used it as a description of the Problem.

Lind argues that a single general template is inadequate for agent patterns [77] and proposes a two part pattern description template having a generic part and a view-specific part. The elements of the generic part of the template are Name, Aliases, Problem, Forces, Entities, Dynamics, Dependencies, Example, Implementation, Known Uses, Consequences and See Also. The elements of the view-specific part for the architecture view include Resource Limitations, Control flow, Knowledge handling, Reasoning capabilities, Autonomy, User interaction, Temporal context, and Decision making. From the review of this template structure, it is noted that some of the elements of the generic part, such as Implementation, may not be applicable to categories of patterns at the higher levels of abstractions. There is some relationship between some generic part elements, for instance Forces, and some view-specific part elements, for instance Resource limitations, and Temporal
context. However, these relationships are not indicated in the structure of the template.

Malyankar [83] proposes a pattern template for social structures in agent systems by adding elements and sub categories to the GoF template. The elements introduced to the GoF template include Dependencies, and State storage. The Dependencies element is subdivided into Dependencies managed and Dependencies affected. The Consequence element is subdivided into Scaling behaviour, Emergent effects and Pathology. These added elements do not reflect the notions of agent technology; rather they describe general software engineering issues such as scalability.

Lind’s and Malyankar’s approaches to agent-oriented pattern templates feature a two-part pattern template structure. In both approaches, some context specific elements are added to general elements to make up their templates. The two-part template structure adds flexibility to the design of agent-oriented pattern templates. These works form bases for the design of agent-oriented pattern template structure presented in Chapter Six.

The different works reviewed present different dimensions to the construction of templates for the description of agent-oriented patterns. These differences in agent-oriented pattern templates support the observation of [111] regarding awareness about the importance of pattern templates and lack of agreement about what constitutes a pattern template. Examination of the various templates used by the different authors reviewed, for describing agent-oriented patterns reveal certain inadequacies which may be summarized as follows [96]:

- Most of the existing templates do not feature the concepts of agent technology. Most templates do not have elements such as roles, goals, agents, and ontology that explicitly represent the concepts of agent technology. The inclusion of these kinds of elements in the pattern templates ensures that the represented concepts are described in agent-oriented patterns. Some of the templates that
do have agent concepts do not structure them in a way that maintains the features of a good pattern. Therefore, these templates cannot effectively facilitate the comprehension of agent technology.

- Most of the templates presented lack clear rationale for the introduction of template elements or for the shortening of existing templates. For instance, two adaptations of the same template end up with two separate templates with different number and structure of elements. This makes it difficult to compare patterns written by different authors; combine patterns written by different authors into pattern languages; or integrate the patterns into existing design.

- A number of the templates proposed are based on the assumption that one template is good for all categories of patterns. That is the templates are not tailored to particular categories of agent-oriented patterns. Using the same template for different categories of agent-oriented patterns will result in either redundancy of template elements or inadequacy of template elements for describing the patterns.

- There is ambiguity in the meaning of the elements of some of the templates proposed or used. This means the same template element which appears in different templates is used to represent different features of a pattern in the different templates. For instance, the Intent element of the GoF template is used in some adaptations of the template to represent the problem feature of a pattern and the solution feature of a pattern in others. This ambiguity impacts on what pattern writers describe under particular elements of a template and how users interpret what is described.

These inadequacies reduce the expressiveness of the description of agent-oriented patterns thereby affecting the quality of these patterns and the delivery of the potentials of agent-oriented patterns.
In order for agent-oriented pattern templates to appropriately express the qualities of agent-oriented patterns and ensure the delivery of the potentials of agent-oriented patterns, they should have elements that will achieve the following objectives:

- Explicit representation of the features of a good pattern, that is problem, context, forces, solution and resulting context
- Representation of the concepts of agent technology
- Facilitation of adaptation, integration and combination of agent-oriented patterns.
- Systematic and rational definition of agent-oriented pattern templates for different categories of agent-oriented patterns
- Expressiveness of agent-oriented pattern contents and simplicity of presentation
- Uniform interpretation of agent-oriented patterns template elements and pattern contents
- Facilitation of universal comprehension of the contents of agent-oriented patterns

It is therefore argued that achieving the objectives enumerated above requires that agent-oriented pattern templates reflect the concepts of agent technology that is structured around the basic features of a good pattern (i.e. problem, context, forces, solution, and resulting context). Also, agent-oriented pattern templates should be relevant to the different categories of agent-oriented patterns and they should be expressive enough to facilitate communication, adaptation, integration and combination of agent-oriented patterns. To this end, this thesis presents a structure for agent-oriented pattern template. This structure is used to define templates for different categories of patterns in the Two Way classification scheme. The template structure for agent-oriented pattern templates and the category specific templates based on this template structure are presented in Chapter Six of this thesis.
2.4 Chapter Summary

This Chapter described agent technology, the concepts of agent technology, the application domains that this technology is relevant to and agent-oriented software engineering. The setbacks to the adoption of agent technology and the challenges facing agent-oriented software engineering were discussed. The significance of interaction analysis to the understanding of the behaviour of multiagent systems and the significance of sharing agent-oriented software engineering experiences to enhancing the adoption agent technology were presented. The Chapter went on to highlight the issues with interaction analysis and agent-oriented software engineering experience sharing in the context of existing work.

The next four Chapters present the contribution of this thesis towards addressing the issues highlighted with interaction analysis and experience sharing. Chapter Three describes the dedicated approach for developing agent interaction protocols. Chapter Four presents the Two Way classification scheme, an agent-oriented pattern classification scheme that provides the framework for classifying, analyzing, and describing agent-oriented patterns. Chapter Five describes a process for analyzing agent-oriented patterns while Chapter Six designs an agent-oriented pattern template structure based on the classification scheme presented in Chapter Four.
Chapter 3

A Dedicated Approach for
Developing Agent Interaction Protocols

3.1 Introduction

The previous Chapter discussed existing work in the modeling of interactions in multiagent systems using agent interaction protocols. Section 2.2.2 of Chapter Two highlighted the issues with the ways agent interaction protocols are currently used in developing multiagent systems as follows:

- Agent interaction protocols developed for general use may not be well suited for use in particular application domains
- The properties of agent interaction protocols are not explicitly described and are not separated from the rules guiding the sequencing of the messages in the interaction
- A proper understanding of the components of an agent interaction protocol is lacking, for instance describing the same message structure only with different parameters as separate protocols.
- A proper understanding of a protocol structure is lacking, for instance, the difference between an agent interaction protocol and a performative is unclear.

This Chapter presents a dedicated approach for developing agent interaction protocols to address these highlighted issues.
Interaction in multiagent systems is at the knowledge level and significantly context sensitive (section 2.2.1) when compared with data communication in traditional software systems. The aspiration of the agent community is to make these interactions as close as possible to human interactions. Multiagent system interactions are reasoned about and modeled in terms of the real life concepts involved in the interactions. Agent interaction protocols are used to model and implement the structure and control of the interactions in a multiagent system.

An agent interaction protocol has its particular identity in a multiagent system, that is, a component part of the multiagent system that is present to serve the interaction needs of the agents while remaining separate from the individual agents within the system. As a result, it is essential to be able to define the components of an agent interaction protocol and the relationships among these components. The arrangement of the components of an agent interaction protocol is referred to in this Chapter as agent interaction protocol structure (or protocol structure).

By their function, agent interaction protocols represent some of the peculiar characteristics and behaviour of a multiagent system application domain. Negotiation in a business to business transaction is different in some attributes from negotiation in a business to consumer transaction. While one may be critical on time, for instance a business to business transaction for raw materials required for production scheduling, the other may not be. Another example is the difference in the cooperation that is required amongst a set of agents assisting surgeons in a critical operation when compared to the cooperation among a set of telecommunications service provider agents in a bid to present a common tariff regime to their customers.

It can be seen from these two examples that the dynamics of the interactions in the different domains reflect the peculiar characteristics of these domains. Hence, the applicability of multiagent systems to different domains is affected by the conceptualization and implementation of inter-agent interactions in the system using agent interaction protocols. Also, these two examples show that the
differences in the agent interaction protocols implementing similar interaction types needed for different domains could either be subtle, requiring certain protocol property modifications or fundamental, impacting on the entire structure of the agent interaction protocol. Agent interaction protocols therefore need to be context sensitive in order to achieve near human interactions among the agents in a multiagent system [13]. The collection of the agent interaction protocol properties is described as agent interaction protocol property suite (or protocol property suite) in this Chapter. Even though they are distinct from one another, there is a connection between the agent interaction protocol property suite and the agent interaction protocol structure.

The definition of the agent interaction protocol structure and the agent interaction protocol property suite are the goals of the dedicated approach for developing agent interaction protocols presented this Chapter.

Section 3.2 of this Chapter presents the agent interaction protocol structure. The components of an agent interaction protocol and their arrangement are described in this section. The definition of the agent interaction protocol structure draws on the work of Holzmann on the design of computer protocols. The agent interaction protocol property suite is presented in section 3.3. Also described in this section is the relationship between the protocol structure and the protocol property suite. Section 3.4 presents the dedicated approach for developing agent interaction protocols. This section discusses domain analysis which is used for identifying the components of the protocol structure and defining the protocol property suite. The phases of the dedicated approach and the deliverables of these phases in terms of the protocol structure and protocol property suite are also described in this section. An example of an agent interaction protocol analysis defined using the protocol structure and protocol property suite is presented in section 3.5. Section 3.6 is the Chapter summary and conclusion.
3.2 Agent Interaction Protocol Structure

An agent interaction protocol structure is the organization of the components of an agent interaction protocol. The agent interaction protocol structure defines the components of an agent interaction protocol and how these components are related to one another. The agent interaction protocol structure can be used to assess the completeness of agent interaction protocols. Also, the agent interaction protocol structure provides a basis for amending, modifying, extending and nesting agent interaction protocols.

The activities involved in describing the agent interaction protocol structure are identifying the components of a complete agent interaction protocol and defining the arrangement of the components of a complete agent interaction protocol. These activities are presented in the following subsections.

3.2.1. Agent Interaction Protocol Components

A specification of the elements of computer network protocols is adapted for the identification of the components of agent interaction protocol. According to Holzmann in [43] a computer protocol specification should have five elements as follows: Service, Assumptions about the environment, Vocabulary, Encoding (format), and Procedure rules.

The service specification describes the objective of the protocol. For instance, a protocol’s objective could be the transfer of images over a fibre optic channel. The assumptions about the environment refer to the assumptions about network elements. Network elements are the nodes on the network and its transmission channel. These network elements constitute the environment in which the protocol is executed. The vocabulary specification defines the types of messages that will be used in the protocol. The type of message is defined by words with associated meanings in the context of the protocol. An example of a vocabulary element is \textit{ack} which represents a message combined with a positive acknowledgement. The
message format specifies the number and types of fields that makes up each message. Examples of types of field include control field and data field. The procedure rules specify the conditions for the exchange of messages among the network elements. This element ensures consistency of message exchanges.

[43] goes on to compare a protocol specification with a language definition. The comparison states that a protocol definition “contains a vocabulary and a syntax definition (i.e. the protocol format); the procedure rules collectively define a grammar; and the service specification defines the semantics of the language.” This analogy emphasizes the importance of these elements to the definition of a protocol.

The description of the protocol elements and the language analogy of a protocol shows that the vocabulary, the syntax (or format) and the procedure rules of a protocol determine the dynamics of the protocol, that is the process of the protocol’s execution. The service and assumption elements determine the characteristics or the specifications of the other three elements.

These elements of a protocol are also applicable to agent interaction protocols because the concept of protocols in data transmission, which is the management and control of interactions, is the same in agent interaction. The difference between both types of interaction is in their respective context sensitivity and knowledge level.

Given that interactions in multiagent systems are context sensitive and at the knowledge level, the components of agent interaction protocols are defined as follows:

- Purpose which represents the objective of the protocol
- Players which represent the details of the protocol’s environment
- Performatives which represent the types of messages that are featured in the protocol
- Message Structure which represents the format of the messages that are featured in the protocol
- Message Exchange Rules which represent the policy for the exchange of messages in the protocol

The Purpose and Players components relate indirectly to the Service and Assumptions about the Environment of the Holzmann protocol elements. Performatives, Message Structure and Message Exchange Rules map to the Vocabulary, Encoding and Procedure rules of the Holzmann protocol elements. However, the agent interaction protocol components are defined to reflect the context sensitivity and knowledge level requirements of interactions in multiagent systems as expressed in the following descriptions of these components.

### 3.2.1.1. Purpose

The purpose component of the protocol structure is the most significant component. It is the representation of the analysis of the interaction that the protocol is being designed to implement. This component provides a basis for the specification of the interaction model (specified in the Players component) in the light of the characteristic features of the interaction domain. It is not just a description of the essence of the agent interaction protocol to be built, it is a structured specification that describes the interaction behind the agent interaction protocol. Hence, this component determines the kind and structure of the messages and the rules that will guide the exchange of these messages amongst the interacting agents. The purpose specification also describes the properties of the protocol and these influence the other components of the protocol structure. See Table 3.1 for the definition of the elements of the purpose component. These elements may not be exhaustive, however, it is believed that they capture the fundamental properties of this component.

### 3.2.1.2. Players

The agents involved in the interaction are described as the players. This component of the protocol structure documents the interacting agents and their roles within the
multiagent systems. The relationships between these agents are also specified in this component. These relationships include organizational hierarchies, buyer/seller, and competitor relationships. Based on the relationships, restriction on interaction involvement to either any or specific instances of certain agent roles are specified in this component as well as the part (initiator, participant or responder) to be played by particular agents in the interaction. The Players component also describes the interaction mode i.e. bilateral or multilateral interaction. See Table 3.2 for the definition of the elements of the Players component. These elements may not be exhaustive, however, it is believed that they capture the fundamental properties of this component.

### 3.2.1.3. Performatives

A Performative is a primitive message type [3, 74]. This component is a listing of all the performatives that will be used in the agent interaction protocol and their meanings. It is important to clearly define the meaning of the performatives in

---

**Table 3.1: Elements of the Purpose Component**

<table>
<thead>
<tr>
<th><strong>Interaction:</strong></th>
<th>A statement of the interaction e.g. Stocks Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Related System Goal:</strong></td>
<td>A statement of the system goal that requires the interaction e.g. optimize investments</td>
</tr>
<tr>
<td><strong>Domain of Application:</strong></td>
<td>A specification of the domain stratification e.g. Business-Stock Exchange-Stock Market</td>
</tr>
<tr>
<td><strong>Domain Type:</strong></td>
<td>A specification of the type of the domain e.g. open, distributed, closed, and real-time</td>
</tr>
<tr>
<td><strong>Interaction Objective:</strong></td>
<td>A description of the essence of the interaction within the system and in the context of the domain e.g. ‘investigate performing stocks’</td>
</tr>
<tr>
<td><strong>Interaction Type:</strong></td>
<td>A specification of the type of interaction e.g. negotiation, collaboration, competition</td>
</tr>
<tr>
<td><strong>Pre conditions:</strong></td>
<td>A specification of the system state necessary to trigger the interaction e.g. presence of an open order on the stock exchange</td>
</tr>
<tr>
<td><strong>Post conditions:</strong></td>
<td>A specification of the expected system state (or possible states) after a successful completion of the interaction e.g. the stock is purchased</td>
</tr>
</tbody>
</table>
order to avoid misinterpretation by the interacting agents. Connections between a performative and an agent or interaction part (initiator or responder) are specified in this component, for instance, the specification that a performative is an interaction initiating performative. Also, the number of times it is permitted to have a performative in an interaction is specified in this component.

3.2.1.4. Message Structure
A message is defined by a performative, however, the structure of each message in the interaction is a specification of the information that the message will carry when it is sent. The different fields that each message should have and a

<table>
<thead>
<tr>
<th>Interacting Agents:</th>
<th>A specification of the type of agents involved in the interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiator:</td>
<td>A specification of the agent that initiates the interaction</td>
</tr>
<tr>
<td>Responder(s):</td>
<td>A specification of the agent(s) allowed to respond to the Initiator</td>
</tr>
<tr>
<td>Inter-agent Relationships:</td>
<td>A description of the association between the interacting agents e.g. Client/Server, Buyer/Seller</td>
</tr>
<tr>
<td>Privileges:</td>
<td>A specification of the permission given to one of the participants to change the rules of the interaction</td>
</tr>
<tr>
<td>Number of Agents:</td>
<td>A specification of the number of agents to be involved in the interaction if known (or range i.e. Greater than two, if the number is not known)</td>
</tr>
<tr>
<td>Diversity of Agents:</td>
<td>A description of the source of the agents in the interaction i.e. Heterogeneous or Homogeneous agents</td>
</tr>
<tr>
<td>Distribution:</td>
<td>A description of how the initiator(s) connect with the responder(s) based on the number of each category of agents interacting i.e. One-to-Many, Many-to-One, or Many-to-Many</td>
</tr>
<tr>
<td>Accessibility:</td>
<td>A specification of the initiator agent’s awareness of the other participants and how to contact each of them (addresses). There could be Complete or Partial or No Accessibility</td>
</tr>
<tr>
<td>Inclusiveness:</td>
<td>A specification whether the number of participating agents is fixed or variable at the start of the interaction</td>
</tr>
</tbody>
</table>
representation of information in each field are specified in this component of the protocol structure.

3.2.1.5. Message Exchange Rules

This component of the protocol structure defines the characteristic behaviour of the interaction. This presents a specification of the different guidelines that direct how messages are exchanged in order to efficiently and effectively realize the interaction. The specification includes how an interaction should be initiated, how the interaction should end, message exchange mode, timing constraints between receiving and sending of messages.

3.2.2. Organization of the Agent Interaction Protocol Components

The different components of the protocol structure are connected to one another as shown in Figure 3.1. The relationships among the components of an agent interaction protocol show how one component influences the definition of another component. These connections provide a better conceptualization of the agent interaction protocol structure.

Figure 3.1: Agent Interaction Protocol Structure
The Purpose component determines the content of the Performatives, Message Structure and Message Exchange Rules components. The Purpose of the interaction dictates the number and type of performatives required to achieve such an interaction. The message structure is affected by the purpose as some of the information to be included or not included in the message structure will be determined by the purpose of the interaction. For instance, a purchase interaction between a buyer and a seller requires settlement details in one of the messages while an advertisement interaction between a seller and potential buyer may or may not include only modes of payment and not settlement details. The major determinant of the message exchange rules is the purpose component since the purpose details what the interaction seeks to achieve, how critical it is to the system, how quickly the interaction should happen, how it should end, how it should handle errors in the interaction.

The Players specification influences the performatives and the message exchange rules since some of the players may have overriding roles in the interaction, hence considerations will be given to such roles where they exist, in defining the performatives and the message exchange rules. Performatives affect message structure since a performative gives meaning to a message. Also, message exchange rules in some cases influence the message structure, an example is where a message exchange rule states that ‘the interaction initiating message must state the purpose of the message’ (where this is not part of the performative’s semantic meaning), the structure for such a message will therefore include a field for this information.

It is necessary to specify these details explicitly because it helps in achieving uniform protocol interpretation and also in appropriate interaction error handling. For instance, if a responder is sending a \textit{cfp} message, which has been declared in the Performatives component to be an initiator performative or if a performative that is defined to be used only once is being used a second time within the same interaction, it indicates a high likelihood of an error or an exception in the interaction being executed.
3.3  Agent Interaction Protocol Property Suite

The preceding section presents the agent interaction protocol structure by defining its components and their arrangement. This section describes the agent interaction protocol property suite and presents the relationship between the Agent Interaction Protocol Structure and Agent Interaction Protocol Property Suite.

3.3.1. Agent Interaction Protocol Property Suite

The properties of a protocol are the features or attributes that define the protocol’s identity. Each of these properties has more than one possible value. The set of values of the properties applicable to an agent interaction protocol make up the Protocol Property Suite. As the interaction is analyzed in the context of the domain, the properties that are applicable to an agent interaction protocol to be designed are identified. The values of the identified properties make up the protocol property suite for this particular agent interaction protocol. Table 3.3 describes the elements Protocol Property Suite. These elements may not be exhaustive, however, it is believed that they capture the fundamental properties of the Protocol Property Suite.

An agent interaction protocol $P_A$ becomes another protocol $P_B$ when the values of its properties are changed because the properties of a protocol define the protocol’s components and therefore its function and identity. It is therefore necessary to separate the protocol property suite from the protocol structure so as to facilitate the process of modifying and upgrading a protocol and making another protocol out of an existing one.

The following example is presented to illustrate some of the concepts introduced here and the claim that two agent interaction protocols with the same protocol structure will differ in function by their properties:
### Table 3.3: Elements of the Protocol Property Suite

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timing Constraint:</td>
<td>A description of the impact of time in achieving the interaction objective e.g. bid submission is deadline constrained</td>
</tr>
<tr>
<td>Security Concerns:</td>
<td>A specification of the impact of this interaction on the security concerns of the system (e.g. confidentiality of information exchanged)</td>
</tr>
<tr>
<td>System Safety:</td>
<td>A specification of the impact of this interaction on the physical safety of the system</td>
</tr>
<tr>
<td>Error Sensitivity:</td>
<td>A description of the sort of error (content and control) that the interaction can cope with e.g. high error sensitivity in air traffic control related interaction</td>
</tr>
<tr>
<td>Messaging Mode:</td>
<td>A specification of the message sequencing mode i.e. Asynchronous or Synchronous</td>
</tr>
<tr>
<td>Messaging Mechanism:</td>
<td>A specification of method of sending messages to the intended recipients i.e. broadcast, multicast</td>
</tr>
<tr>
<td>Interaction Mode:</td>
<td>A specification of the number of agents that an agent can simultaneously connect with for message exchange i.e. Multilateral, Bilateral</td>
</tr>
<tr>
<td>Ontology Matching:</td>
<td>A specification of the uniformity or otherwise of the participating agents’ representations of the real world in a heterogeneous environment</td>
</tr>
</tbody>
</table>

Consider two agent interaction protocols $P_A$ and $P_B$. They both have the same set of performatives *ask*, *tell* and *end* with the message sequence in the order *ask* – *tell* – *end* (an example of a subset of their protocol structure). A property *timing sensitivity* with values *False* for $P_A$ and *True* for $P_B$ (the values *False* and *True* for property *timing sensitivity* represent the protocol property suite) will differentiate the behaviour of the protocols by defining the following message exchange rules for the two protocols.

$P_A$: An agent A that sends the message *ask* to another agent B does not send the message *end* to close the interaction until it receives the message *tell* within a time interval of 0 – 300 seconds, after which it may close the interaction with the message *end*.  

62
PB: If an Agent A does not receive a message *tell* within a time interval of 0 – 5 seconds after sending the message *ask* to an agent B, A should send the message *ask* to another agent C. If A receives a *tell* message from either of B or C within 5 seconds of sending to C, A closes both interactions by sending *end* messages to B and C, otherwise it closes interaction with B alone.

This illustration is a clear instance of how the values of a property differentiate the functions of two agent interaction protocols with the same structure. Agent interaction protocol $P_B$ represents a time critical system, while $P_A$ represents a system that is not time critical. The essence of expressiveness is evident by this illustration as the level of details in specifying the protocols will reduce ambiguity in the protocol interpretation.

3.3.2. Relationship between Protocol Structure and Protocol Property Suite

The protocol structure defines the components of an agent interaction protocol and the organization of these components for a particular protocol. The property suite defines the properties of an agent interaction protocol and the values of these properties for a particular protocol. The relationship between the components of the protocol structure and the protocol property suite is as follows.

The first two components of the Protocol Structure, that is, the Purpose and Players components provide *input* into the definition of the Protocol Property Suite as well as to the definition of the other components of the Protocol Structure, which are the Performatives, Message Exchange Rules and Message Structure. The Protocol Property Suite is used to *refine* the Performatives, Message Exchange Rules, and Message structure components of the Protocol Structure. Figure 3.2 shows these relationships among the different components of the Protocol Structure and the Protocol Property Suite.
3.4 Dedicated Approach for Developing Agent Interaction Protocols

Sections 3.2 and 3.3 describe the agent interaction protocol structure and agent interaction protocol property suite respectively. This section presents the dedicated approach for developing agent interaction protocols. This dedicated approach describes how to define the components of an agent interaction protocol structure and the values of an agent interaction protocol property suite using domain analysis.

The dedicated approach for developing agent interaction protocols is made up of two phases. These are the Domain-directed Analysis phase and the Design/Verification phase. These phases are described in the following subsections.

3.4.1. Domain-directed Analysis Phase

The contexts and attributes of a multiagent system’s application domain determine the properties of the agent interaction protocols that are required for effective and efficient interactions within the multiagent system. Issues such as time criticality, safety criticality, concurrency, control hierarchy, and goal diversity are domain
dependent and these all influence the way and manner in which effective interactions in multiagent systems should occur. Analyzing a multiagent system’s application domain to determine its peculiar features is essential to the success of the interactions in the multiagent system. Therefore, domain analysis is a primary driver of the dedicated approach for the agent interaction protocol development process.

This phase of the dedicated approach involves two activities. The first activity is the analysis of the system which is carried out in the context of the application domain. This activity obtains the properties of the agent interaction protocol to be developed by analyzing the characteristics of the application problem in the context of the application domain. Identity is given to the protocol developed and its appropriate applicability is assured when its requirements are analyzed in the context of the domain. Domain-directed analysis is carried out using the following process:

1. At the completion of the multiagent systems requirements analysis, identify the system goals that require more than one agent to achieve them.
2. For system goals that require more than one agent to fulfill them, define if the agents need to interact with one another or with external sources in order to achieve the goal.
3. For system goals that are achievable by only one agent, define if the agent requires information or assistance (resources) from other sources in order to achieve the goal.
4. Where interactions are identified from steps 2 and 3 above, analyze the goal and the application problem in the context of the application domain in order to define the elements of the Purpose component (section 3.2.1.1) for each interaction required.
5. Identify the agents that are required for each interaction.
6. Analyze the goal, the application problem in the context of the application domain and the agents involved in each interaction in order to define the elements of the Players component (section 3.2.1.2) for each interaction.
7. Using the analysis of the application problem in the context of the application domain and inputs from the Purpose and the Players component, define the Protocol Property Suite.

The outcomes of this phase of the engineering process are the specifications of the Purpose and Players components and the Protocol Property Suite of our protocol definition. These components present a detailed and structured representation of the analysis in the context of the domain such that it can be effectively translated into design with very minimal ambiguity.

In [110], an approach for carrying out domain modeling is described. This approach called Application-based Domain Modeling approach (ADOM) could be used for the analysis of the characteristics of the application problem in the context of the application domain in order to generate a model of the domain (reference model). This model can be used for specifying the Purpose and Player components and the Protocol Property Suite of an interaction protocol being designed.

The second activity of this phase of the dedicated approach is the search for existing agent interaction protocols. This is separately represented and emphasized to show our recognition of the existing body of work on agent interaction protocol specifications and to emphasize our consideration for reuse. The specifications generated from the analysis are used as a basis to search for an existing agent interaction protocol that is most suited to the agent interaction protocol to be developed. The outcome of the directed search could be an existing agent interaction protocol that suits the intended agent interaction protocol, an existing agent interaction protocol that needs to be modified to suit the intended agent interaction protocol or no existing agent interaction protocol that is close to the intended agent interaction protocol, hence requiring design from scratch.
3.4.2. Design/Verification Phase

The design phase of our agent interaction protocol development approach has two possible paths depending on the outcome of the directed search in the analysis phase. Where a similar existing agent interaction protocol is found, the reuse path is taken. If no similar agent interaction protocol is found, the develop path is followed. Where an agent interaction protocol that matches the intended agent interaction protocol is found after the directed search, the process proceeds to the Verification phase, Figure 3.3.

Figure 3.3: Proposed agent interaction protocol engineering process

Where the design follows the reuse path, the first stage in reuse is to determine if the existing agent interaction protocol can be modified to make it fit into the requirements of the intended agent interaction protocol. A major consideration at this stage is to assess the cost of modifying in terms of time and effort, against that of developing the protocol from scratch. Some of the things to consider in
determining if an existing protocol can be readily modified include method of specification, understanding of the heuristics or logic behind the design, and complexity of the protocol. If it is determined that this existing protocol can be readily modified, the next stage along this design path is to customize the protocol using the specifications from the analysis phase.

Where there is no existing agent interaction protocol that matches the intended agent interaction protocol, or the existing agent interaction protocols are not customizable, the develop path is followed i.e. the intended agent interaction protocol is designed from scratch. The develop path of the design phase starts by defining the set of performatives to be used by the agent interaction protocol and their semantics using the specifications from the analysis phase. Then the message exchange rules are defined and a model of the message structure for each of the performatives is also defined. Subsequently, the protocol is graphically specified. Here, we propose the use of Scenario-Based Programming (SBP) for graphically specifying the protocol. Scenario-Based Programming is based on the formal language of Live Sequence Charts [38]. SBP representation of an agent interaction protocol creates expressive specifications of agent interaction protocols.

Verification of the agent interaction protocol developed (specified or modified) and the unmodified existing agent interaction protocol is the final stage of the design/verification phase of our agent interaction protocol engineering process. The verification process is dependent on the method used in specifying the agent interaction protocol. SBP has automated techniques for carrying out the verification of the accuracy of the specified agent interaction protocols. The design/verification phase is iterative. It is repeated until the verification proves that the protocol has been properly specified.

The implementation of agent interaction protocols is not captured in the dedicated approach for developing agent interaction protocols presented here. Different techniques are used for the implementation of agent interaction protocols in the development of multiagent systems. Some of the factors resulting in variations in
the implementation of agent interaction protocols include differences in agent-oriented software engineering methodology, programming technology, type of agents and multiagent system, differences in agent communication languages and the complexity of the interactions in the multiagent systems. The harmonization of these factors for the definition of a strategy for the implementation of agent interaction protocols to be incorporated in the dedicated approach is beyond the scope of this thesis.

3.4.3. Process and Product

The relationship between the Protocol Structure components, the Protocol Property Suite and the processes involved in the dedicated approach for developing agent interaction protocols is shown in Figure 3.4. The analysis phase of the dedicated approach produces the Purpose and Players components of the Protocol Structure and the Protocol Property Suite. The design/verification phase generates the Performatives, Message Structure and Message Exchange Rules.

![Figure 3.4: The Process and The Product](image-url)
3.5 Example

This section illustrates the use of the Protocol Structure components in describing the domain analysis for developing an agent interaction protocol. The agent interaction protocol used for this illustration is the Provisional Agreement Protocol (PAP) for Global Transportation Scheduling [106, 105]. This protocol was developed for interaction in military operations transportation scheduling. According to [106], a typical military transportation operation is to move large quantities of resources on a global scale. As a result, a transportation operation may require the services of many transportation organizations. Each of these transportation organizations is usually only capable of moving a portion of the quantity of the resource through only a portion of the distance to be covered. The domain is open and dynamic. Transportation organizations enter and leave the system at will with the possibility of their capabilities continually changing.

The specifications of the Purpose, Players and Protocol Property Suite components of the PAP are presented in the following tables 3.4, 3.5, and 3.6 respectively:

<table>
<thead>
<tr>
<th>Table 3.4: PAP Purpose Component</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interaction:</strong></td>
</tr>
<tr>
<td><strong>Related System Goal:</strong></td>
</tr>
<tr>
<td><strong>Domain of Application:</strong></td>
</tr>
<tr>
<td><strong>Domain Type:</strong></td>
</tr>
<tr>
<td><strong>Interaction Objective:</strong></td>
</tr>
<tr>
<td><strong>Interaction Type:</strong></td>
</tr>
<tr>
<td><strong>System Safety:</strong></td>
</tr>
<tr>
<td><strong>Pre conditions:</strong></td>
</tr>
<tr>
<td><strong>Post conditions:</strong></td>
</tr>
</tbody>
</table>
Table 3.5: PAP Players Component

| Interacting Agents:       | Manager Agents – Military Organization  
|                          | Transportation Agents – Transportation Organizations |
| Initiator:               | Manager Agent                                |
| Responder(s):            | Transport Agents                              |
| Inter-agent Relationships: | Client / Service Provider                   |
| Privileges:              | None                                          |
| Number of Agents:        | Greater than two                              |
| Diversity of Agents:     | Heterogeneous                                 |
| Distribution:            | One-to-Many                                   |
| Accessibility:           | Complete                                      |
| Inclusiveness:           | Variable                                      |

Table 3.6: PAP Protocol Property Suite component

| Timing Constraint:       | Time sensitive. Bids are deadline driven      |
| Security Concerns:       | Low                                           |
| Error Sensitivity:       | High                                          |
| Messaging Mode           | Asynchronous                                  |
| Messaging Mechanism:     | Broadcast                                     |
| Interaction Mode         | Multilateral                                  |
| Ontology Matching:       | Uniform                                       |

3.6 Chapter Summary

Software realizing agent interaction protocols is a peculiar kind of software as it serves as the software infrastructure for interacting agents in a system that seeks to closely model real world interactions. The behaviour of the real world system being modeled is represented and implemented by the agent interaction protocol. Agent interaction protocols are different from communication protocols as agent interaction protocols bring contextual dimension into the interactions they
implement instead of transporting data packets. The context of an application domain is a fundamental consideration in conceptualizing and developing agent interaction protocols.

Most of the existing works on agent interaction protocols focus on either design or implementation without a dedicated approach for developing agent interaction protocols. Also, the crucial aspect of application domain analysis is not given the attention it demands. As a result, existing agent interaction protocols, which may not necessarily be appropriate in modeling the particular system interactions, are plugged into multiagent systems development projects. To achieve good software quality, dedicated development approaches are required for the aspect of the software being developed [59], in this case, agent interaction protocols.

The aim of this Chapter is to present a well defined process for the design and development of agent interaction protocols that facilitates productive reusability, based on the agent interaction protocol structure and property suite. This Chapter presents a dedicated approach for developing agent interaction protocols. This approach, which specifies the Analysis and Design/Verification phases of the development process, is driven by the analysis of the application domain as they affect interaction. This Chapter presents an agent interaction protocol structure and an agent interaction protocol property suite which form the bases for the activities in the dedicated approach.

The next three Chapters present the contribution of this thesis towards addressing the issues with agent-oriented patterns in order to enhance the comprehension and adoption of agent technology.
Chapter 4

A Classification Scheme for Agent-Oriented Patterns

4.1 Introduction

Chapter Two identified two approaches to addressing the challenges facing agent-oriented software engineering and the adoption of agent technology. The first approach is analysis and structuring of the interactions in multiagent systems. The issues with this approach and how these issues impact on the adoption of agent technology were discussed in Chapter Two. In Chapter Three, a dedicated approach for developing agent interaction protocols was presented to address the issues with analysis and structuring of interactions in multiagent systems.

The second approach to addressing the challenges facing agent-oriented software engineering and the adoption of agent technology is sharing agent-oriented software engineering experiences. The issues limiting the potential of this approach from enhancing the adoption of agent technology were discussed in section 2.3.5 of Chapter Two and can be summarized as follows:

- There are either no rationale for the design of agent-oriented pattern classification schemes or rationale provided by different authors for the same type of agent-oriented patterns are inconsistent
- A good number of the classifications are not comprehensive. That is, they are not designed to capture all the different phases and processes of agent-oriented software engineering.
This Chapter presents a classification scheme to address the issues relating to the classification of agent-oriented patterns. The levels of abstractions in agent-oriented software engineering serve as the basis for the design of the classification scheme. Also, in this Chapter, the properties of the levels and categories of the Two Way classification scheme are defined. These properties represent the attributes that are expected of the patterns that fall into these levels and categories and therefore represent a tool for the analysis of agent-oriented patterns. The analysis of agent-oriented patterns using level and category attributes is presented in Chapter 5. The level and category attributes are used in Chapter 6 for the design of agent-oriented pattern template structure and derivation of description templates for agent-oriented patterns.

Section 4.2 of this Chapter presents principles and structure of the classification scheme. Section 4.3 describes the general phases of software development and the phases of software development in two agent-oriented software engineering methodologies, ROADMAP and Prometheus. Based on the analysis of the phases of software development presented in section 4.3, section 4.4 defines the levels of abstractions of the classification scheme. Section 4.5 defines the categories of the classification scheme according to the software development tasks that are associated with each of the levels of abstractions. In section 4.6, the dimensions of the Two Way classification scheme are combined to give the Two Way classification scheme. Agent-oriented pattern attributes are presented in section 4.7. In this section, the definition and significance of agent-oriented pattern attributes are discussed and the level and category attributes of the vertical and horizontal dimensions of the Two Way classification scheme are described. Section 4.8 is the Chapter summary and conclusion.

4.2 Principles and Structure of the Classification Scheme

The concepts of agent technology which include autonomy, proactive, reactive, social behaviours, adaptability, agents among others, differ from those of traditional software development paradigms (see Chapter 2). In addition, the
concepts of agent technology that are predominant at different phases of
development of multiagent systems differ across the different levels of abstractions
in agent-oriented software engineering. For instance, whereas the emphasis is on
goals and organization of roles at the early stages of analysis and design, the design
of concepts such as autonomy and proactive behaviour become the important tasks
at the later stages of design. Also, the development of multiagent system
experiences the transformation of concepts of agent technology across the levels of
abstractions. For instance, roles at the analysis stage are transformed to agents at
the design stage; roles’ responsibilities become agents’ tasks across the levels. The
various concepts and the relationships among them generate different agent-
oriented software engineering problems for which agent-oriented patterns are
written by a variety of agent technology researchers and practitioners. The
challenge therefore is to identify how the concepts of agent technology and levels
of abstractions in agent-oriented software engineering will be reflected in the
classification of agent-oriented patterns.

This section presents the principles that guide the design and the structure of the
classification scheme

4.2.1. Principles of the Classification Scheme

The classification scheme presented in this chapter is based on the following
principles. These principles are derived from the analyses of the existing work in
agent-oriented patterns (section 2.3.5).

A classification scheme for agent-oriented patterns should be structured according
to the two dimensions of agent-oriented software development, that is, the
classification scheme should be modeled according to the levels of abstractions and
the software development tasks at each of those levels of abstractions.

A classification scheme should be comprehensive but concise. It should be
comprehensive, such that it captures the major categories of patterns. It should be
concise, such that it does not have too many dimensions or categories that may cause confusion or result in individual interpretations.

The classification scheme should be problem driven. The design of the classification scheme should be design problem based. That is, the classification scheme should be structured to logically categorize the types of problems that can be encountered in the development multiagent system. The justification for this principle of agent pattern classification scheme design is as follows. There are two main parts to a pattern which are the problem to be solved and the solution to the problem. Since what a potential multiagent system developer encounters first is the problem after which a solution to the problem is sought for, it follows that the guide to the solution should be based on the problem which is what the potential developer is familiar with. Therefore, the classification scheme should be designed to group the patterns that solve a particular category of problems together.

4.2.2. Structure of the Classification Scheme

Based on the principles presented above, the classification of agent-oriented patterns is based on the definition of the levels of abstractions and the software development tasks/ concepts of agent technology at each level of abstraction. The categorization of agent-oriented patterns should therefore be structured to clearly, comprehensively and concisely capture these differences in levels of abstractions, as well as agent concepts and development tasks at these levels of abstractions. From the foregoing, this thesis presents a Two Way classification scheme for the classification of agent-oriented patterns. By way of definition:

The Two Way classification scheme is an agent-oriented pattern classification that defines two dimensions for classifying agent-oriented patterns. These dimensions are based on the levels of abstractions in agent-oriented software engineering and the software development tasks at each of these levels of abstractions.
An agent-oriented pattern’s category in the Two Way classification scheme is defined by the level of abstraction and the type of software development task that the pattern’s problem element relates to. The levels of abstractions dimension is referred to as the Vertical dimension. The software development tasks dimension is referred to as the Horizontal dimension.

These two dimensions ensure that agent-oriented patterns can be classified relative to the concepts of agent technology that they describe as well as the software development task and level of abstraction at which they are relevant. Table 4.1 is a representation of the Two Way classification scheme’s structure showing the two dimensions and a Category C in the classification scheme that is defined by a combination of the $level_2$ and $task_3$.

<table>
<thead>
<tr>
<th>Software Development Tasks – Horizontal Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
</tr>
<tr>
<td>Level 1</td>
</tr>
<tr>
<td>Level 2</td>
</tr>
<tr>
<td>Level 3</td>
</tr>
</tbody>
</table>

The next section describes the basis for defining the levels of abstraction (vertical dimension) of the Two Way classification scheme.

### 4.3 Basis of the Classification Scheme

This section discusses the general phases of software development and studies the phases of development in two agent-oriented software engineering methodologies. This serves as a basis for specifying the levels of abstractions of the classification scheme in section 4.4.
4.3.1. General Phases of Software Development

There are certain phases of development that are general to building software systems regardless of the conceptual modeling/implementation paradigm. Engineering any type of software system involves five main activities which are defining the problem to be solved; analyzing the problem to create a platform for design; designing an appropriate software system on the basis of the analysis of the problem; implementing the design using some programming technology and testing the implemented software system to ensure that it actually does what it is intended to do and that it is coded to achieve user requirements. All programming paradigms and methodologies generate techniques, phases and processes that center around these five main activities of developing software systems. Difficulties in the comprehension of a software engineering methodology occur, where the methodology does not define its processes and techniques to correspond to these five main phases of software development.

The phases of software development life cycle in agent-oriented software engineering should also draw from the general lifecycle phases of software engineering. It is expected that the specific processes and techniques of agent-oriented software engineering methodologies will be different from those of traditional software engineering methodologies. However, it should be possible to relate the software development phases of the different agent-oriented software engineering methodologies to these five general phases of software development. This lack of correlation between the phases of some agent-oriented software engineering methodologies and the general phases of software development is one of the main difficulties associated with the comprehension of agent technology and its application.
Figure 4.1: General Phases of Software Development

The following two sections reviews ROADMAP [58, 57, 73] and Prometheus [97] agent-oriented software engineering methodologies to assess the levels of abstractions they present and the concepts of agent technology they feature at these levels of abstractions. ROADMAP and Prometheus are chosen for this review because they feature phases of agent-oriented software engineering that could be mapped to some of the general phases of software development. These reviews are done towards the identification of the relatively general levels of abstractions in agent-oriented software engineering that correspond to the general phases of software development discussed in this section.

4.3.2. ROADMAP

ROADMAP methodology is an architecture independent agent-oriented software engineering methodology. It is not based on a particular agent internal architecture.

ROADMAP presents models for the analysis and design phases of agent-oriented software engineering. The ROADMAP models are divided vertically into three sections. These areas are Domain specific models, Application specific models and Reusable services models (see Figure 4.2). The Domain specific models are the Environment and Knowledge models. These models capture information about a
specific domain. They do not relate to any of this methodology’s four phases of software development in particular.

Figure 4.2: ROADMAP Models

The Application specific models relate to the software system being developed. They define the main features of the system being modeled. Application specific models include the Goal, Role, Agent and Interaction models.

The Reusable services models include the Social and Service models. These models capture the reusable functionalities and organizational structures of the entities in the system.

ROADMAP methodology features four levels of abstractions. These are Analysis, Design, Detailed Design and Implementation and Testing. The models under the Application specific and Reusable services areas of the methodology relate to the analysis and design phases of software development. The Environment and Knowledge models under the Domain specific models are not assigned to these two levels of abstractions. The models that relate to the Analysis phase are the Goal and
Role models under the Application specific models and the Social model of the Reusable services models. The Agent and Interaction models of the Application specific models as well as the Services model of the Reusable services models relate to the Design phase.

ROADMAP does not specify models/techniques/processes for detailed design and implementation and testing. The four levels of abstractions of the ROADMAP methodology can be indirectly related to four of the five general phases of software development discussed in the previous section. These general phases are analysis, design, implementation and testing.

The differences in the concepts of agent technology featured by ROADMAP at the different levels exemplify the transformation of the concepts across the levels of abstractions. The Role and Goal models at the analysis phase are not dropped at the design phase, rather, the Roles translate to agents and the Goals are represented by plans in individual agents.

4.3.3. Prometheus

Prometheus [98, 97] was developed with a goal of having an agent-oriented software engineering process with associated deliverables that can be easily followed to develop intelligent agent systems.

The Prometheus agent-oriented software engineering methodology based on the BDI agent architecture. The BDI agent architecture models an agent as having mental attitudes of Beliefs, Desires and Intentions [109, 107, 8]. These mental attitudes determine the behaviour of the agent. The methodology is divided into three phases which are system specification, architectural design and detailed design phases (see figure 4.3).

At the System specification phase, the environment of the system, the goals and functionality of the system are identified. The environment of the system is identified by the information coming into the system from the environment.
(percepts), the means by which the system influences its environment (actions) and any shared data sources. Goals and functionality of the system are defined by describing how the functionalities of the system achieve its goals. These descriptions of functionalities are captured by use cases.

At the Architectural design phase, the outputs of the system specification phase are used to carry out three main activities. These are definition of agent types, design of the overall system structure and definition of the interactions among the agents in the system. Agent types are defined according to the functionalities identified at the system specification phase. The interactions among the agents are based on the scenarios described at the system specification phase and they are documented as interaction diagrams. The system structure defines the arrangement and operation of the system as a whole. The system structure captures all the agents in the system and how they are related with one another and with data sources.

Figure 4.3: The Prometheus Methodology
At the Detailed design phase, the internals of each agent as well as how each agent will achieve its goals within the system are described. The capabilities, internal events, plans and detailed data structures of each agent are defined at this phase of the methodology. It is designed mainly for engineering closed systems.

The three levels of abstractions of the Prometheus methodology can be indirectly related to two of the five general phases of software development discussed in the previous section. These general phases are analysis and design.

At the analysis phase, ROADMAP specifies goal and role models and the relationships among roles in social models. Goal models are defined to depict the break down of goals into sub goals and the connection of roles to goals. This phase of the methodology therefore addresses the analysis of the system to be developed in terms of goals and roles and the relationships among these concepts. The concepts of agent technology are introduced at this early stage of the development process.

The architectural design phase of Prometheus focuses on the definition of the structure of the multiagent system and the interactions that will exist within the system as shown by the System Overview deliverable. Interaction model in ROADMAP is also defined at the design phase. The activities of interaction design and system structure designs are a logical progression in software development from the analysis phase.

The detailed design phase of Prometheus focuses on the specification of the internal structure of each individual agent in the system. This phase is the last phase of the Prometheus methodology. ROADMAP on the other hand specifies two more phases which are Detailed Design and Implementation & Testing phases. However, no models are specified for these phases. Prometheus features the System Overview deliverable at the Architectural design level and Agent Overview at the Detailed Design level. The System Overview and Agent Overview represent multiagent system design and the design of the internal architecture of an agent.
respectively. Since Prometheus is both architecture and programming technology specific, it is easy to map the detailed design models to JACK programming language constructs. Hence, the Implementation phase of Prometheus is not explicitly separated from the detailed design. This will not be the case if the methodology is not programming technology specific, hence, it will be important to clearly specify the implementation phase of the methodology.

4.4 Vertical Dimension of the Classification Scheme

In this section, the reviews of ROADMAP and Prometheus in the context of the general phases of software development presented in section 4.3 is used as a basis for defining the levels of abstractions of the Two Way classification scheme.

The levels of abstractions of the Two Way classification scheme are defined by comparing the concepts of agent technology that are relevant to the different phases of software development in ROADMAP and Prometheus (the agent-oriented software engineering methodologies reviewed in section 4.3). This comparison gives an insight into the concepts of agent technology that are critical to the engineering of multiagent systems at the different levels of abstractions, in the context of these two methodologies. The concepts that are critical at the different levels according to this comparison are put together as follows to specify the general levels of abstractions.

At the Analysis level, according to both methodologies, the focus is on the definition of Goals, Roles, Role relationships and Environment and Knowledge even though different descriptors are used by the methodologies (see section 4.2). The Design level (ROADMAP) or Architectural Design level (Prometheus) focuses on defining/designing Agents from the Roles and designing the Interactions. Also emphasized at this level, is the System Overview which is the overall architecture of the agents and other entities in the system. The Detailed Design level (Prometheus) focuses on the design of the internal architecture of the each agent in the multiagent system being developed.
From this comparison, three levels of abstractions in agent-oriented software engineering are identified. These are Agent-Oriented Analysis level, Multiagent System Architecture level, and Agent Internal Architecture level. Four more levels are added to the levels of abstractions in agent-oriented software engineering as follows. Before the Agent-Oriented Analysis level, Problem Definition level is added. After, the Agent Internal Architecture level, three more levels are added. These are the Detailed Design level, the Implementation level and the Testing level. The Detailed Design level focuses on the translation of the Multiagent System Architecture and Agent Internal Architecture designs into the design constructs of the programming technology that will be used to implement a multiagent system. The Implementation level is the level for coding the multiagent system in the programming language of the programming technology of implementation. The Testing level focuses on the testing of the implemented multiagent system. These four levels are added to give a comprehensive view of the process, of engineering multiagent systems, that corresponds to the general phases of software development. Figure 4.4 is a diagram showing the seven identified general levels of abstractions in agent-oriented software engineering.

![Figure 4.4: 7 Levels of Abstractions in Multiagent System Development](image-url)
The seven levels of abstractions described above are compressed into four levels for the purpose of the design of the classification scheme for agent-oriented patterns. The four levels derived are the levels of abstractions where the concepts of agent technology are featured, hence, where agent-oriented patterns are applicable. However, deriving them from the seven levels allows these four levels to be perceived in the context of the general levels of software development. The first three of these levels of abstractions are the Agent-Oriented Analysis level, the Multiagent System Architecture level, and the Agent Internal Architecture level which correspond to the second to the fourth of the seven levels. The Problem Definition level is included in the Agent-Oriented Analysis level. The Multiagent System Realization level is a combination of the Detailed Design, the Implementation and the Testing levels of the seven general levels (see figure 4.5).

![Figure 4.5: 4 Levels of Abstractions in Multiagent System Development](image)

The detailed descriptions of the four levels of the classification scheme are presented in the following subsections. A diagram of these four levels of abstractions and the software engineering activities/deliverables at each of the levels is presented in figure 4.6.
The characteristics of an agent introduces fundamental but subtle differences to the way agent based systems are conceptualized, that is analyzed and designed. Since the concepts of autonomy, social behaviour, proactive and reactive behaviours are essential to the definition of an agent, an agent based system is conceptualized in terms of roles, goals, interactions and relationships rather than in terms of functional requirements (or functionalities). The functionalities of the system are defined as the responsibilities of the roles in the system.

![Figure 4.6: Levels of Abstractions of the Two Way Classification Scheme](image-url)
Another fundamental difference about agent-oriented software engineering is the fact that concepts of agent technology transforms or changes as the development progresses from one level to another. For instance, goals will translate to responsibilities at the next level of analysis and responsibilities at the analysis level will translate to actions and plans at the agent architecture level. Roles will translate to agents and the agents will be designed in terms of the features of autonomy, proactive and reactive and social behaviours. Coordination, control and interaction at the analysis level will translate into agent interaction protocols at the multiagent system architecture level. Organization of the roles and agents at the analysis level will translate to structural arrangement (architectures) at the multiagent system architecture levels.

This difference in concepts at the levels of abstractions requires the separation of analysis from design in agent-oriented software engineering. According to [73] separating analysis from design in agent-oriented software engineering allows for adequate reasoning about the behaviour of the real life system at the analysis level before introducing design concepts that will appropriately achieve this behaviour in the multiagent system to be developed. Though these levels differ in concepts, there should be traceability in the analysis and design across the levels in order to avoid the problem of ad hoc and unconnected implementation of multiagent system as identified by [117]. In other words, it is essential to address the development tasks at the individual levels of abstractions in the context of the concepts of agent technology applicable at that level while ensuring a coherent design across the levels.

The importance of these fundamental differences is that it changes the way agent based systems are analyzed and designed from one level of abstraction to the other. This paradigm shift is factored in to the specification and description of the levels of abstractions in the following sub sections.
4.4.1. Agent-Oriented Analysis Level

At the Agent-Oriented Analysis level, Goal/Role analysis, that is, analysis of requirements specification is carried out. Goal/Role analysis involves the definition of the goals, the sub goals, the quality goals of the system and the definition of the roles within the system. The responsibilities of the roles in the system are defined as part of this analysis. Roles identified at this level include human roles as well as software system roles.

The connection between the roles, the goals and the relationships among the roles are also defined by Goal/Role analysis. Possible role-role relationships include collaboration, control or competition relationships. Collaboration relationship is the relationship that exists among a set of roles that are working together to realize a common (super) goal. An example of this kind of relationship is a team structure. For instance, a team of defense staffs collaborating to conquer enemy forces or a team of production line workers collaborating to manufacture a product.

A control relationship defines a supervisor – subordinate kind of relationship. For example, the relationship between a commanding officer and a troop member; the relationship between a production line supervisor and production line worker. Competition relationship is the relationship that exists amongst a set of roles where each role has a goal that is inconsistent with the goal of another role in the system. The relationship between these roles is towards the achievement of their separate goals. For example, the relationship between two roles negotiating in a Dutch auction is a competition relationship.

The interactions amongst the roles are also defined at this level. The kind of relationship determines the kind of interaction that occurs amongst the roles in the system. Therefore, the analysis of the requirements specification is done to define interactions among the roles in a manner that reflect the relationships already defined for these roles.
The software development tasks at the Agent-Oriented Analysis level can be summarized as identification of roles, identification of goals, identification of the relationship among the roles (role organization), and the interactions among the roles.

4.4.2. Multiagent System Architecture Level

The Multiagent System Architecture level follows the Agent-Oriented Analysis level. It is the first level in the design of multiagent systems. The focus of this level of abstraction is the design of the overall structural layout of the multiagent system being developed.

The first task at the multiagent system architecture is the specification of the agents to represent the roles in the system. Agents are the software abstractions of the roles identified during the requirements analysis. Therefore, roles are translated into agents. The translation of roles to agents will be determined by the responsibilities of the roles and the goals that the roles are meant to achieve. Some roles in the system have ‘heavyweight’ goal(s) to achieve or responsibilities to fulfill. Such roles can be translated into more than one agent. Two roles which have ‘lightweight’ and compatible goal(s) to achieve can be translated into a single agent. Therefore, a single role may be decomposed into more than one agent while certain multiple roles could be aggregated into one agent.

The definition of the relationships among the agents in the system to create a virtual organization based on the role relationships already analyzed is another task that is carried out at this level. Once agents are defined to represent all the roles in the system, the relationships as well as interactions among the agents need to be modeled. Where certain roles in the system have been decomposed or aggregated, the relationship and interaction among the resulting agents need to be carefully modeled in order to accurately reflect the relationship and interaction in the analyzed system. For instance, consider a hypothetical system of three roles $R_A$, $R_B$, and $R_C$. Role $R_C$ relates to roles $R_A$ and $R_B$ for the services $ca$ and $cb$ respectively.
(Figure 4.7(a)). In the process of translation of roles to agents, role \( R_C \) gets translated to agent \( A_C \) and roles \( R_A \) and \( R_B \) are combined to form one agent \( A_{AB} \) (Figure 4.7(b)), the interactions and relationships among agents \( A_C \) and \( A_{AB} \) in the system will become different (Figure 4.7(c)) from the interactions and relationships among the roles \( R_A, R_B, \) and \( R_C \) that these agents represent. Such differences have to be captured and factored into the modeling of the interactions and relationships among the agents in order to maintain the behaviour of the real life system being modeled.

![Diagram](image)

**Figure 4.7(a)**  
**Figure 4.7(b)**

![Diagram](image)

**Figure 4.7(c)**

**Figure 4.7: Definition of Agent-Agent Relationships**

Another task at the multiagent system Architecture level is the design of a feasible and efficient software system structure from the agents and agent relationships. That is, the organization of the components of the multiagent system (agents, environment) into logical units that will guide the implementation of the multiagent system. Multiagent system architecture is driven by considerations such as the quality goals (Non Functional Requirements) of the system and efficient arrangement of the agents in the system as well as other software entities necessary to make the software system work. Examples of such other software entities
include knowledge repositories, communication infrastructure, representations of the environment (websites), legacy systems, perception and action sub systems. Multiagent system architecture is the first step in bridging the gap between the role organizations which are models of the real life organization and the actual software system.

Other tasks at this level include defining the interfaces with the environment, legacy systems and data/information sources; defining the interaction mechanisms (agent interaction protocols) and data communication infrastructure. At this level is the design of the interactions in the multiagent system and the interaction/communication infrastructures that are required for efficient communication in the developed software system. Agent Interaction Protocols are developed at this level to execute the interactions among the agents in the system. The interaction/communication among different logical units of the multiagent system architecture, with legacy systems as well as other sources of information or resource is specified at this level. Also at this level is the design of the platform of navigation and the type of navigation of mobile agents carried out.

The tasks at this level can be summarized as the translation of the roles in the system to software agents, the definition of the relationships among these agents, structuring of the agents in the multiagent system and other software entities and design of the interactions among the agents.

4.4.3. Agent Internal Architecture Level

The agent architecture level is the level where the internal structure of an individual agent is designed. At this level, the notions of agent technology of autonomy, proactive behaviour, reactivity, and social behaviour, are represented in the design of the internal architecture of each agent in a multiagent system according to the role or roles that the agent has to fulfill.
The notions of agent technology are featured at different degrees in different agents based on the roles that the agents are designed to fulfill. For instance, an agent that is to fulfill a *supervisory role* in a multiagent system might require strong autonomy while subordinate agents to this agent may require weaker autonomy.

There are different types of agents based on the different types of roles that exist in an organization. Types of agents [90] include but are not limited to assistant agents [29], information agents [64], collaborative agents [27], and mobile agents [39].

An agent type determines the components of its internal architecture and how these components should be structured. The agent type, based on the role the agent is fulfilling, also determines which concept of agent technology is featured in the agent’s architecture and to what degree the concept is featured. There are different types of agent internal architectures. Types of agent architectures include deliberative architectures, reactive architectures, and hybrid architectures. Each of the types of architectures emphasizes one notion of agent technology over another according to the purposes they are designed for. For instance, deliberative agent architecture emphasizes planning based action i.e. proactive behaviour, over reactive behaviour. In this case, the components of the internal structure of the agent will be designed to reflect the emphasis on planning based action over reactive behaviour. The type of agent architecture that is used to model an agent depends on the agent’s type.

At the Agent Internal Architecture level, the software development tasks include identification of the components and the design of the structural arrangement of the components of the architecture; definition of the interaction strategy (how the agent will process messages received against the goals to present appropriate responses) and the design of each agent’s behavioural strategy, that is, the degree of autonomy; mix of proactive/reactive behaviours, considering the goals it has to achieve; the environment in which it has to operate; and its relationship with the other agents with which it has to interact.
4.4.4. Multiagent System Realization Level

This level is composed of three main sub levels according to the general phases of software development (section 4.2). These are Multiagent System Detailed Design, Multiagent System Implementation and Multiagent System Testing. The first step in multiagent system Detailed Design is taking a decision on the programming paradigm that will be used to implement the multiagent system. This is in recognition of the fact that multiagent systems are currently being implemented using different programming technologies such as Object Oriented, logic programming, Object Oriented extensions. Once a choice of the programming technology has been made, the multiagent system and the Agent internal architectural designs (the products of the two preceding levels of abstractions) will be further designed at the level of details required for implementation, using the programming constructs of the chosen programming paradigm. For instance, if Object Oriented programming paradigm is chosen for the implementation of a multiagent system, the designs of the multiagent system’s architecture, its interactions and the designs of the internal architectures of the agents will be converted to detailed designs in preparation for coding using classes and objects and other constructs and concepts of Object Oriented programming. Multiagent system Implementation is the conversion of the detailed designs of the multiagent system to code. Multiagent system Testing involves the examination of the implemented multiagent system to validate and verify it.

In summary, the Vertical dimension of the Two Way classification scheme has four levels which are Agent-Oriented Analysis, Multiagent System Architecture, Agent Internal Architecture and Multiagent System Realization levels from top to bottom. Table 4.2 below shows the vertical dimension of the Two Way Classification Scheme.
Table 4.2: Vertical Dimension of the Two Way Classification Scheme

<table>
<thead>
<tr>
<th>Vertical Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent-Oriented Analysis (Level 1)</td>
</tr>
<tr>
<td>Multiagent System Architecture (Level 2)</td>
</tr>
<tr>
<td>Agent Internal Architecture (Level 3)</td>
</tr>
<tr>
<td>Multiagent System Realization (Level 4)</td>
</tr>
</tbody>
</table>

4.5 Horizontal Dimension of the Classification Scheme

This section presents the horizontal dimension of the classification scheme. The horizontal dimension is designed to reflect the concepts of agent technology and software development tasks that are relevant to each of the levels of abstraction identified in section 4.4.

The Horizontal dimension (classification) of the classification scheme is defined separately at each of the levels of the vertical dimension (levels of abstractions). This is due to the fact that the software development tasks as well as the concepts of agent technology vary from one level to another thereby rendering a single horizontal classification across the levels either inadequate or redundant. The following subsections present the horizontal dimensions of each of the levels of the vertical dimension.

4.5.1. Agent-Oriented Analysis Level

The horizontal dimension at this level is designed to have two categories of agent-oriented patterns. These categories are Organizational Patterns and Interactional Patterns. Examples of recurring problems for which patterns will be defined at this level include ‘How should the relationship between roles $x$ and $y$ in context $z$ be
modeled?’ Table 4.3 is the horizontal dimension of the classification scheme at the Agent-Oriented Analysis level.

**Table 4.3: Horizontal Dimension at the Agent-Oriented Analysis level**

<table>
<thead>
<tr>
<th>Agent-Oriented Analysis (Level 1)</th>
<th>Organizational</th>
<th>Interactional</th>
</tr>
</thead>
</table>

**Agent-Oriented Analysis-Organizational Patterns**

The patterns in this category describe approaches to modeling Role/Role relationship models in analyzing real life systems, that is, using the roles and goals to model real life organizations.

**Agent-Oriented Analysis-Interactional Patterns**

This category of patterns defines interaction issues such as type of interaction; structuring of interaction for instance into sessions. An example is to have one session of interaction for a task or two sessions of interactions, that is, nested interactions.

### 4.5.2. Multiagent System Architecture Level

The horizontal dimension at this level is designed to have three categories of agent-oriented patterns. These categories are Definitional Patterns, Structural Patterns and Interactional Patterns. Examples of recurring problems for which patterns will be defined at this level include ‘How should a role $x$ in the context $y$ be translated into agent(s)?’, ‘How should the agents in the multiagent system be arranged to maximize the performance of the system?’; ‘How should the interaction mechanism among the agents in the multiagent system be modeled?’ Table 4.4 is the horizontal dimension of the classification scheme at the multiagent system Architecture level.
Table 4.4: Horizontal Dimension at the Multiagent Architecture level

<table>
<thead>
<tr>
<th>Multiagent System Architecture (Level 2)</th>
<th>Definitional</th>
<th>Structural</th>
<th>Interactional</th>
</tr>
</thead>
</table>

**Multiagent System Architecture-Definitional Patterns**

The patterns in this category describe different ways of combining or breaking roles up to create agents since agents are the design phase abstractions of the roles within the system.

After the translation of roles to agents, there would be a different number of agents corresponding to the roles in the role model at the analysis level. This category of patterns describes ways of modeling virtual organizations with agents. That is, these patterns model the agent version of relationships in the virtual organization.

This category of patterns recommends the type of agent to use for representing a role in the system. Types include information agent, assistant agents, and process agents. Defining the type of agent will inform the decision on how to design the agent’s internal architecture. It will inform the decision on how to combine a pattern in this category with a pattern in the Agent Internal Architecture-Structural category of patterns.

**Multiagent System Architecture-Structural Patterns**

This category of patterns describes how to partition agent organizations into architectural components and include necessary knowledge components in a way that ensures the quality attributes of the application are addressed in the design.

Patterns in this category will address how to model the environment of a multiagent system of the environment such as the internet, the intranet, and other software systems and how they are connected in the particular multiagent system.

**Multiagent System Architecture-Interactional Patterns**

The patterns in this category describe the interaction amongst (interfaces) the logical units modeled in the multiagent system-Structural patterns as well as how to model mobility platforms and infrastructure in the multiagent system. The patterns
in this category describe efficient communication mechanism for the domain, type of application and environment of the multiagent system.

4.5.3. Agent Internal Architecture Level

The horizontal dimension at this level is designed to have three categories of agent-oriented patterns. These categories are Structural Patterns, Interactional Patterns, and Strategic Patterns. Examples of recurring problems for which patterns will be defined at this level include: ‘What type of agent architecture should be used to design an agent x in the context c?’; ‘How should the components of the agent architecture for agent a be structured?’ Table 4.5 is the horizontal dimension of the classification scheme at the Agent Architecture level.

Table 4.5: Horizontal Dimension at the Agent Internal Architecture level

<table>
<thead>
<tr>
<th>Agent Internal Architecture (Level 3)</th>
<th>Structural</th>
<th>Interactional</th>
<th>Strategic</th>
</tr>
</thead>
</table>

Agent Internal Architecture-Structural Patterns
This category of patterns defines the design of the internal structure of a single agent. They describe the components of an agent’s internal structure and how these components are arranged or connected to achieve the agent’s goals or represent the agent’s type.

Agent Internal Architecture-Interactional Patterns
This category includes patterns that focus on the components of the architecture that describe how the agent handles interaction with other agents; how the agent interacts with its environment; and how the components of an agent’s internal structure interact with themselves to achieve the intended behaviour of the agent i.e. the control mechanism of the agent. It is possible for two architectures with the same geometry (layout of the components) to have different control mechanisms. For example, there are different types of layered architecture for agents. Though they are all layered, they differ in their control mechanisms.
**Agent Internal Architecture-Strategic Patterns**
These are patterns that focus on how specific notions of agent technology (autonomy, reactivity) are modeled in an agent for certain roles or agent types e.g. how should autonomy be modeled in an assistant agent?

### 4.5.4. Multiagent System Realization Level

The design patterns that are applicable at this level are based on the implementation paradigm chosen. For instance, the object oriented patterns such as the Gang of Four patterns become applicable at this level for multiagent system Detailed Design. Hence, this level is beyond this scope of the classification scheme presented in this thesis. There is the potential for future work at this level when Agent-Oriented Programming matures and appropriate agent technology programming constructs are defined.

Hence the only decision considered in this classification scheme at this level of abstraction is the choice of the programming paradigm to use in implementing the multiagent system.

**Table 4.6: Horizontal Dimension at the Multiagent System Realization level**

<table>
<thead>
<tr>
<th>Multiagent System Realization (Level 4)</th>
<th>Definitional</th>
<th>Others</th>
</tr>
</thead>
</table>

**Multiagent System Realization-Definitional Patterns**

The patterns in this category define the programming paradigm for implementing a particular multiagent system based on the domain, the type of agents, multiagent system design and other factors.

In summary, the horizontal dimension of the classification scheme has ten categories defined across the four levels of abstraction of the classification scheme.
The vertical and horizontal dimensions of the classification scheme are combined in the next section to give the Two Way classification scheme.

4.6 Combination of the Dimensions

Tables 4.3, 4.4, 4.5 and 4.6 are combined with table 4.2 to create the Two Way classification scheme. The Two Way classification scheme features four levels of abstraction namely Agent-Oriented Analysis, Multi-agent System Architecture, Agent Internal Architecture and Multi-agent System Realization levels from top to bottom. These levels represent the vertical dimension of the classification scheme. The first level, Agent-Oriented Analysis level, has two categories namely Organizational and Interactional categories. The second level, Multi-agent System Architecture level has three categories namely Definitional, Structural and Interactional categories. The third level, Agent Internal Architecture level has three categories namely Structural, Interactional and Strategic. The fourth level, Multi-agent System Realization level, has two categories namely Definitional and Others. These categories represent the horizontal dimension of the classification at each of the levels.

Table 4.7 presents the Two Way classification scheme showing the combination of its vertical and horizontal dimensions. The next section introduces the concept of agent-oriented pattern attributes.
4.7 Agent-Oriented Pattern Attributes

The Two Way classification scheme presented in the previous section describes the levels and categories for classifying agent-oriented patterns. This section presents the attributes of the patterns that fall into the categories of the different levels of classification.

4.7.1. Definition of Agent-Oriented Pattern Attributes

Categorizing agent-oriented patterns in the Two Way classification scheme, like any other classification scheme, is based on the premise that patterns that possess similar features are placed in the same category. The features that are common to the patterns in a category are the attributes of the category. In other words, the defining characteristics of a category of the Two Way classification scheme are the set of the common attributes of the patterns that belong to that category.

The attributes of agent-oriented patterns are divided into level and category attributes based on the dimensions of classification in the Two Way classification scheme. Level attributes are the attributes of the patterns that belong to a particular level of the Two Way classification scheme. The level attributes of a level may not...
necessarily apply to all the patterns in the different categories at that level. For instance, the attributes of the MAS Architecture level apply to any of the three categories at that level. However, each of the categories may not possess all the attributes of this level.

Category attributes are the attributes of the patterns that belong to a particular category of the Two Way classification scheme. The category attributes are applicable to only the patterns that belong to a category. They define the individual categories of a particular level of abstraction. For instance, the attributes of the Structural category are different from those of the Interactional category both at the same multiagent system Architecture level.

The agent-oriented pattern attributes are important for three main reasons. They provide a basis for: placing patterns in a category and for searching for patterns in the various categories; assessing the structure and quality of agent-oriented patterns; and defining description templates for agent-oriented patterns.

The application of agent-oriented pattern attributes for placing patterns in and retrieving patterns from the Two Way classification scheme as well as assessing the quality of patterns are described in Chapter 5. Also, detailed work on how pattern attributes provide a basis for the definition of description templates is presented in Chapter 6.

4.7.2. Identification of Agent-Oriented Pattern Attributes

The sources of the level and category attributes of the Two Way classification scheme include the concepts of agent technology, and the elements of software development problems and solutions that are applicable to the different levels and categories of the Two Way classification scheme. The following activities were carried out in order to outline the level and category attributes of the Two Way classification.
The first activity is the identification of the concepts of agent technology that are characteristic of each of the levels and categories of the classification scheme. This is achieved by the analysis of the levels of abstractions in agent-oriented software engineering and the software development tasks that define the categories. From this analysis, the concepts of agent technology that are characteristic of the levels and categories are identified and outlined as attributes of these levels and categories. For instance, analysis of the Agent-Oriented Analysis level of abstraction shows that concepts such as roles, goals, interaction are characteristic of this level. Further analysis of the individual software development tasks at this level provides the concepts that are peculiar to the different categories. An example is the concept of responsibility which is peculiar to the Organizational category at the Agent-Oriented Analysis level.

The second activity is the examination of documented analyses and designs of multiagent system development projects at different levels of abstractions. This examination is carried out in order to identify the elements of design problems and design solutions that are relevant to the different levels and categories of the classification scheme. Different documented analyses and designs of multiagent systems development are available [24, 121, 44]. The examination of two documented designs of agent architecture Process Agent and Supply Chain Integration and Coordination (SCIC) agent are presented here as illustrations of this activity for the purpose of identifying attributes relevant to particular categories of the classification scheme.

The main components of the Process Agent architecture [140] are Perception Generator, Reasoning Engine, Enactment Engine and Application Server. Other components include an Underlying Communication Service, a User Interface and Knowledge/Data Repositories. The Perception Generator uses input from the Underlying Communication Service to generate perceptions of the agent’s environment. The generated perceptions are sent to the Reasoning Engine to be used in carrying out bottom-up or top-down reasoning to produce plans and actions. The results of the Reasoning Engine are sent to the Enactment Engine for
execution. The Enactment Engine also controls the activities of the other components of the architecture. Execution of decided actions requiring the invocation of other applications is sent to the Application Server to execute. The knowledge/data repositories provide inputs to the four main components while some of the repositories are also updated by the Enactment Engine. See Figure 4.8 for the diagram of the Process Agent Architecture.

Figure 4.8: Process Agent Architecture

An examination of this agent architecture shows that it is composed of the internal components of an agent, how these components are arranged and how they interrelate to achieve the goals and behaviour of the agent. The components of this agent’s internal architecture include Knowledge Component, Interaction Component, Environmental Interface Component, Strategy Components and Geometry.

The design of the internal architecture of a supply chain integration and coordination (SCIC) agent is presented in [113]. The SCIC agent architecture consists of an Internal Data/Knowledge Base, Role Capabilities Module, BDI Kernel Module, Negotiations Module and a Communications Module. The Internal Data/Knowledge Base Module is responsible for Knowledge Management of the
agent, the BDI Kernel Module is the agent’s center of control. The Negotiations Module handles the analysis of incoming messages and construction of outgoing messages. The Role Capabilities Module is responsible for striking the balance between the reactive and deliberative (goal directed) behaviours of the agent and the Communication Module is the agent’s interface with its environment. This design also describes the connections among the different components of the architecture for the purposes of control and transfer of information. See Figure 4.9 for the diagram of the SCIC agent architecture.

An examination of this agent architecture shows that it is composed of the internal components of an agent, how these components are arranged and how they interrelate to be able to achieve the goals and behaviour of the agent. The components of this agent’s internal architecture include Knowledge Component, Strategy Component, Interaction Component, Environmental Interface Component.

The components of the SCIC Agent architecture and their arrangement are similar to those of the Process Agent architecture. Both architectures are about single agents, they both present internal components of single agents and they describe the way in which these internal components are arranged. The internal components of the two architectures include knowledge component, interaction component,
environmental interface component, strategy component and geometry. The results of these examinations identify level attributes of the Agent Internal Architecture level and Structural category of the same level of the classification respectively. The level attributes are Single Agents, Agent Internal components and Arrangement of Agent Internal components. The category attributes of the Structural category are Knowledge component, Interaction component, Environmental Interface component, Strategy component and Geometry.

The third activity is the analysis of some of the existing agent-oriented patterns, to identify elements of the patterns' problems and solutions that are peculiar to the levels and categories the patterns belong to. There are a number of agent-oriented patterns written by different authors [127, 2, 117, 61, 40]. The analyses of six agent-oriented patterns are presented here as illustrations of this activity for the purpose of identifying attributes that are relevant to particular categories of the classification scheme.

The TROPOS project [65, 66] presents 10 Organizational Styles (patterns) for modeling multiagent systems. Organizational styles model the relationships and coordination of business stakeholders (i.e. individuals, physical or social systems) to achieve common goals. Based on a review of the Takeover and Vertical Integration organizational styles, it is observed that they represent different ways to structure an organization in order to achieve its strategic objectives. Takeover and Vertical Integration styles are structured to define the roles of entities in the organization, their responsibilities and resources as well as the goals of the organization. The review of these two styles was used to identify the problem and solution elements that are common to both of them and therefore are characteristic of the level and/or category of the classification scheme that the styles belong to. Figures 4.24 and 4.25 are diagrams of the Takeover and Vertical Integration organizational styles respectively. The analysis of these styles shows that their problem elements include coordination, control, associations and relationships. Their solution elements include roles, goals, responsibilities, and resources. These
problem and solution elements can be identified as attributes of the Organizational category of the Agent Analysis level of abstraction.

Furthermore, the analysis of organizational styles helps to distinguish between organization and architecture of multiagent system as follows. Whereas organizational styles focus on the arrangement of the real life system being modeled, architecture focuses on the arrangement of the components of the software system to be built for the real life system. Organizational styles therefore may contain non software entities in the modeling. Architecture will however not include non software entities in its modeling. Where non software entities must be included in the architecture, they may be referenced as users of the system.

Heinze in [41] describes a number of patterns for intention recognition. The focus of these patterns is the structuring of the interaction aspect of individual agent architecture. The examination of one of these patterns, the Hybrid Recognizer pattern, shows that the pattern defines a particular strategy for message input, perception handling, and message processing in a defined context. These elements of the solution are relevant to the Interactional category of the Agent Architecture level of abstraction and are therefore candidates of the attributes of this category.

The attributes of the different levels and categories are identified by a combination of the three activities explained above. The attributes identified for the levels and categories of the Two Way classification scheme are described in the following subsection.

4.7.3. Two Way Classification Scheme Level Attributes

Table 4.8, is a listing of the level and category attributes of the Two Way classification scheme. The following are the descriptions of the level attributes for all the levels of the Two Way classification scheme.
Table 4.8: Listing of the Level and Category Attributes

<table>
<thead>
<tr>
<th>Agent-Oriented Analysis (Level 1)</th>
<th>Organizational</th>
<th>Interactional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles/Multiple Roles</td>
<td>Roles</td>
<td>Interaction objective</td>
</tr>
<tr>
<td>Organization of Roles</td>
<td>Goals</td>
<td>Interaction type</td>
</tr>
<tr>
<td>Interaction</td>
<td>Coordination/Control</td>
<td>Domain Rules</td>
</tr>
<tr>
<td>Application Domain</td>
<td>Associations/Relationships</td>
<td>Interaction Roles</td>
</tr>
<tr>
<td></td>
<td>Responsibility</td>
<td>Distribution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accessibility</td>
</tr>
<tr>
<td>Multiagent System Architecture (Level 2)</td>
<td>Definitional</td>
<td>Structural</td>
</tr>
<tr>
<td>Roles and Agents</td>
<td>Combination Rules</td>
<td>Knowledge Repositories</td>
</tr>
<tr>
<td>Multiple Agents</td>
<td>Division Rules</td>
<td>Other entities</td>
</tr>
<tr>
<td>Arrangement of Agents</td>
<td>Goals</td>
<td>Geometry</td>
</tr>
<tr>
<td>Quality Goals</td>
<td>Responsibility</td>
<td>Bandwidth</td>
</tr>
<tr>
<td>Interaction/Communication</td>
<td></td>
<td>Host Stability</td>
</tr>
<tr>
<td>Constraints</td>
<td></td>
<td>Data Volume</td>
</tr>
<tr>
<td>Infrastructure</td>
<td></td>
<td>Control flow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Introduced Agents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Message Exchange Rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Performatives/Message Format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ontology</td>
</tr>
<tr>
<td>Agent Internal Architecture (Level 3)</td>
<td>Structural</td>
<td>Interactional</td>
</tr>
<tr>
<td>Single Agent</td>
<td>Knowledge Component</td>
<td>Message Input</td>
</tr>
<tr>
<td>Agent Internal Components</td>
<td>Strategy Component</td>
<td>Message Output</td>
</tr>
<tr>
<td>Arrangement of Agent Components</td>
<td>Interaction Component</td>
<td>Message Processing</td>
</tr>
<tr>
<td>Constraints</td>
<td>Environmental Interface Component</td>
<td>Perception Handling</td>
</tr>
<tr>
<td></td>
<td>Geometry</td>
<td>Action Handling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Autonomy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reactivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proactive Behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Knowledge Management</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adaptability</td>
</tr>
<tr>
<td>Multiagent System Realization (Level 4)</td>
<td>Definitional (Technology)</td>
<td>Others</td>
</tr>
<tr>
<td>Data Definitions (Classes and Objects)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dependencies/Associations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDE/Programming Language Constructs</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sample Code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functions/Procedures</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.7.3.1. Agent-Oriented Analysis level – Level 1

The following are the attributes of this level of abstraction.

Roles/Multiple Roles: role is a concept of agent technology that is featured at this level of abstraction. The analysis of multiagent systems involves multiple roles either in organization or in interaction modeling. Patterns at this level are about
solutions to problems relating to arrangement of multiple roles in the analysis of multiagent systems

**Organization of Roles:** organization of roles is a multiagent system analysis problem. Organization of roles deals with the modeling of virtual organizations. Some of the patterns at this level will model the relationship of the members of an existing organization with roles.

**Interaction:** interaction is the representation of the dynamic relationship of members of a multiagent system. It is also a part representation of the social behaviour of a multiagent system. Some of the patterns at this level will analyze the interactions in the virtual organization.

**Application Domain:** application domain is important at this level because it influences the modeling of the virtual organization and the analyses of the interactions in the organizations. The patterns at this level will recognize the application domain in their problem context definition.

### 4.7.3.2. MAS Architecture – Level 2
The following are the attributes of this level of abstraction.

**Roles and Agents:** roles and agents are concepts of agent technology. They appear together as an attribute of this level because roles are converted to agents at this level. As a result, patterns that address the conversion of roles to agents at this level will feature both roles and agents.

**Multiple Agents:** since roles are translated to agents at this level, models equivalent to virtual organizations will now feature multiple agents at this level. Multiple agents will be involved in the architectural structures of multiagent system and interactions will involve many agents playing the different roles at the analysis level. Most of the patterns at this level will feature on multiple agents.

**Arrangement of Agents:** whereas the organizational models at the analysis level are about modeling the virtual organization to reflect the real life organization; the architectural models are about arranging the agents and other entities into modules/components in order to create a functional software system that will achieve the objectives of the virtual organization. Some of the patterns at this level
will be about the arrangement of the agents other entities into a functioning software system

**Quality Goals:** quality goals are software system performance requirements. The design considerations of multiagent system architectures are impacted by quality goals. Some of the patterns at this level will be based on incorporating the quality goals into the design of the multiagent system architectural structures

**Interaction/Communication:** interaction at this level involves the design of the interaction protocols among the agents in the multiagent system. Communication at this level refers to network layout and communication issues. Patterns at this level will address agent interaction protocols, intra and inter software subsystem communication considerations and navigation strategies for mobile agents.

**Constraints:** constraints refer to the limitations facing specific design considerations at this level. Some of the patterns at this level will highlight and address the constraints surrounding the type and dynamics of interactions among the agents in the multiagent system

**Infrastructure:** infrastructure relates to the hardware components of the multiagent system architecture and the hardware facilities for communication. The patterns at this level will address issues relating to the hardware infrastructure on which the multiagent system will be built. Some of the patterns that belong to this level will consider issues such as network bandwidth, number of nodes on the network, capacity of the nodes on the network.

### 4.7.3.3. Agent Internal Architecture level – Level 3

The following are the attributes of this level of abstraction.

**Single Agent:** the focus of the design activities at this level of abstraction is on one agent. A major distinguishing feature of this level is that the number of agents involved in a problem to be solved should be one. Therefore, a quick indication that a pattern belongs to this level is that it is about a single agent

**Agent Internal Components:** agent internal component is a function of the design tasks at this level. The task is to define the internal components that represent the different behaviours of an agent. Some of the patterns that belong to this level deal
with problems concerning the identification of the internal components of a single agent and the functions of these components

**Arrangement of Agent Components:** arrangement of agent internal components is a function of the design tasks at this level. The task is to design the organization of the internal components of the agent such that they will function to carry out its responsibilities and achieve its goals. Some of the patterns at this level deal with the challenges facing the design of the structural arrangement of an agent’s internal components

**Constraints:** constraints refer to the limitations facing specific design considerations at this level. Some of the patterns at this level will highlight and address the constraints surrounding the choice of components and their arrangement

### 4.7.3.4. MAS Realization – Level 4

This is the level where the implementation paradigm is determined. The design decisions and activities are then based on the chosen implementation paradigm. Hence, the design patterns that are used at this level are also based on the chosen implementation paradigm.

**Data Definitions (Classes and Objects):** data definitions are the units of representation of the data and behaviour of the system at the detailed design level. The patterns at this level address the problems with the design of the data definitions of the multiagent system in the language of the chosen paradigm of implementation

**Dependencies/Associations:** dependencies or associations are the representations of the static relationship of the members of the system at the detailed design level. The patterns at this level address the problems with the design of the static relationships in multiagent system using the language of the chosen paradigm of implementation

**Programming Language Constructs:** programming language an integrated development environment (IDE) constructs provide abstractions of certain design concepts. Some of the patterns at this level will focus on how to maximize the potential of these constructs in solving specific design problems
**Sample Code:** sample code is a feature of design patterns provided to illustrate the implementation of the pattern being described. This is a common attribute of the patterns at this level of software development

**Functions/Procedures:** functions or procedures or methods indicate another level of refinement of the design of a software system. As a result, the featuring of functions or methods in the solution of some patterns indicates that such patterns are relevant to the implementation level of development

4.7.4. **Two Way Classification Scheme Category Attributes**

The preceding subsection describes the level attributes of the Two Way classification scheme. This subsection presents the descriptions of the category attributes for all the categories of the Two Way classification scheme.

4.7.4.1. **Agent-Oriented Analysis-Organizational**

The main agent-oriented software engineering task relevant to this category of patterns is the analysis and modeling of real life organizations in terms of the concept of agent technology. The following are the attributes of this category.

**Roles:** role is a concept of agent technology that is critical to the analysis and modeling of real life organizations for multiagent system development. The patterns in this category address issues relating to the identification of role and role hierarchies in the organization being analyzed for the purpose of modeling the representation of the real life organization that is, modeling the virtual organization.

**Goals:** goal is a concept of agent technology that is critical to the analysis and modeling of real life organizations for multiagent system development. The patterns in this category deal with issues relating to the definition of goals, sub goals, and goal dependencies. Also, goals will feature in the patterns addressing other issues in this category.

**Coordination/Control:** the focus of this category is the representation of the real life organization in terms or roles and goals. Some of the patterns in this category are concerned with how to model the coordination of the roles in the virtual organization in order to achieve the overall goal of the system.
**Associations/Relationships:** associations or relationships represent the lines of control and flow of information and resources in the virtual organization. Associations represent an analysis problem. The patterns in this category model the kind of associations or relationships such as hierarchies, team structures, and provider-consumer amongst the members of the virtual organization.

**Responsibility:** responsibility is a concept of agent technology that represents the tasks of the roles within the virtual organization. Responsibilities features in the patterns in this category when identifying the roles in the organization and defining role relationships.

### 4.7.4.2. Agent-Oriented Analysis-Interactional

The main agent-oriented software engineering task relevant to this category of patterns is the analysis of the interactions in the real life organizations in terms of the concept of agent technology. That is analysis of the interactions among the roles in the virtual organization. The following are the attributes of this category.

**Interaction objective:** interaction objective describes the purpose of an interaction in a virtual organization. Interaction objective features in patterns that deal with the analysis of interactions.

**Interaction type:** interaction type is an interaction analysis problem. The interaction type impacts of design considerations at the other levels of abstractions. Interaction types include negotiation, collaboration, and argumentation. Some of the patterns in this category address are concerned with the determination of the type of the interactions in a virtual organization.

**Domain Rules:** domain rules express the nature of the domain and the conditions that the domain imposes on the interactions in the virtual organization being analyzed. Domain rules feature in some of the patterns in this category as considerations for a particular interaction solution.

**Interaction roles:** interaction roles describe the different parts to be played by the participants in an interaction. Examples of interaction roles include initiator and participants, broker, providers and clients, and mediator. Some of the patterns in this category address the identification of the interaction roles and the number of the different interaction roles in an interaction.
**Distribution:** distribution is the connection structure among the interaction roles in an interaction. Distribution is an analysis problem that describes the number of participants an interaction role is connected with in an interaction. Examples include One-Many and Many-Many. Some of the patterns in this category address the distribution aspect of an interaction in a virtual organization.

**Accessibility:** accessibility describes an interaction role’s awareness of the other participants in the interaction. Accessibility is based on the domain of the virtual organization. Compare agents in an Intelligent Manufacturing System with an agent on the internet. The patterns in this category consider accessibility in the analysis of the interactions in the virtual organization.

4.7.4.3. **Multiagent System Architecture-Definitional**

The main agent-oriented software engineering task relevant to this category of patterns is the identification of the agents that will encapsulate the software system roles in the virtual organization defined at the Agent-Oriented Analysis level. That is the translation of roles at the analysis level to agents at the multiagent system architecture level. The following are the attributes of this category.

**Combination Rules:** combination rules are the rules or rationale for combining two or more roles into agents. This is an agent identification decision. The patterns in this category feature combination rules in their solutions.

**Division Rules:** division rules are the rules or rationale for dividing one role into two or more agents. This is an agent identification decision. The patterns in this category feature division rules in their solutions.

**Agent Type:** agent type is the kind of agent that a role is translated into. Examples of agent type include, information agents, negotiating agents, assistant agents, and mobile agents. Agent type is a design problem that is addressed by some of the patterns in this category.

**Goals:** goal is a concept of agent technology. It is relevant in the considerations of the characteristics of a role for the purpose of translating the role to agent or agents. The patterns in this category consider nature of the goal of each role in the decision of how to translate the role to agent or agents.
Responsibility: responsibility is a characteristic of a role in a virtual organization. The responsibilities of a role are considered in the decision regarding the translation of the role to agent or agents by the patterns in this category.

4.7.4.4. Multiagent System Architecture-Structural
The main agent-oriented software engineering task relevant to this category of patterns is the design of the multiagent system architectural structure. The following are the attributes of this category.

Knowledge Repositories: knowledge repositories include the software system units that hold the knowledge required in the multiagent system. The patterns in this category feature knowledge repositories in architectural design solutions.

Other entities: other entities are the non agent members of the multiagent system architecture. They are introduced for the realization of the responsibilities of the agents in the multiagent system. An example is an image processing system. The patterns in this category feature other entities in architectural design solutions.

Geometry: geometry is the relative arrangement of the components and entities of multiagent system architecture into a functional system that can be developed. The patterns in this category describe the geometry of the multiagent system’s architecture.

Bandwidth: bandwidth describes the amount of information that can be transmitted per time over the communication channels in a system’s infrastructure. This is a critical consideration regarding navigation of mobile agents. Bandwidth is featured in the patterns of this category that address mobility options.

Host Stability: host stability describes the consistency of the hosts in a multiagent system’s infrastructure. This is a consideration that impacts on the navigation strategies of mobile agents. The patterns in this category that relate to mobile agents consider host stability in the design of navigation strategies.

Data Volume: data volume is the amount of data and information that an agent is required to process in order to achieve its goals. This is a consideration that impacts on the navigation strategies of mobile agents. The patterns in this category that relate to mobile agents consider data volume in the design of navigation strategies.
4.7.4.5. Multiagent System Architecture-Interactional

The main agent-oriented software engineering task relevant to this category of patterns is the design of the interaction mechanisms and policies among the agents in multiagent system. That is the design of agent interaction protocols. The following are the attributes of this category.

**Control flow:** control flow describes the degree of influence of one agent on the other agents in the same interaction. Control flow affects decisions such as which agent can initiate and/or terminate an interaction. The patterns in this category consider control flow in the design of interaction protocols.

**Introduced Agents:** introduced agents are participants in interagent interactions which were not identified by the translation of the software system roles to agents. Introduced agents are defined solely for the purpose of facilitating interactions among the agents in the multiagent system. An example of an introduced agent is the Embassy agent in the Embassy pattern [40]. The patterns in this category include introduced agents in the solution to interaction protocol designs.

**Message Exchange Rules:** message exchange rules govern the exchange of information among interacting agents in a multiagent system. An example of message exchange rules is the English Auction rules. The patterns in this category describe message exchange rules in their interaction protocol design solutions.

**Performatives/Message Format:** performatives are the speech acts used in the design of interactions in multiagent systems. Message format describes the structure and content of each message that is exchanged among interacting agents. The patterns in this category describe the performatives and message format in their interaction protocol designs.

**Ontology:** ontology is the structure of the knowledge that is to be used in a multiagent system. The patterns in this category will consider ontology is their design of the performatives and message format.

4.7.4.6. Agent Internal Architecture-Structural

The main agent-oriented software engineering task relevant to this category of patterns is the design of the internal components of an agent and how they are structured in the most efficient way to realize the goal of the agent. That is the
design of agent internal architecture. The following are the attributes of this category.

**Knowledge Component:** knowledge component of an agent holds its beliefs rule base, plans and representations of its environment based on the type of agent it is and its goals. The patterns in this category feature the knowledge component of the agent in their design solutions

**Strategy Component:** strategy component of an agent holds its representation of autonomy, control, proactive and reactive behaviours. The patterns in this category feature the strategy component of the agent in their design solutions

**Interaction Component:** interaction component of an agent holds its interaction protocols and message processing. The patterns in this category feature the interaction component of the agent in their design solutions

**Environmental Interface Component:** environmental interface component of an agent describes the agent’s mode of connection with its environment. It defines the agent’s mechanism for action and perception and message sending and receiving. The patterns in this category feature the environmental interface component of the agent in their design solutions

**Geometry:** geometry is the relative arrangement of the internal components of an agent into a functional system with the components working together to realize the behaviour of the agent and achieve its goals. The patterns in this category describe the geometry of the agent’s architecture in their design solutions

4.7.4.7. Agent Internal Architecture-Interactional

The main agent-oriented software engineering task relevant to this category of patterns is how the agent interfaces with its environment and how the agent processes high level contextual messages. The focus of the task is the design of mechanisms for the realization of the environmental interface and interaction management components of the agent internal architecture. The following are the attributes of this category.

**Message input/processing/output:** message IO and processing describes how the agent handles the processing of messages received and how it constructs messages
to be sent. The patterns in this category describe the message IO and processing tasks of the agent in their design solutions

**Perception handling:** perception handling describes the agent’s mechanism for observing or sensing its environment or receiving feedback from its environment. The patterns in this category describe the perception handling mechanism of the agent in their design solutions

**Action Execution:** action execution describes the agent’s mechanism for carrying out its actions on its environment. The patterns in this category describe the action execution mechanism of the agent in their design solutions

**Environment:** an agent’s environment is the context of its existence and its sphere of influence. The patterns in this category feature an agent’s environment as a consideration for the design of its interaction strategies.

### 4.7.4.8. Agent Internal Architecture-Strategic

The main agent-oriented software engineering task relevant to this category of patterns is how the agent realizes its behavioural characteristics. That is how the agent achieves autonomy, the right mix of reactive and deliberative behaviour, learning and such characteristics. This task concerns the degree of details required in designing these behavioural characteristics of the agent. The following are the attributes of this category.

**Autonomy:** autonomy is the ability of the agent to maintain control over its internal state and behaviour. The patterns in this category describe the agent’s strategy for realizing autonomy in their design solutions

**Reactivity:** reactivity is the ability of the agent to react appropriately and promptly to changes in its environment without maintaining an internal representation of the elements of its environment. The patterns in this category describe the agent’s strategy for realizing reactivity in their design solutions

**Proactive Behaviour:** proactive behaviour is the ability of the agent to take goal directed initiatives in its environment. The patterns in this category describe the agent’s strategy for realizing proactive behaviour in their design solutions

**Knowledge Management:** knowledge management is an agent’s strategy for building and updating its belief base. Knowledge management describes how the
agent structures, updates and uses its knowledge (beliefs). The patterns in this category describe the agent’s strategy for realizing knowledge management in their design solutions.

**Adaptability:** adaptability is the ability of an agent to change its behaviour due to changes in its environment. The patterns in this category describe the agent’s strategy for realizing adaptability in their design solutions.

### 4.8 Chapter Summary

Agent-oriented pattern classification scheme should set the framework for incorporating the concepts of agent technology into the categorization and description of agent-oriented patterns. A classification scheme provides the platform for classifying, analyzing and describing agent-oriented patterns. That is a classification scheme provides the platform for developing an agent-oriented pattern catalog, analyzing the quality and usability of agent-oriented patterns and designing agent-oriented pattern templates that reflect the concepts of agent technology. A classification scheme should be comprehensive and universal.

This Chapter identified the levels of abstractions in agent-oriented software engineering to provide the basis for the design of an agent-oriented pattern classification scheme. The Two Way classification scheme is then presented to provide the platform for classifying, analyzing and describing agent-oriented patterns. This classification scheme comprises of two dimensions, the vertical dimension and the horizontal dimension. The vertical dimension is based on the levels of abstractions in agent-oriented software engineering while the horizontal dimension is defined by the software development tasks at each of the levels of abstractions. The Two Way classification scheme is comprehensive because it covers different levels of abstractions in agent-oriented software engineering and it is universal because it is not locked in to a particular agent-oriented software engineering methodology or a particular agent based development project.
The properties of the levels and categories of the Two Way classification scheme were defined in this Chapter. These properties represent the attributes that are expected of the patterns that fall into the levels and categories of the classification scheme. The attributes represent the criteria for the analysis of agent-oriented patterns and the basis for the design of agent-oriented pattern templates.

In Chapter Five, twenty eight existing agent-oriented patterns are analyzed and classified according to the categories of the Two Way classification scheme using the level and category attributes defined in this Chapter as the criteria for analysis. Chapter Six presents eight agent-oriented pattern templates according to the categories of the Two Way classification scheme using the attributes defined in this Chapter as the guide for defining the elements of these templates.
Chapter 5

Analysis of Agent-Oriented Patterns

5.1 Introduction

Chapter Four presented the Two Way classification scheme along with its level and category attributes. Level attributes are the distinguishing properties of the patterns that belong to the levels in the Two Way classification scheme. Category attributes are the distinguishing properties of the patterns that belong to the categories in the Two Way classification scheme. There is a different set of level attributes and category attributes for each of the levels and the categories of the Two Way classification scheme (see section 4.7.3).

This Chapter presents a process for analyzing agent-oriented patterns using level and category attributes as the basis of the analysis. The process of analyzing agent-oriented patterns using level and category attributes is termed attribute-based analysis in this thesis. Analysis of patterns facilitates the placement of existing patterns in the appropriate category of the classification scheme. Analysis of patterns is useful for pattern writers to determine the appropriate category of the classification scheme to place the patterns they write. Attribute-based analysis provides a repeatable process for classifying patterns into the categories of the Two Way classification scheme. This classification of patterns using attribute-based analysis demonstrates the applicability of the Two Way classification scheme.

For this Chapter, Ninety seven (97) agent-oriented patterns were studied from fifteen (15) authors. Many of the patterns by the same authors address similar
agent-oriented software engineering problems. Twenty eight (28) patterns were chosen to be presented in depth in this Chapter.

The next section, section 5.2 describes attribute-based analysis of agent-oriented patterns and discusses the uses of this analysis for placing patterns and assessing quality and complexity of patterns. This section also presents attributes tables which are the tools for carrying out attribute-based analysis. Section 5.3 describes the step by step process of attribute-based analysis for classifying agent-oriented patterns into the categories of the Two Way classification scheme. In section 5.4, the analyses and classifications of twenty eight (28) agent-oriented patterns are described. General comments on the various analyses of patterns and the results of the analyses are presented in section 5.5.

5.2 Attribute-based Analysis: Concept and Tools

This section describes attribute-based analysis showing how it is applied to demonstrate the applicability of the Two Way classification scheme. The attribute tables that represent the tools of this analysis are also presented in this section.

5.2.1. Attribute-based Analysis: Concept

Attribute-based analysis is the examination of an agent-oriented pattern to determine its attributes and using those attributes to evaluate the pattern for different purposes. That is, attribute-based analysis is the evaluation of an agent-oriented pattern using the attributes present in the pattern. The attributes of an agent-oriented pattern are derived from the elements of the patterns’ problem and/or solution as well as concepts of agent technology featured in the pattern (see section 4.7.2). Attribute-based analysis is therefore useful for the three following purposes.

- Attribute-based analysis can be used to assess the versatility of an agent-oriented pattern’s solution. The information contained in and structure of a pattern’s solution determines its versatility. Some agent-oriented patterns can
be applied to solving problems in different but similar software development tasks. For instance, a pattern could be applied to address both structural and strategic problems at the Agent Internal Architecture level. Another pattern could be applied to address interaction problems at both the agent-oriented analysis and multiagent system architecture levels. Examples of such patterns are discussed in section 5.4.

- Attribute-based analysis can be used to assess the quality of an agent-oriented pattern. The quality of a pattern is determined by its expressiveness, completeness, solution clarity and adaptability. The qualities of a pattern are discussed in section 2.3.6 of Chapter Two. The presence or otherwise of the necessary properties and features that ensure quality in a pattern, can be assessed by attribute-based analysis. Chapter 6 describes the use of agent-oriented pattern attributes in the design of agent-oriented pattern description templates and how the templates ensure quality of patterns described by them.

- Attribute-based analysis is necessary for classifying existing agent-oriented patterns according to the categories of the Two Way classification scheme. The process of attribute-based analysis presented in section 5.3 describes how to apply attribute-based analysis to the classification of agent-oriented patterns according to the categories of the Two Way classification scheme.

The outcome of attribute-based analysis is a set of agent-oriented pattern attributes that are present in a pattern. This list of attributes present in a pattern is used to determine the versatility quality or classification of the pattern.

5.2.2. Attribute-based Analysis: Tools

Attributes Table is a tool for carrying out attribute-based analysis. An attributes table comprises of a list of agent-oriented pattern attributes and placeholders to indicate the presence or otherwise of each attribute in an agent-oriented pattern being analyzed. Two types of attributes tables are presented for carrying out
attribute-based analysis for the purpose of classifying patterns in the Two Way classification scheme. These are a level attributes table and a category attributes table.

The level attributes table comprises of a column for each attribute of the four levels of the Two Way classification scheme and a row for each pattern. A cell in the table indicates the presence or otherwise of each attribute in a pattern being analyzed. Hence there is only one level attribute table for the Two Way classification scheme (see table 5.1).

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Vertical Classification – All levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Agent-Oriented Analysis</td>
</tr>
<tr>
<td>Roles/Multiple Roles</td>
<td>Organization of Roles</td>
</tr>
<tr>
<td>1.</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td></td>
</tr>
</tbody>
</table>

A category attributes table comprises of a column for each attribute of all the categories at a particular level of the Two Way classification scheme and a row for each pattern. A cell in the table indicates the presence or otherwise of each attribute in a pattern being analyzed. This thesis presents three category attributes tables, one for each of the first three levels of the Two Way classification scheme. There is no category attributes table for the MAS Realization level since the categories and therefore category attributes of this level are not defined in this thesis (see section 4.7.4). Tables 5.2, 5.3 and 5.4 are the category attributes tables for the Agent-Oriented Analysis, MAS Architecture and Agent Internal Architecture levels of abstractions of the Two Way classification scheme respectively.
### Table 5.2: Category Attributes Table – Agent-Oriented Analysis Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Organizational</th>
<th>Interactional</th>
<th>Analysis Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Roles</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Coordination/Control</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Association/Relationship</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Goals</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Responsibility</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction Objective</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction Type</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interaction Roles</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Domain Roles</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Distribution</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Accessibility</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.

2.

### Table 5.3: Category Attributes Table – Multiagent System Architecture Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Definitional</th>
<th>Structural</th>
<th>Interactional</th>
<th>Analysis Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Combination Roles</td>
<td>Division Rules</td>
<td>Goals</td>
<td>Responsibilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.

2.

### Table 5.4: Category Attributes Table – Agent Internal Architecture Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Structural</th>
<th>Interactional</th>
<th>Strategic</th>
<th>Analysis Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1.

2.
5.3 Attribute-based Analysis: The Process

Section 5.2 described the attribute-based analysis, the level attributes table and three category attributes table for analyzing agent-oriented patterns. This section describes the process of attribute-based analysis for classifying agent-oriented patterns into the categories of the Two Way classification scheme. The different activities involved in the attribute-based analysis process are described in this section.

The attribute-based analysis process involves the following activities: examining the pattern; identifying the level attributes that are present in the pattern; determining the Two Way classification scheme level that the pattern belongs to; identifying the category attributes of the pattern’s Two Way classification scheme level, that are present in the pattern; and determining the category that the pattern belongs to.

Activity 1
Examination of an agent-oriented pattern for the purpose of classification in the Two Way classification scheme is the review of the parts of the pattern to discover the attributes of the pattern. This involves reviewing the pattern for its problem and solution components that are characteristic of general software development tasks. For instance a pattern’s problem could relate to the relationship of the components of a system, another pattern’s solution could define a strategy for interoperability of different systems. Also involved in the examination of an agent-oriented pattern for the purpose of classification is the review of the pattern to discover the concepts of agent technology that are featured in the pattern. For instance, a pattern could feature only roles or only agents and another one could feature both roles and agents.

Activity 2
The level attributes table (table 5.1) is used for identifying the level attributes that are present in an agent-oriented pattern for the purpose of classification in the Two
Way classification scheme. This activity involves placing the pattern on the level attributes table and indicating which of the level attributes listed in the table are present in the pattern, based on the result of the examination of the pattern already carried out.

**Activity 3**
The next activity in the process is the determination of the Two Way classification scheme level that the pattern belongs to. This activity uses the result of the preceding activity, that is, identification of the level attributes that are present in the pattern. The set of level attributes that are present in the pattern is compared with each of the attributes of the individual levels of the classification scheme. The level whose attributes have the best match with the attributes that are present in the pattern is determined to be the pattern’s level in the classification scheme. Level of best match means the level where the majority of the attributes in the pattern are found. For instance, if a pattern has five (5) attributes that are found in two or more levels, the level of best match is the level that has three (3) of the attributes that are present in the pattern. This activity determines the vertical classification of the pattern.

**Activity 4**
Identification of the category attributes that are present in the pattern is the fourth activity in this process. One of the three category attributes tables (table 5.2, 5.3 or 5.4) is used for identifying the category attributes that are present in the pattern. The category attributes table that is used for this analysis is determined by the result of the preceding activity. For instance, if the result of the preceding activity identifies the pattern’s vertical classification as MAS Architecture level, the category attributes table that will be used for this activity is the category attributes table at the MAS Architecture level (table 5.3). This activity involves placing the pattern on the category attributes table and indicating which of the category attributes listed in the table are present in the pattern, based on the result of the examination of the pattern already carried out. This activity is not carried out for patterns identified to belong to the MAS Realization level because the categories
and therefore category attributes of this level are not defined in this thesis (see section 4.7.4).

**Activity 5**
The last activity in the process is the determination of the category that the pattern belongs to at its identified level in the Two Way classification scheme. This activity uses the result of the preceding activity, that is, identification of the category attributes that are present in the pattern. In this activity, the set of category attributes that are present in the pattern is compared with each of the attributes of the individual categories of the pattern’s vertical classification level. The category whose attributes have the best match with the attributes that are present in the pattern is determined to be the pattern’s category at the pattern’s vertical classification level. The concept of best match as it is applicable to the determination of the patterns level of classification is applicable here as well. This activity determines the horizontal classification of the pattern.

The process of classifying a pattern $P$ into a category $C$ that is defined by the vertical and horizontal classifications of the Two Way classification scheme can be illustrated by the following activities. Pattern $P$ is examined for the purpose of identifying the attributes in it. Then the level attributes $LA_P$ that are present in $P$ are identified. The level attributes $LA_P$ are used to determine the Two Way classification scheme level $L_P$ that $P$ belongs to. The category attributes $CA_P$ of level $L_P$ that are present in $P$ are identified. Then the category attributes $CA_P$ are used to determine the category $C_P$ of level $L_P$ that $P$ belongs to. Figure 5.1 is a representation of the attribute-based analysis process.
Figure 5.1: The Attribute-based Analysis Process

Section 5.4 presents the analysis and classification of twenty eight (28) agent-oriented patterns using the process described in this section.
5.4 Pattern by Pattern Analysis and Classification

This section analyses and classifies twenty eight (28) agent-oriented patterns with respect to the categories of the Two Way classification scheme. The rationale for the choice of the patterns analyzed, the summary of the results, and the presentation format of the individual pattern analysis are presented before the analysis of the twenty eight patterns.

A collection of Ninety seven (97) agent-oriented patterns were reviewed in the course of this research (table 5.5). These patterns were collated from a broad range of literature on agent technology. This collection of patterns represents the works of fifteen different authors written to capture experiences gained from different applications of agent technology. A common trend observed from the collection of patterns is that in most cases, the patterns written by the same author or authors address similar issues. For instance, the set of five patterns presented by Hayden et al in [40] address interaction among agents or systems with dissimilar capabilities. Another example is the set of six patterns presented by Heinze in [41], which all address intention recognition in multiagent systems. The twenty eight patterns analyzed and classified in this section were chosen from the Ninety seven (97) agent-oriented patterns. Sample patterns were randomly picked from the works of twelve out of the thirteen different authors in the collection to make up the twenty eight patterns that are analyzed and classified in this section.

The result of the analysis and classification of the twenty eight patterns is as follows: four of the patterns belong to the Organizational category and one to the Interactional category at the Agent-Oriented Analysis level of the Two Way classification scheme. One pattern belongs to the Definitional category, three to the Structural and five to the Interactional category at the MAS Architecture level. One pattern belongs to the
Table 5.5: Ninety Seven Existing Agent-Oriented Patterns

<table>
<thead>
<tr>
<th>Agent-Oriented Pattern</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broker, Embassy, Mediator, Monitor, Wrapper</td>
<td>[40]</td>
</tr>
<tr>
<td>Synchronizer, Environment Mediated Communication, Updating Shared State, Behavior-based Decision</td>
<td>[117]</td>
</tr>
<tr>
<td>Reactive Agent, Deliberative Agent, Opportunistic Agent, Interface Agent, Intention, Prioritizer, Adaptable Active Object, Message Forwarder, Plan as Command, Plan and Intention Factory, Conversation, Facilitator, Agent Proxy, Protocol, Emergent Society, Clone, Remote Configurator, Broker, Migration Thread Factory, Agent Builder, Layer Linker</td>
<td>[61]</td>
</tr>
<tr>
<td>Basic Mobility, Itinerary, Star-shaped Movement, Branching, Contract Net Protocol, Direct Interaction, Mediation, Traveling, Task, Interaction, Basic Action (collection), Plan Generation/Execution (collection), Security/Safety (collection)</td>
<td>[126]</td>
</tr>
<tr>
<td>Booking, Subscription, Call-For-Proposals, Bidding, Matchmaker</td>
<td>[25]</td>
</tr>
<tr>
<td>Sentinel Agent Behaviour, Simplified Sentinel Agent Behaviour</td>
<td>[14]</td>
</tr>
<tr>
<td>InteRRaP</td>
<td>[77]</td>
</tr>
<tr>
<td>Organisation scheme, Protocols, Marks, Influences, BDI architecture, Vertical architecture, Horizontal architecture, Recursive architecture, Iniquity, Discretisation, Physical entity</td>
<td>[116]</td>
</tr>
<tr>
<td>Agent Society, Agent as Delegates, Agent as Mediator, Common Vocabulary, User Agent, Task Agent</td>
<td>[128]</td>
</tr>
<tr>
<td>Itinerary, Forwarding, Ticket, Master-Slave, Plan, Meeting, Locker, Messenger, Facilitator, Organized Group</td>
<td>[2]</td>
</tr>
<tr>
<td>Structure in 5, Joint Venture, Pyramid, Flat Structure, Auction/Bidding, Takeover, Arm's length Agreement, Hierarchical Contracting, Vertical Integration, Co-optation</td>
<td>[65]</td>
</tr>
<tr>
<td>Hybrid Recognizer, Sense-and-Infer, Assisted Sense and Infer, Ecological Recognizer, Assisted Ecological Recognizer, Clairvoyant</td>
<td>[41]</td>
</tr>
<tr>
<td>Basic Negotiating Agent</td>
<td>[4]</td>
</tr>
<tr>
<td>ADEPT Agent Architecture</td>
<td>[52]</td>
</tr>
<tr>
<td>SCIC Agent Architecture</td>
<td>[113]</td>
</tr>
</tbody>
</table>

Structural category, two to the Interactional and six to the Strategic category at the Agent Internal Architecture level. At the MAS Realization level, five patterns are classified under the Others category since the categories of this level are not defined in this thesis. There are thirty entries in the categories of the classification.
scheme because two patterns are each classified under two categories. The result of
the analysis and classification of the twenty eight patterns is presented in table 5.6.
Note that there are thirty (30) entries in the table because two of the patterns appear
twice, as will be explained when the patterns are discussed. Some of the patterns in
the collection of Ninety seven patterns, for example Common Vocabulary could
not be classified into the classification scheme. The case of Common Vocabulary
and some other observations from the analyses of patterns presented in the
following subsections are discussed in section 5.5.

The following analysis and classification of the patterns are documented in line
with the five activities of the classification process described in section 5.3. The
examinations of the patterns analyzed are presented as descriptions of their
Problems and Solutions. The patterns’ Problems and Solutions are described based
on their literature while maintaining the originality of the materials as much as
possible. The identification of the patterns’ level attributes is documented in the
level attributes table 5.7. The determination of the patterns’ levels is presented
under the Level attributes analysis segment of the documentation for each pattern.
The identification of the patterns’ category attributes is documented in the category
attributes tables 5.8, 5.9 and 5.10. The determination of the patterns’ categories is
presented under the Category attributes analysis segment of the documentation for
each pattern. The analysis and classification of the patterns are arranged in the
following sub sections according to the levels of the Two Way classification
scheme.

5.4.1. Agent-Oriented Analysis level

Four agent-oriented patterns are analyzed and classified at this level. There is a
family of three patterns by the same authors and one individual pattern.
Table 5.6: Classification of 28 Agent-Oriented Patterns

<table>
<thead>
<tr>
<th>Agent-Oriented Analysis (Level 1)</th>
<th>Organizational</th>
<th>Interactional</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent as Mediator</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiagent System Architecture (Level 2)</td>
<td>Definitional</td>
<td>Structural</td>
</tr>
<tr>
<td>Agent as Delegate</td>
<td>Star-shaped Movement</td>
<td>Branching</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Agent Internal Architecture (Level 3)</td>
<td>Structural</td>
<td>Interactional</td>
</tr>
<tr>
<td>Basic Negotiating Agent</td>
<td>Sense and Infer</td>
<td>Ecological Recognizer</td>
</tr>
<tr>
<td>InteRRaP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAS Realization (Level 4)</td>
<td>Definitional (Technology)</td>
<td>Others</td>
</tr>
<tr>
<td>Synchronizer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Environment Mediated Communication</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Behaviour-Based Decision</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Master-Slave</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meeting</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.4.1.1. Structure-in-5, Joint Venture and Pyramid Patterns
These three patterns are chosen from a set of ten patterns written by the same authors. These patterns address the same kind of problem, hence the result of their analysis and classification are combined after describing the three of them.

Name: Structure-in-5

Source: [65]
Problem: The structure of a system from which multiagent system is to be developed has middle management with a high degree of autonomy and minimal control from top management. Top management basically provides strategic direction to the middle management. The middle management coordinates the activities of the lower level roles in the system in order to achieve the strategic goals of the system. How should the existing system be analyzed in order to define an agent based virtual organization, that is, a role-goal structure which can be readily converted to multiagent system architecture?

Solution: Structure-in-5 provides an analysis model with three levels. The strategic executive roles which constitute the top of the organization form the apex of the structure. The goal of this apex is strategic management. The roles at the middle level are the Coordination, Middle Agent technology and Support roles. At the lowest level are the Operational core roles, that is, the roles that will carry out the basic tasks and operations that are required to achieve the overall goal of the system. The middle level roles are related to one another by the goal dependencies of Control and Logistics while the operational core roles are related to the middle level roles by task dependencies. Figure 5.2 shows the representation of the Structure-in-5 pattern.

![Figure 5.2: The Structure-in-5 pattern](image)
Name: *Joint Venture*

Source: [65]

**Problem:** A multiagent system is to be developed for the partial combination of the structure and resources of two or more existing systems to achieve a common goal. There is the need for a central control for coordinating operational activities and managing the sharing or resources, knowledge and responsibilities while the existing systems retain their individual identities. How should the existing systems be analyzed in order to define an agent based virtual organization to achieve the common goal, that is, a role-goal structure which can be readily converted to multiagent system architecture that will implement the intended system?

**Solution:** The Joint Venture pattern models the virtual organization of collaborating existing systems around a Joint Management role. The Joint Management role reports to the Principal Partners role (Joint Venture) who delegates authority for operational running of the partnership to the Joint Management role. A Contractual Agreement goal which defines the objective to be achieved connects the Joint Venture to the Principal Partners. Also, Resource Exchange connects the Principal Partners to the Joint Venture. The Joint Management role is expected to Add Value (soft goal) to each the Principal Partners. The Principal Partners are connected to each other for Knowledge Sharing (soft goal) and they are connected to Secondary Partner roles for support and service provision. Figure 5.3 is a representation of the Joint Venture pattern.
Name: *Pyramid*

Source: [65]

**Problem:** The structure of a system from which multiagent system is to be developed is based on the hierarchical authority structure. Roles at lower levels have minimal autonomy and depend on higher level roles. Majority of key operational decisions are made at the higher levels and are forwarded down through the middle management roles to the operational roles. How should the existing system be analyzed in order to define an agent based virtual organization to achieve the common goal, that is, a role-goal structure which can be readily converted to multiagent system architecture that will implement the system?

**Solution:** The Pyramid pattern models a virtual organization with three levels. The model is based on direct supervision and control from the top. The Apex roles delegate responsibilities and strategic authority to the middle level roles which are Managers and Supervisors. Managers and Supervisors are mainly intermediate
roles with the goal of routing decisions from the Apex roles to the Operations roles. Operations roles carry out the basic operational tasks of the system and rely on the middle level roles for monitoring, coordination, conflict resolution in addition to routing of strategic decisions. The level of autonomy of the roles decreases significantly from top down. Figure 5.4 is the representation of the Pyramid pattern.

![Figure 5.4: The Pyramid pattern](image)

**Level Attributes Analysis: Structure-in-5, Joint Venture and Pyramid Patterns**

The Structure-in-5, Joint Venture and Pyramid patterns were analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Structure-in-5, Joint Venture and Pyramid patterns are Roles/Multiple Roles and Organization of Roles. Note that both the Problems and Solutions for these patterns mention roles explicitly. These two attributes represent two out of the four (2/4) attributes of the Agent-Oriented Analysis level of the Two Way classification scheme. The other two attributes of the Agent-Oriented Analysis level, Application Domain and Interaction are not featured in these patterns. Structure-in-5, Joint
V Venture and Pyramid do not have attributes that belong to another level. Therefore, it is concluded that these patterns belong to the Agent-Oriented Analysis level of the Two Way classification scheme.

**Category Attributes Analysis: Structure-in-5, Joint Venture and Pyramid Patterns**

The Structure-in-5, Joint Venture and Pyramid patterns were analyzed using the category attribute table (table 5.8) of the Agent-Oriented Analysis level to determine the category that the pattern belongs to at this level. From this analysis, the set of category attributes that are present in the Structure-in-5, Joint Venture and Pyramid patterns include Roles, Coordination/Control, Association/Relationship, Goals and Responsibility. These attributes represent all the attributes of the Organizational category of the Agent-Oriented Analysis level of the Two Way classification scheme. Structure-in-5, Joint Venture and Pyramid do not have other attributes that belong to another category at this level of abstraction.

The intention of the authors of the Structure-in-5, Joint Venture and Pyramid patterns is to narrow the gap between requirements modeling and software architecture. From the analysis of these patterns using the level and category attribute tables, the patterns feature the components of organizations and how these components relate to achieve the organizations’ objectives. These components include roles, goals, and responsibilities but do not include agents or a structural arrangement of agents. The patterns provide a way to analyze the behaviour and/or structure of an organization in terms of the components of the organization, their relationships and coordination in particular context. This representation of the organization can be built on to analyze the interactions among the components. The resultant organization model can then be used as a basis for the design of a multiagent system architecture that should implement the virtual organization. Hence, Structure-in-5, Joint Venture and Pyramid are most appropriately Agent-Oriented Analysis patterns that serve as bases for the design of multiagent system architectures in particular contexts. The result of this analysis is further
strengthened by another work by the same authors using these patterns for early requirements analysis [67].

5.4.1.2. Agent Society Pattern
This pattern with the Agent as Delegate and Agent as Mediator patterns are written by the same authors. The analyses of Agent as Mediator and Agent as Delegate patterns are presented in subsections 5.4.2.1 and 5.4.2.5 respectively.

Name: Agent Society

Source: [128]

Problem: A software system is to be developed for a domain. The domain data, control, knowledge, and/or resources are decentralized. The domain is made up of independent entities or components that cooperate to achieve a common goal OR the components should collaborate to achieve their individual goals. How should the software system be modeled to achieve the requirements of this domain OR to achieve the features/characteristics of the domain?

Solution: Analyze the domain in terms of goals and the roles that achieve these goals. Model the application as a society of agents. The agents represent the roles. These agents will interact/cooperate to achieve the goals of the system. Therefore, the society is the nature/structure of the relationship of the agents that will achieve the goals of the system. An agent is a computational entity that is situated in an environment, and that is capable of autonomous, reactive, proactive and social behaviours. The notion of autonomy in agents describes the ability of an agent to govern its own actions, that is, the ability to start, direct and end its operations with minimal or no human/other system intervention and control. Proactive behaviour in agents means that agents do not only act in response to some prompting, that is, agents take initiatives. Reactive behaviour means agents are capable of capturing changes in the environment and decide on how to respond to such changes in a manner consistent with its goals. Social behaviour in agents refers to the context
sensitive interaction among agents. Society of agents develops by both design and evolution. Agent society develops by design when the interactions among agents are constrained by a set of conversation protocols. Agent society develops by evolution when the structure of the society is defined by the interactions that exist among the agents in the system.

**Level Attributes Analysis: Agent Society Pattern**

The Agent Society pattern is analyzed using the level attribute table (table 5.7), in order to determine the level of abstraction that the pattern belongs to. From this analysis, the level attributes that are present in the Agent Society pattern are Roles/Multiple Roles, Multiple Agents, Interaction and Roles – Agents. Two of these attributes, Roles/Multiple Roles and Interaction represent two out of the four attributes of the Agent-Oriented Analysis level while the other two, Multiple Agents and Roles and Agents represent two out of the seven attributes of the MAS Architecture level. The attribute distribution of two out of four belonging to two levels suggests that this pattern relates to two levels of abstractions. However, the relative representation of the attributes at each level, that is two out of four as against two out of seven, makes the pattern tend more towards belonging to the Agent-Oriented Analysis level. Therefore, it is concluded that this pattern belong to the Agent-Oriented Analysis level of the Two Way classification scheme.

**Category Attributes Analysis: Agent Society Pattern**

The Agent Society pattern is analyzed using the category attribute table (table 5.8) of the Agent-Oriented Analysis level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Agent Society pattern are Roles, Goals, and Responsibilities. These attributes represent three out of the five attributes of the Organizational category of the Agent-Oriented Analysis level. It is worthy of note that this pattern does not have Coordination/Control or Association/Relationship as some of its attributes. Therefore this pattern cannot be adapted to model specific organization of roles in the analysis of a particular system to be developed.
From further study of the pattern and its attributes, the pattern does not present particular structures for modeling either role organizations or multiagent system architectures. It presents the considerations for deciding to use agent technology, it presents the conditions that should be satisfied for a system to be conceptualized as a multiagent system and then developed as one. It also provides the different concepts that define agent technology. It describes agents, characteristics of agent, roles, role-agent mappings, society, and approaches for modeling agent systems. This pattern is therefore a general overview of applicability and concepts of agent technology described as a software pattern. However, this specification of the requirements for the application of agent technology in the framework of a software pattern, that is description of context and forces, facilitates the comprehension of agent technology applicability and concepts by non agent researchers. On the basis of the foregoing analysis, it is concluded that Agent Society pattern belongs to the Organizational category of the Agent-Oriented Analysis level of the Two Way classification scheme.

5.4.2. Multiagent System Architecture level

Nine agent-oriented patterns are analyzed and classified at this level. There are two families of three patterns each by two sets of authors. The other three patterns are individual patterns from different authors.

5.4.2.1. Agent as Delegate Pattern

Name: Agent as Delegate

Source: [128]

Problem: There is a user role in a system that is to be modeled as a multiagent system. This role involves carrying out both non trivial operational tasks and deciding on the execution of the tasks using user information. The user information should not be included in the execution of the operational tasks for security and
confidentiality reasons. How should the role of this user be converted to agents in an agent based system while maintaining confidentiality of user information?

Solution: This pattern describes an approach for translating a role into agents. It prescribes translating a complex user with sensitive data into two types of agents which are User Agent and Task Agents. The pattern specifies the relationship and control that should exist between these two types of agents. The pattern prescribes translating a role with sensitive data with non trivial responsibilities into an Administration Agent and an Operations Agent.

Divide the role of the user into a User agent and a Task agent. The User agent is the delegating agent while the Task agent is the executing agent and will represent the user in different task contexts. The User agent holds the information about the user, user preferences and goals. That is the User agent builds a user profile. The User agent also, updates the user profile as it learns about changes to the user’s preferences. The User agent delegates authority to the Task agent based on the user’s preferences and goals per time and controls the Task agent’s degree of autonomy. The User agent also controls how much information about the user is made available to the Task agent. The number of Task agents that the user will have depends on the complexity and diversity of the tasks to be carried out. See Figure 5.5 for the diagram of the Agent as Delegate pattern.

![Figure 5.5: Agent as Delegate Pattern](image)

Figure 5.5: Agent as Delegate Pattern
**Level Attributes Analysis: Agent as Delegate Pattern**

The Agent as Delegate pattern is analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the level attributes that are present in the Agent as Delegate pattern are Roles and Agents, Multiple Agents and Interaction. These attributes represent three out of the seven attributes of the MAS Architecture level. Therefore, it is concluded that Agent as Delegate pattern belongs to the MAS Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Agent as Delegate Pattern**

The Agent as Delegate pattern is analyzed using the category attribute table (table 5.9) of the MAS Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Agent as Delegate pattern are Division Rules and Responsibilities. The Division Rules in the pattern are implicit and can be deduced from the details of the solution of the pattern. These attributes represent two out of the four attributes of the Definitional category of the MAS Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Definitional category of the MAS Architecture level.

### 5.4.2.2. Star-Shaped Movement, Branching and Itinerary Patterns

These three patterns are written by the same authors and relate to the same kind of problem, hence the result of their analysis and classification are combined after describing the three of them.

**Name:** Star-Shaped Movement

**Source:** [126]

**Problem:** A mobile agent has a goal to achieve. However, the resources for achieving this goal are distributed across different hosts on a network. The hosts
with the resources are predetermined and are relatively stable. The network bandwidth is also relatively broad. How should the agent efficiently navigate among the different hosts in the multiagent system to achieve its goal?

Solution: In the Star-Shaped Movement pattern, the mobile agent moves from the base host to a destination host holding some of the resources, executes its task at the destination host and returns to the base host. It moves to another destination host and returns to the base host after executing its task there. It navigates all the required hosts on the network by the base host to destination host and back to base host movement until the goal is achieved. See Figure 5.6 for a representation of the Star-Shaped Movement pattern.

![Figure 5.6: Star-Shaped Movement Pattern](image)

Name: Branching

Source: [126]

Problem: A mobile agent has to choose one of a number of results from execution of tasks carried out at different hosts on a network or an agent needs to combine the results of the execution of tasks carried out at different hosts on a network. The network bandwidth is relatively broad; however, the volume of data to be transported to and from each host where execution is required is large. The hosts
where executions are required are not fixed in advance and are not stable. How should the agent efficiently navigate among the different hosts to achieve its goal?

**Solution:** The agent generates as many copies of itself as the number of hosts it intends to visit. Each copy then moves to a host, executes its task there, using the resources at the host and returns to the base host with the results. When all the copies have returned with their results, one of the results is chosen or all of them are merged depending on the goal of the agent. See Figure 5.7 for a representation of the Branching pattern.

**Figure 5.7: Branching Pattern**

**Name:** Itinerary

**Source:** [126]

**Problem:** A mobile agent has a goal to achieve with the resources for achieving this goal distributed across different hosts on a network. The volume of data required for executing the tasks at the different hosts is minimal. The hosts with the resources are predetermined and are relatively stable. However, the network bandwidth is narrow. How should the agent efficiently navigate among the different hosts in the multiagent system to achieve its goal?
**Solution:** The agent moves to the first destination host and executes its task there. It moves directly on to the next host carrying the results of the execution at the first host if there are any to carry. The agent moves on from one host to the next until it has navigated all the destination hosts. Then it returns to the base host. If the volume of data is considerable, then this pattern becomes inapplicable as this would result in congestion on the narrow bandwidth of the network. See Figure 5.8 for a representation of the Itinerary pattern.

![Figure 5.8: Itinerary Pattern](image)

**Level Attributes Analysis: Star-shaped Movement, Branching and Itinerary Patterns**

The Star-shaped Movement, Branching and Itinerary patterns are analyzed using the level attributes table in order to determine the level of abstraction that these patterns belong to. From these analyses, the set of level attributes that are present in these patterns are Infrastructure and Single Agent. Infrastructure is a MAS Architecture level attribute while Single Agent is an Agent Internal Architecture attribute. Although these patterns relate to a single agent, that is the mobile agent, the considerations for deciding on the mobility strategy that the agent will adopt are based on the communication/hardware infrastructure of the multiagent system. As a result, the decision on the mobility strategy has to be made at the MAS Architecture level. Also, these patterns do not define the internal components or the arrangement of the internal components of an agent. Rather, they define how a mobile agent will move amongst the different components of the multiagent
system. Based on this analysis, it is concluded that these patterns belong to the MAS Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Star-shaped Movement, Branching and Itinerary Patterns**

The Star-shaped Movement, Branching and Itinerary patterns are analyzed using the category attribute table (table 5.9) of the MAS Architecture level in order to determine the category that the patterns belong to at this level. From this analysis, the category attributes that are present in the Star-shaped pattern are Bandwidth, Host Stability and Data Volume. These attributes represent three out of the six attributes of the Structural category of the MAS Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Structural category of the MAS Architecture level.

**5.4.2.3. Contract Net Protocol Pattern**

The Contract Net Protocol is well known and applied, therefore, it can be regarded as an agent-oriented pattern as presented in [77].

**Name:** Contract Net

**Source:** [122, 32]

**Problem:** An agent (initiator) must perform a task using one or more other agents (participants) and it must ensure the optimization of an attribute of the task for instance time to completion, or cost.

**Solution:** The initiator requests bids of potential participants. The participants respond by either sending a bid or refusing to send a bid within a stipulated time frame. At the expiry of the stipulated time, the initiator selects one or more agents to perform the task based on the evaluation of their bids while rejecting the bids of
others. The agents selected to execute the task then send the results to the initiator at the completion of the task. See Figure 5.9 for a representation of the Contract Net pattern.

Figure 5.9: The FIPA Contract Net Protocol

**Level Attributes Analysis: Contract Net Protocol Pattern**

The Contract Net Protocol pattern is analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Contract Net Protocol pattern are Multiple Agents, Interaction, Arrangement of Agents and Constraints. These attributes represent four out of the seven attributes of the MAS Architecture level. Even though Interaction, which is also an attribute of the Agent-Oriented Analysis level, is one of the attributes of the Contract Net Protocol pattern, it relates to MAS Architecture since the pattern is about agents and not roles. Therefore, it is concluded that Contract Net Protocol pattern belongs to the MAS Architecture level of the Two Way classification scheme.
**Category Attributes Analysis: Contract Net Protocol Pattern**

The Contract Net Protocol pattern is analyzed using the category attribute table (table 5.9) of the MAS Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Contract Net Protocol pattern are Control Flow, Message Exchange Rules, and Performatives/Message Format. These attributes represent three out of the five attributes of the Interactional category of the MAS Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Interactional category of the MAS Architecture level.

**5.4.2.4. Embassy, Monitor and Wrapper Patterns**

These three patterns are written by the same authors and relate to the same kind of problem, hence the result of their analysis and classification are combined after describing the three of them.

**Name:** Embassy

**Source:** [40]

**Problem:** The World Wide Web as an agent environment comprises many different domains. Agents in domains which may be completely different, in communication languages and content formats may wish to interact occasionally. How should interaction among agents belonging to such different domains be achieved?

**Solution:** The Embassy pattern introduces an agent to represent multiagent system for the purpose of interaction with other heterogeneous domains. The introduced agent is called the Embassy agent through which foreign agents may communicate
with and gain access to the multiagent system. The Embassy agent provides access control to the multiagent system, allows translation of languages and establishment of interaction protocol where the foreign agent is heterogeneous. Some KQML performatives that are useful for this pattern include request-access, granted, deny, translate and bye. See Figure 5.10 for a representation of the Embassy pattern.

![Figure 5.10: The Embassy Pattern]

**Name:** Monitor

**Source:** [40]

**Problem:** Agents in multiagent system need to be updated about the changes to the state of a non agent entity in the system. Different agents need to keep up with the changes to the non agent entity. The changes to the state of the non agent entity that these agents want to know about are different. How should interaction be structured to achieve efficient and relevant access by an undefined number of agents to changes in the state of a non agent entity in multiagent system?

**Solution:** The Monitor pattern introduces a Monitor agent to facilitate and coordinate the interaction of the agents in the multiagent system with the non agent entity. The agents that need to receive updates about the changes to the entity...
notify the Monitor agent by subscribing for notification of particular changes to the entity. The Monitor agent then requests notification of the entity. When there are changes in the state of the entity, it notifies the Monitor agent. The Monitor agent in turn notifies the subscriber agent that the change is relevant to. See Figure 5.11 for a representation of the Monitor pattern.

![Figure 5.11: The Monitor Pattern](image)

**Name:** Wrapper

**Source:** [40]

**Problem:** The agents in a multiagent system need to interact with a legacy system because the legacy system is expensive to convert to multiagent system or it is outside of the control of the developers of the multiagent system or the legacy system will be converted to a multiagent system only at a later date. How should the agents in multiagent system interact with a legacy system without redundancy?

**Solution:** In this pattern, a Wrapper agent is introduced to serve as the interface between the legacy system and the agents in the multiagent system. The Wrapper agent receives requests from the agents using the appropriate multiagent system communication language and interaction protocol. The Wrapper agent then maps these requests to legacy system code. It also maps legacy systems responses back to agent communication language for communication with the agents. With this
pattern, the agents in the multiagent system interact indirectly with the legacy system using agent communication languages and protocols while the legacy system is not re implemented and not directly coupled to each agent. See Figure 5.12 for a representation of the Wrapper pattern.

![Figure 5.12: The Wrapper Pattern](image)

**Level Attributes Analysis: Embassy, Monitor and Wrapper Patterns**

The Embassy, Monitor and Wrapper patterns were analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Embassy, Monitor and Wrapper patterns are Multiple Agents, Interaction, Arrangement of Agents and Constraints. These attributes represent four out of the seven attributes of the MAS Architecture level. Therefore, it is concluded that these patterns belong to the MAS Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Embassy, Monitor and Wrapper Patterns**

The Embassy, Monitor and Wrapper patterns were analyzed using the category attribute table (table 5.9) of the MAS Architecture level in order to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Embassy, Monitor and Wrapper patterns are Control Flow, Introduced Agents, and Message Exchange Rules. These attributes represent three out of the five attributes of the Interactional category of the MAS.
Architecture level. In addition to these three attributes, the Embassy pattern also has the Performatives/Message Format and Ontology attributes making the Embassy pattern’s attributes five out of the five attributes of the Interactional category of the MAS Architecture level. These patterns have the Introduced Agents attribute since the Embassy, Monitor and Wrapper agents in the solution of the patterns are introduced specifically for the purpose of interaction. The patterns do not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the Embassy, Monitor and Wrapper patterns belong to the Interactional category of the MAS Architecture level.

5.4.2.5. Agent as Mediator Pattern

Name: Agent as Mediator

Source: [128]

Problem: An agent in a multiagent system willing to interact does not have information about other agents within the multiagent system with which it can interact. The agent does not know which agents exist or are available at a particular time, does not know the capabilities of the agents, does not know the nature and rationality of the agents, does not know the interaction protocol and ontology of the other agents and desires to have safe and secure interaction with some of them. How should the interaction between such agents that are aware of themselves be facilitated?

Solution: Introduce a Mediator agent to facilitate interaction among non familiar agents. Analogies of mediator include translators, directories, market makers, and rating services. The Mediator agent holds information about the different agents in the system. It holds information relating to their availability, capabilities, and rationality/credibility. The Mediator agent may just serve to connect agents that are willing to interact or it may serve to coordinate the interaction between two agents for example where there are differences in the agents representation of concepts or
in order to ensure security of information involved in the interactions. Mediator agent is a relatively general term as the mediation can be in different forms. Therefore, different types of Mediator agents can be defined depending on the context of the multiagent system and the agents to be developed. Other patterns that are based on the principle of mediation include Embassy (Figure 5.10), Broker [128], and Matchmaker [25] among others.

**Level Attributes Analysis: Agent as Mediator Pattern**
The Agent as Mediator pattern is analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Agent as Mediator pattern are Multiple Agents, Interaction and Constraint. These attributes represent three out of the seven attributes of the MAS Architecture level. Therefore, it is conclude that Agent as Mediator pattern belongs to the MAS Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Agent as Mediator Pattern**
The Agent as Mediator pattern is analyzed using the category attribute table (table 5.9) of the MAS Architecture level to determine the category that the pattern belongs to at this level. From this analysis, there is only one category attribute that is present in the Agent as Mediator pattern and this is Introduced Agents. This attribute is one of the attributes of the Interactional category at the MAS Architecture level. However, this singular attribute is not representative of the category.

Analyzing this pattern with the category attribute table of the Agent-Oriented Analysis level yields the following result. The category attributes of the Agent-Oriented Analysis level that are present in the pattern include Interaction Roles, Distribution and Accessibility. From the analysis of this pattern, the interaction roles defined by the pattern are Task, Mediator and Resource. The pattern describes different possible distributions of these roles such as Task-Mediator-Task and Task-Mediator-Resource. Also, the pattern comments on different accessibility
situations. These attributes represent two out of the six attributes of the Interactional category of the Agent-Oriented Analysis level. From this analysis, it can be seen that this pattern is a generic case of Mediation in agent interaction. It does not stipulate a particular interaction protocol for a specific type of agent interaction.

These analyses reveal that this pattern is not precise enough to belong to a specific category at a level of abstraction. The pattern does not proffer a solution that is adaptable to agent-oriented software engineering projects. It rather discusses certain issues with interaction in agent-based systems. Hence, to an extent, the pattern is an Interactional pattern at the Agent-Oriented Analysis level while to another extent; the pattern is an Interactional pattern at the MAS Architecture level.

5.4.3. Agent Internal Architecture level

This level of abstraction is the most studied area in agent technology and the starting point of a lot of research. The view taken in this thesis is to distill patterns from some of the works in this area even though the authors may not describe their work in such terms. No claim is made to be exhaustive in the coverage of the extent of work in this area or about which of these works should be distilled as patterns. However, an example of a pattern distilled from previous works is Basic Negotiation Agent pattern which is analyzed in this subsection. Two other examples of such are the Process Agent Architecture and the SCIC Agent Architecture patterns described in section 4.7.2 of Chapter Four.

InteRRaP and BDI were not originally described as agent-oriented patterns, however some other work has regarded them as agent-oriented patterns. InteRRaP is described in [77] as an agent-oriented pattern while [116] describes BDI as an agent-oriented pattern.

The patterns at this level are generally more complex than the patterns at the other levels partly because many of the deeper concepts of agent technology such as
autonomy, reactivity, proactive behaviour, intelligence, and deliberation are featured at this level.

Ten patterns are analyzed and classified at this level. There are two families of patterns at this level. One has two patterns by the same author and another has four patterns by the same authors. The other four patterns are by different authors.

5.4.3.1. Basic Negotiating Agent Architecture Pattern

Name: Basic Negotiating Agent

Source: [4]

Problem: An agent is to be developed to participate in auctions or buyer-seller negotiations. The agent is required to negotiate with other agents having competing claims on scarce resources and come to an agreement on the division of the scarce resources. The agent is required to receive, evaluate and make proposals in agreement with the agent’s goals. The agent should carry out its activities in compliance with the rules guiding the outcome of the auction. How should the agent be designed to be able to address the requirements?

Solution: The Basic Negotiating Agent Architecture is composed of an Illocution Interpreter, Locution Generator, Proposals History, Negotiation Protocols, Protocol Reasoner, Proposal Evaluator, Opponent Model, Mental Attitudes, Environment Model and Response Generator. The Illocution Interpreter and Locution Generator represent the agent’s interface with its environment. Messages come in through the Illocution Interpreter and messages are constructed and sent out through the Locution Generator. The Proposals History, Negotiation Protocols, Opponent Model, Mental Attitudes and Environment Model represent the knowledge component of the architecture. The Protocol Reasoner assesses the validity of the proposals received in the light of the negotiation protocol. The Proposal Evaluator evaluates the proposals received against the goals of the agent and sends the result
of its evaluation to the Response Generator to prepare an appropriate response to the proposal. The flows of information and control among the different components have been designed to effectively handle the negotiation requirements of the agent.

![Figure 5.13: Basic Negotiating Agent Architecture](image)

**Level Attributes Analysis: Basic Negotiating Agent Architecture pattern**

The Basic Negotiating Agent Architecture pattern is analyzed using the level attributes table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Basic Negotiating Agent Architecture pattern are Single Agent, Agent Internal Components and Arrangement of Agent Internal Components. These attributes represent three out of the four attributes of the Agent Internal Architecture level. Therefore, it is concluded that Basic Negotiating Agent Architecture pattern belongs to the Agent Internal Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Basic Negotiating Agent Architecture pattern**

The Basic Negotiating Agent Architecture pattern is analyzed using the category attributes table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Basic Negotiating Agent Architecture pattern are Knowledge Component, Interaction Component, Environmental
Interface Component, and Geometry. These attributes represent four out of the five attributes of the Structural category of the Agent Internal Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Structural category of the Agent Internal Architecture level.

5.4.3.2. InteRRaP Agent Architecture Pattern

InteRRaP is analyzed and classified in this subsection because it is well known and it features many of the deeper concepts of agent technology. There is a whole book about the use of InteRRaP, only enough information required for the analysis and classification of the pattern is presented here.

**Name:** InteRRaP

**Source:** [87, 88]

**Problem:** The agent architecture requires distinct representation of the reactive, pro-active and social interaction related functionalities of the agent. There should not be conflicts in the discharge of the duties of the separate functional components. An efficient representation of the world relative to each functional component is required. How should the agent be designed to be able to address the requirements?

**Solution:** The components are represented in a two pass layered structure with a control pass and a knowledge representation pass. The control layers from lowest to highest deal with reactive, pro-active/everyday planning and social interactions. There are three knowledge representation layers as well mapping directly to each of the control layers. The layers interact with each other and use a common world interface for perception and action in order to avoid the overhead of a supervisory control component. Figure 5.14 is a representation of the InteRRaP architecture.
**Figure 5.14: InteRRaP Agent Architecture**

**Level Attributes Analysis: InteRRaP Agent Architecture Pattern**
The InteRRaP pattern is analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the InteRRaP pattern are Single Agent, Agent Internal Components and Arrangement of Agent Internal Components. This set of attributes present in the InteRRaP pattern represent three out the four attributes of the Agent Internal Architecture level. Therefore, it is concluded that the pattern belongs to the Agent Internal Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: InteRRaP Agent Architecture Pattern**
The InteRRaP pattern is analyzed using the category attribute table (table 5.10), to determine the category that the pattern belongs to at the Agent Internal Architecture level of the Two Way classification. From this analysis, the set of category attributes that are present in the InteRRaP pattern are Knowledge, Strategy and Environmental Interface components, Geometry, Reactivity, Deliberation and Knowledge Management attributes. The set of Knowledge, Strategy, Environmental Interface components, and Geometry attributes represent four out of the five attributes of the Structural category while the set of Reactivity, Deliberation and Knowledge Management represent three out of the seven attributes of the Strategic category.
The result of this analysis shows that the InteRRaP pattern has features that qualify it to belong to two of the categories at the Agent Internal Architecture level. These are Structural and Strategic categories. This reveals that the InteRRaP pattern addresses different types of problem of agent internal architecture, which are structural and strategic according to the Two Way pattern classification. This result indicates that different segments of the InteRRaP pattern represent different categories of patterns. Therefore, the InteRRaP pattern can be seen as a combination of at least two patterns. Based on this analysis, the InteRRaP pattern may be described as a combination of patterns. This therefore implies that, the Structural segment of the InteRRaP pattern can be applied as a solution to structural problems in the design of agent internal architectures independent of the Strategic segment of the pattern and vise versa for the Strategic segment as well.

### 5.4.3.3. Sense-and-Infer and Ecological Recognizer Patterns

These two patterns are part of six patterns written for intention recognition in intelligent agent systems by Heinze’s in his PhD thesis [41]. The result of their analysis and classification are combined after describing them.

**Name:** Sense-and-Infer

**Source:** [41]

**Problem:** An intelligent agent (recognizing agent) must recognize the intention of other agents (intending agents) where other agents are human users of the system or there is the internal mental states of other agents are not accessible. Also, the set of state data which is used to infer the intending agent’s intentions is relatively small. How should the agent be designed to overcoming these limitations and achieve intention recognition?

**Solution:** In this domain, the basic architecture of an agent is made up of three components which are Perception, Reasoning and Action. To address this problem, introduce a Sensor module, and Action Inferencer and an Intention Inferencer
module to the architecture of the recognizing agent. The Sensor module is part of
the Perception component while the Action Inferencer and Intention Inferencer
modules are part of the Reasoning component of the architecture. The Sensor
module senses the state data and transforms it as necessary before sending it to the
Action Inferencer module. The Action Inferencer processes the transformed state
data to recognize actions of the intending agent. The recognized actions are passed
to the Intention Inferencer which processes them and recognizes the intentions of
the intending agent. Figure 5.15 is a representation of the Sense-and-Infer pattern.

![Figure 5.15: Sense-and-Infer Pattern](image)

**Name:** *Ecological Recogniser*

**Source:** [41]

**Problem:** An intelligent agent (recognizing agent) must recognize the intention of
other agents (intending agents) but has access only to the state data. Furthermore,
the state data is complex and the intentions to be recognized are difficult to
describe by sets of rules. How should the agent be designed to overcoming these
limitations and achieve intention recognition?
**Solution:** In this domain, the basic architecture of an agent is made up of three components which are Perception, Reasoning and Action. To address this problem, include a recognition module called Pattern Matcher in the architecture of the recognizing agent. This Pattern Matcher is however a part of the Perception component of the agent and it maps the pattern of incoming data stream onto learned, stored or otherwise supplied examples of the recognized intention. Figure 5.16 is a representation of the Ecological Recogniser pattern.

![Figure 5.16: Ecological Recogniser Pattern](image)

**Level Attributes Analysis: Sense-and-Infer and Ecological Recognizer Patterns**

The Sense and Infer and Ecological Recognizer patterns were analyzed using the level attribute table (table 5.7) in order to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Sense and Infer and Ecological Recognizer patterns are Multiple Agents, Agent Internal Components and Arrangement of Agent Components. These patterns feature two agents, an Intending agent and a Recognizing agent. However, these patterns do not specify the arrangement of these agents. The essence of featuring the Intending agent in the patterns is to specify how this agent’s actions affect the environment. Also, stated in these patterns is the fact that the internal state of the intending agent is inconsequential to the patterns’ definition and purpose.
Therefore, Multiple Agents which is an attribute of the MAS Architecture level is not critical to the categorization of this pattern. The other two attributes reflect the emphasis of the pattern which is the internal state of the recognizing agent. These attributes represent two out of the four attributes of the Agent Internal Architecture level. As a result, it is concluded that these patterns belong to the Agent Internal Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Sense-and-Infer and Ecological Recognizer Patterns**

The Sense and Infer and Ecological Recognizer patterns were analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level in order to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Sense and Infer and Ecological Recognizer patterns are Strategy component, Environmental Interface component, Geometry, Perception Handling and Environment. Strategy, Environmental Interface and Geometry represent three out of the five attributes of the Structural category. Perception Handling and Environment represent two out of the six attributes of the Interactional category of the Agent Internal Architecture level.

These patterns present a strategy for perceiving and interpreting the state of the environment based on the actions of the other agents in the multiagent system. Therefore, the structural components in the patterns are for perceiving and interpreting the state of the environment. The other structural components required for plan building, action execution, knowledge management and others are not featured in these patterns. As a result, it is concluded that these patterns belong to the Interactional category of the Agent Internal Architecture category of the Two Way classification scheme. In addition, the structural components and arrangement presented by these patterns could be adapted to solve some other structural problems where the context is similar to that of this pattern.
5.4.3.4. Reactive Agent, Deliberative Agent, Opportunistic Agent and Interface Agent Patterns

These four patterns are part of twenty one patterns presented in an early agent-oriented pattern paper by Kendall et al [61]. The results of their level attribute analyses and classifications are combined after describing them. However, their category attribute analyses are presented separately as they feature different attributes.

These patterns describe generic concepts without addressing specific agent-oriented software engineering problems. It is therefore not clear whether they should be included in these analyses of agent-oriented patterns. However, they have been included for complete coverage of studied patterns. Further comments on this type of patterns are presented in section 5.5.

Source: [61]

Name: Reactive Agent

Problem: An agent operates in an environment that is dynamic and non deterministic and the agent needs to respond to the changes in the environment. The goal of the agent does not require complex knowledge representation. The response to the changes in the environment cannot be pre planned and the response should be timely. How should the agent be designed to be successful in such an environment?

Solution: Design the agent to place emphasis on the reactive behaviour of agents. A predominantly reactive agent will not have an internal symbolic representation of the environment. Therefore, the agent will be designed to have minimal knowledge management system or process. Also, the design of the agent will not have a plan building or plan execution function. The reactive agent will have a perception component that is suited to the environment for gathering sensory input. The
sensory inputs are then responded to using situated action rules. The reasoning pattern of the agent is stimulus/response type of behaviour.

**Name:** Deliberative Agent

**Source:** [61]

**Problem:** An agent operates to achieve goals in an environment that is relatively stable, deterministic and non-episodic. The domain of application has knowledge based solutions that can be identified by experts. The agent needs to take initiatives and proactively take planned actions to be able to achieve its goals. The goals of the agent are well defined. How should the agent be designed to take goal directed actions?

**Solution:** Design the agent to emphasize the proactive behaviour of agents i.e. a deliberative agent. The design of the deliberative agent will feature a strong knowledge management component or belief system. The deliberative agent will have a proper representation of its goals, plans as well as its environment. The deliberative agent will be designed with some degree of intelligence. The reasoning pattern of the agent is goal to plan generation to action execution. Hence, it will take goal directed actions thereby demonstrating initiative or proactive behaviour. Also, a deliberative agent will require a feedback structure by which it can measure the impact of its actions on the environment and consider the need to make updates or amendments to its plans.

**Name:** Opportunistic Agent

**Source:** [61]

**Problem:** An agent has a problem to solve in achieving its goal. The agent faces a situation that is mid way between that of a reactive agent and a deliberative agent in that it has a symbolic representation of the problem that it has to tackle but does
not have a predetermined plan to tackle such problems. However, the agent has knowledge of things that must not be done while addressing the problem. How should the agent be designed to arrive at a solution to the problem and achieve its goal while avoiding the constraints?

**Solution:** Design the agent as an Opportunistic agent. An opportunistic agent is an agent that is designed to reason by working around constraints rather than by building plans. Design the knowledge management component or the belief system of the agent to consist of constraints found in the problem. Since there is no known solution to the problem, the reasoning system of the agent is designed to take opportunity of the solution space that is left once all the constraints are satisfied.

**Name:** *Interface Agent*

**Source:** [61]

**Problem:** An agent has a goal of assisting a human user in using particular complex applications or to find information over the internet. The agent has to capture the individual preferences of any user that it has to assist. That is, the agent does not have a universal predetermined set of preferences to work by. How should the agent be designed to effectively meet the needs of the user?

**Solution:** Design the agent as an Interface agent. An interface agent is an agent that is designed to act according to the preferences of a particular user. The knowledge management component or belief system of the agent consists of parametric user models. The sensors of the agent are designed to monitor user’s actions. Therefore the user’s actions constitute a major part of the changes in the agent’s environment. The agent updates its beliefs by the changes in the user’s actions, desires or preferences. The agent also needs to keep track of changes in the applications that are of interest to the user. The agent’s reasoning system is geared towards determining the user’s preferences per time and coordinating its actions according to these preferences.
Level Attributes Analysis:
Reactive Agent, Deliberative Agent, Opportunistic Agent and Interface Agent Patterns
These patterns are analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the patterns belong to. From these analyses, the set of level attributes that are present in these patterns are Constraints, Single Agent, and Agent Internal Components. These attributes represent three out of the four attributes of the Agent Internal Architecture level. Therefore, it is concluded that these patterns belong to the Agent Internal Architecture level of the Two Way classification scheme.

Category Attributes Analysis: Reactive Agent Pattern
The Reactive Agent pattern is analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Reactive Agent pattern are Reactivity and Knowledge Management. These attributes represent two out of the five attributes of the Strategic category of the Agent Internal Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Strategic category of the Agent Internal Architecture level.

Category Attributes Analysis: Deliberative Agent Pattern
The Deliberative Agent pattern is analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Deliberative Agent pattern are Deliberation and Knowledge Management. These attributes represent two out of the five attributes of the Strategic category of the Agent Internal Architecture. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Strategic category of the Agent Internal Architecture level.
Category Attributes Analysis: Opportunistic Agent Pattern
The Opportunistic Agent pattern is analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Opportunistic Agent pattern are Deliberation and Knowledge Management. These attributes represent two out of the five attributes of the Strategic category of the Agent Internal Architecture. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Strategic category of the Agent Internal Architecture level.

Category Attributes Analysis: Interface Agent Pattern
The Interface Agent pattern is analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that are present in the Interface Agent pattern are Autonomy, Knowledge Management and Adaptability. These attributes represent three out of the five attributes of the Strategic category of the Agent Internal Architecture. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the pattern belongs to the Strategic category of the Agent Internal Architecture level.

5.4.3.5. BDI Pattern
BDI is analyzed and classified here because it is well known and because it features many of the deeper concepts of agent technology.

Name: BDI Architecture

Source: [109, 107, 8]

Problem: An agent’s goals are structured in a hierarchy of goals and sub goals. To achieve a goal, the agent has to make a choice among alternative sub goals of the
goal that is to be achieved. Changes in the state of the environment may render a chosen sub goal incapable of achieving the parent goal. This agent needs to reason about alternative sub goals and actions in a dynamic environment. How should agents that need to reason in response to external events be designed?

**Solution:** BDI models a rational agent using the mental attitudes of beliefs, desires and intentions. The agent’s knowledge is captured in beliefs such as the information about the agent’s current environment. Desires represent the agent’s current possible courses of actions i.e. the different goals the agent could pursue. An intention could be described as the goal or sub goal that the agent commits to at a particular point in time.

The BDI architecture takes a perceptual input from the environment and feeds this into the belief revision function for necessary updates to the belief base. Updates to the beliefs are passed to the option generation and filter functions. The updates passed to these two functions are combined with the existing state of the intentions to update both the desires and intentions as necessary. Current intentions are fed into the action selection function to define an appropriate plan of action for achieving these intentions. The process of reasoning in BDI is represented in Figure 5.17. BDI attempts to balance both reactive and deliberative behaviour by using practical reasoning which involves deliberation and means end reasoning. BDI does not include specifications of the agent’s interactions with other agents and also does not specify environmental interface functions of the agent.

BDI was used in implementing the OASIS air traffic management system [78], BDI is the basis of the agent-oriented programming environment JACK AOS [12] and the Prometheus methodology [97].
Level Attributes Analysis: BDI Pattern
The BDI pattern is analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the BDI pattern are Single Agent and Agent Internal Components. These attributes represent two out of the four attributes of the Agent Internal Architecture level of the Two Way classification. The BDI pattern has no other attribute on the level attribute table that matches the attributes of any of the other levels. Therefore, it can be concluded that BDI belongs to the Agent Internal Architecture level of the Two Way classification.

Category Attributes Analysis: BDI Pattern
The BDI pattern is analyzed using the category attribute table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the set of category attributes that are present in the BDI pattern are Knowledge and Strategy components, Reactivity, Deliberation and Knowledge Management.

Two of the BDI pattern’s set of attributes, Knowledge and Strategy components represent two of the five attributes of the Structural category of the Agent Internal
Architecture level. However, since Geometry is not one of the attributes of the BDI pattern, it can be concluded that BDI does not belong to the Structural category.

The other three attributes of the BDI pattern, Reactivity, Deliberation and Knowledge Management, represent three out of the five attributes of the Strategic category of the Agent Internal Architecture level. Therefore it can be concluded that BDI belongs to the Strategic category of the Agent Internal Architecture level.

The BDI pattern is not about the structural arrangement of the internal components of an agent. Rather, BDI is essentially a strategy for reasoning about the goal to commit to and pursue at a particular point in time given a set of possible courses of actions and changing real world situations. The BDI pattern can therefore be applied as the reasoning mechanism in different components of an agent’s internal architecture that require reasoning.

5.4.3.6. Sentinel Pattern

Name: Sentinel

Source: [14]

Problem: An agent has to monitor its environment for a condition and react appropriately at the occurrence of that condition. The information obtained from the environment cannot be evaluated in its original form. If the condition is not satisfied at the first pass at observing the environment, the environment may need to be monitored indefinitely, until the condition is satisfied or until some other constraint is met. How should the agent’s behaviour be designed to carry out the expected behaviour?

Solution: Design the reactive behaviour of the agent using the Sentinel pattern. The Sentinel pattern offers a feedback system with three processes and then a check to determine whether to continue monitoring the condition or to end the
behaviour. The first process is for retrieving information about the state of the environment. The second process is required because the information retrieved from the environment cannot be directly evaluated. Hence, the second process is for processing the information retrieved into a form that the agent can evaluate. The third process evaluates the processed retrieved information and determines if the condition is satisfied, in which case, an appropriate reaction is carried out and the process is ended; if another constraint is met, in which case the process is ended without any reaction carried out; or if the process of monitoring the environment should resume for another pass. See Figure 5.18

![Figure 5.18: The Sentinel Agent Pattern](image)

**Level Attributes Analysis: Sentinel Pattern**

The Sentinel Agent Architecture pattern is analyzed using the level attributes table (table 5.7), to determine the level of abstraction that the pattern belongs to. From this analysis, the set of level attributes that are present in the Sentinel pattern are Single Agent, Agent Internal Components and Constraints, Single Agent. These attributes represent three out four of the attributes of the Agent Internal Architecture level. Therefore, it is concluded that Sentinel agent-oriented pattern belongs to the Agent Internal Architecture level of the Two Way classification scheme.

**Category Attributes Analysis: Sentinel Pattern**

The Sentinel agent-oriented pattern is analyzed using the category attributes table (table 5.10) of the Agent Internal Architecture level to determine the category that the pattern belongs to at this level. From this analysis, the category attributes that
are present in the Sentinel agent-oriented pattern are Reactivity and Knowledge Management. These attributes represent three out of the five attributes of the Strategic category of the Agent Internal Architecture level. The pattern does not have attributes that belong to any other category at this level of abstraction. Therefore, it is concluded that the Sentinel agent-oriented pattern belongs to the Strategic category of the Agent Internal Architecture level.

5.4.4. Multiagent System Realization level

Only one category is defined in this thesis at the MAS Realization level of abstraction. The definition of other categories at this level of abstraction is beyond the scope of this thesis as discussed in section 4.7.2. Therefore, there are no category attribute analyses for the five patterns that have been analyzed to belong to this level.

The five patterns at this level are in two groups. The first group has three patterns by the same authors and the second group has two patterns from the same authors.

5.4.4.1. Synchronizer, Environment Mediated Communication and Behaviour-Based Decision Patterns

These three patterns are written by the same authors and relate to the same kind of problem, hence the result of their analysis and classification are combined after describing them.

**Name:** Synchronizer

**Source:** [117]

**Problem:** Different agents in a multiagent system act simultaneously on a centralized environment. The agents in the multiagent system have fixed positions and carry out certain actions in parallel. For instance, the design of the multiagent system has two agents that are lifting an object together. How should the
multiagent system be implemented to ensure that the actions of the agents are executed simultaneously and not sequentially?

**Solution:** The Synchronizer pattern uses four classes to achieve execution of simultaneous actions. The classes are Agent, Synchronizer, Environment and Influences. The Synchronizer pattern achieves simultaneous execution of agent actions by organizing the activity of the agents in cycles. Each cycle has three phases of perception, action (or influence) and reaction. During the second phase of a cycle, the Agents perform influences on the Environment. The environment then adds these influences to the Influence object. At the beginning of the third phase, the Synchronizer object then triggers the Environment to react to all the influences at once. The Environment then puts all the influences together and decides what their collective outcome should be based on the law of the domain of application. The Environment informs all the agents of the result of their actions or influences and informs the Synchronizer of the end of reactions. Thus, a cycle is ended and another one begins. Figure 5.19 represents the structure of and Figure 5.20 represents the collaborations in the Synchronizer pattern.

![Figure 5.19: Structure of the Synchronizer Pattern](image-url)
Name: Environment Mediated Communication

Source: [117]

Problem: An agent in the multiagent system does not have complete knowledge of the other agents in the multiagent system with which it has to interact. A type of the mediation metaphor for instance, the Broker pattern, the Matchmaker pattern has been used to facilitate the interaction between the agents in designing the multiagent system. How should the Mediator pattern used in the design of the multiagent system be realized in the implementation of the multiagent system?

Solution: The Environment Mediated Communication pattern allows agents to engage in indirect interaction using the environment as the facilitator of the interaction. This pattern uses Environment to represent the Mediator in implementing indirect communication among agents, hence the name Environment Mediated. The Environment among other things is composed of a Communications Broker and a Location Broker. Indirect communication is achieved by an agent sending a message to the Communications Broker. The message sent contains parameters that will guide in determining the intended recipients of the message. The Communications Broker sends a request to the Location Broker to obtain the
targets of the message based on the parameters included in the original message. The Location Broker sends a result of the targets to the Communication Broker who in turn forwards the original message to the target agents. Figures 5.20 and 5.21 are representations of the structure of and the collaborations in the Environment Mediated pattern respectively.

Figure 5.21: Structure of the Environment Mediated Pattern

Figure 5.22: Collaborations in the Environment Mediated Pattern
Problem: An agent operating in an environment has to decide on which action to carry out, that is, the behaviour to exhibit, to appropriately respond to the current state of its environment. The agent has a limited set of independent behaviours. However, this set of behaviours is extensible. How should an agent make decisions on how to respond to the current state of its environment? Or how should an agent select the appropriate action to take from a given set of behaviours in response to the current state of its environment?

Solution: The Behaviour-Based Decision pattern implements an agent’s strategy for selecting the behaviour to exhibit by using a priority like scheme which allows the behaviours themselves to determine which one is to be executed in a particular condition. This is achieved by encapsulating the behaviours in separate Behaviour objects. An aggregate object, BehaviourCollection, then holds all the individual Behaviour objects and is responsible for the final decision on which behaviour is to be chosen. When a decision is to be made, each Behaviour object indicates its ‘willingness’ to be chosen by returning a value, say a priority, to the BehaviourCollection object. The BehaviourCollection object then compares all the priority values returned and uses the result of the comparison to decide on which behaviour(s) will be chosen. Each of the behaviours chosen will then take necessary actions through the Environment interface object. See Figure 5.23 for the structure of the Behaviour-Based pattern.

Level Attributes Analysis:

Synchronizer, Environment Mediated Communication and Behaviour-Based Decision Patterns
The Synchronizer, Environment Mediated Communication and Behaviour-Based Decision patterns were analyzed using the level attribute table (table 5.7), to
determine the level of abstraction that these patterns belong to. From this analysis, the set of level attributes that are present in these patterns are Data Definitions (Classes), Associations and Functions. These attributes represent three out of the five attributes of the MAS Realization level. Therefore, it is concluded that these patterns belong to the MAS Realization level of the Two Way classification scheme.

5.4.4.2. Master-Slave and Meeting Patterns

These two patterns, written by Aridor and Lange [2] are one of the first set of agent-oriented patterns and they relate to implementation of mobility in multiagent systems. The result of their analysis and classification are combined after describing them.

Name: Master-Slave

Source: [2]

Problem: An agent needs to be able to perform two tasks at the same time on different hosts on a network. For instance, an assistant agent that maintains a GUI needs to perform a task at a remote host to achieve some of its goals. For instance, to obtain information after some negotiation that is to be relayed to the user. How should the execution of tasks in parallel by the same agent be implemented?
**Solution:** The Master-Slave pattern uses two classes Master and Slave to achieve the implementation of the same agent running two tasks at the same time. The Master agent carries out the task on the base host, assigns tasks to the Slave agent which then migrates to destination hosts to carry out the assigned task and returns with the result of the task carried out. This pattern can be used to implement the Branching pattern which requires an agent to generate copies of it and send those copies to other hosts to execute tasks. See section 5.5 for a discussion of the importance of this. Also, the Master-Slave pattern can be used to implement the Agent as Delegate pattern where two agents are defined to represent a particular role with one agent having the responsibility of delegating tasks to the other agent. See Figure 5.24 for the structure of the Master-Slave pattern.

![Structure of the Master-Slave Pattern](image)

**Figure 5.24: Structure of the Master-Slave Pattern**

**Name:** Meeting

**Source:** [2]

**Problem:** A collection of mobile agents should carry out a task involving exchange of their resources e.g. mobile commerce agents to buy and sell goods on behalf of their clients. They should be able to continue interacting to carry out their task even when their origins (client’s machines) are disconnected from the network.
or located inside firewalls. The agents need to be able to identify each other when they arrive at the common destination for the transaction.

**Solution:** This pattern uses a `Meeting` class that encapsulates a destination and a unique identifier for the meeting. Participating agents are equipped with a meeting object and will locate a meeting manager object using the unique identifier to register itself. The meeting manager object notifies the registered participants about the newly arrived agent and vice versa. Each agent should unregister itself before leaving the meeting place. See Figure 5.25.

This brief description of the meeting pattern shows that it addresses implementation issues (using the object orientation implementation technology) regarding additional entities that will realize the problem and interactions amongst participating agents/entities. Hence, the horizontal classification is structural, social and mobile while the vertical classification is agent implementation design.

![Figure 5.25: Meeting Pattern](image)

**Level Attributes Analysis: Master-Slave and Meeting Patterns**

The Master-Slave and Meeting patterns are analyzed using the level attribute table (table 5.7), to determine the level of abstraction that the patterns belong to. From this analysis, the set of level attributes that are present in these patterns are Data Definitions (Classes), Associations, Sample Code and Functions. These attributes
represent four out of the five attributes of the MAS Realization level. Therefore, it is concluded that these patterns belong to the MAS Realization level of the Two Way classification scheme.

5.5 Observations from Analysis and Classification of Agent-Oriented Patterns

This section discusses the observations from the analyses of various agent-oriented patterns as documented in the preceding sections of this Chapter. Seven observations from these analyses are discussed in the following subsections.

5.5.1. Concepts versus Experience

There are several agent-oriented patterns written to highlight specific concepts of agent technology. The authors use the structure of software patterns, that is, general solution to a problem in context, to describe the concepts of agent technology. These patterns describe the concepts of agent technology in terms of the kinds of problems they solve, when they are applicable and how such concepts effectively address the problems. This type of patterns does not address specific agent-oriented software engineering challenges from specific projects. Examples of this kind of patterns include the Reactive Agent and Deliberative Agent (section 5.4.3.4), and Agent Society (section 5.4.1.2). Using patterns to describe concepts of agent technology has a positive effect in communicating such concepts to non agent software practitioners.

However, describing certain other concepts as patterns makes it difficult to classify some patterns. For instance, it is difficult to fix Common Vocabulary pattern [127] into a category of the Two Way classification scheme. The attributes of the pattern could not be identified through the attribute tables because of the nature and content of the pattern. Common Vocabulary pattern is a generic description of the concept of ontology based on the recognition of the need for a common representation of the concepts that must be used if two or more agents must interact.
with each other. This does not constitute a pattern in the sense of a general solution to a recurring problem as ontology is a concept that is fundamental to agent technology and will feature in all agent system implementations in one way or the other. Different ways to structure ontology in different application domains more readily constitutes a pattern.

5.5.2. Versatile Patterns

Several agent-oriented patterns can fit into more than one category of the Two Way classification scheme. The information contained in and the structuring of the solutions of these patterns makes them applicable to more than one category of the classification scheme. An example is the InteRRaP pattern that is applicable to two categories Structural and Strategic at the Agent Internal Architecture level (section 5.4.3.2). Some other patterns are applicable to categories across levels. The Agent as Mediator pattern (section 5.4.2.5) is an example of this type of patterns. Agent as Mediator is applicable to the Interactional categories at the Agent-Oriented Analysis and MAS Architecture levels.

5.5.3. Essence of Context

The solution of a pattern resolves the pattern’s context and the resultant system of forces. A problem may require two solutions where there are two contexts and systems of forces. The Embassy, Monitor and Wrapper patterns are three of a set of five coordination patterns by the same authors. The general structure of the solution of these patterns is the same. This general structure has a ‘middle’ agent and sets of agents on either side as shown in Figure 5.12 of section 5.4.2.4 for instance. The differences in these patterns are in the details of their collaborations, their purposes or applicability and their participants. The fact that these five patterns have the same general solution structure emphasizes the importance of the context and the forces of a pattern and the importance of describing the solution in a way to demonstrates how the system of forces is resolved. The choice of one pattern over
another in the same category will be based on the context of the software engineering problem at hand.

The importance of the context and forces of a problem to the solution structure is demonstrated by the analyses of three MAS Architecture – Structural patterns. The patterns are Branching, Star-Shaped Movement and Itinerary (section 5.4.2.2). Even though the solutions of these patterns have the same number of participants, that is the mobile agent and the hosts, the dynamics of their solutions are different based on the context and forces of the system.

5.5.4. Authors’ Classifications versus Attribute-based Analysis

The analyses of some of the patterns resulted in classifying them into levels and categories of the Two Way classification scheme that are different from their authors’ classifications or different from the classifications that their names suggest. The organizational styles of Structure-in-5 and Joint Venture for instance are described as architectural styles by the authors. This suggests that they should belong to the MAS Architecture level of the Two Way classification scheme. However, the results of the attribute-based analyses carried out on them lead to the decision that they belong to the Organizational category of the Agent-Oriented Analysis level (section 5.4.1.1). The kind of attributes in them informed this decision. The Agent Society pattern is another example. The name of this pattern that contains Agent, suggests it should belong to the MAS Architecture level as agent starts to feature at this level in the development process. The analysis of this pattern reveals that it is more applicable to the Agent-Oriented Analysis level (section 5.4.1.2).

5.5.5. Duplicate Patterns

From the analyses carried out, it is observed that some of the patterns are more or less duplications of existing patterns. Some authors present the same problem and solution with the same name but different descriptions. Examples include Protocols
pattern in [61] presents the same problem and solution as the Protocols pattern in [116] though with different descriptions. Also, Facilitator pattern in [61] presents the same problem and solution as the Facilitator pattern in [2] though with different descriptions. In other circumstances, same pattern concept is described by different authors using different pattern names. For instance, the Contract Net Protocol (section 5.4.2.3) and the Call-For-Proposal [25] patterns are basically the same solution concept described using different names. Some other patterns presented by the same authors could be seen as variants of the same solution concept. For example, the Broker and Mediator patterns described in [40] are variations of the same solution concept. Work is required to identify duplicate patterns and note them in building a catalogue of agent-oriented patterns using the classification scheme.

5.5.6. Classification by Critical Attributes

There are instances where patterns have attributes across two categories; however, they are not versatile patterns and therefore cannot be classified into more than one category. The decision on where to classify such patterns is based on the essence of the pattern. That is the main problem that the patterns are written to address. Also considered are the presence or otherwise of other features that are critical to the affected levels or categories. For example, the Star Shaped Movement and Branching patterns have attributes across MAS Architecture and Agent Internal Architecture levels, the decision on where to classify them based on their essence and the critical attributes of the levels is documented in section 5.4.2.2. Another example is the Sense-and-Infer and Ecological Recogniser patterns at the Agent Internal Architecture level with attributes across categories at the Agent Internal Architecture level (section 5.4.3.3).

5.5.7. Vertical Composition of Patterns

The analyses and classification process revealed instances of vertical composition of patterns, that is, connections among patterns by different authors at different
levels of abstractions. For instance, the Agent as Mediator (section 5.4.2.5) and Broker [40] patterns at the MAS Architecture level are connected to the Environment Mediated Communication pattern (section 5.4.4.1) at the MAS Realization level. Also, the Branching pattern (section 5.4.2.2) at the MAS Architecture level is connected to the Master-Slave pattern (section 5.4.4.2) at the MAS Realization level. The Master-Slave pattern represents an implementation of the Branching pattern. Vertical composition is one way by which agent-oriented patterns address of connection between conceptual modeling and implementation [117] in agent-oriented software engineering.

These observations demonstrate the value of the process of analyses of the patterns for classification with regards to the categories of the Two Way classification scheme. Also, the observations indicate the different aspects of agent-oriented patterns that can be improved upon to boost the quality and potential of agent-oriented patterns in enhancing the adoption of agent technology.

5.6 Chapter Summary

The usefulness of the Two Way classification scheme can be demonstrated by classifying agent-oriented patterns from different sources according to the categories of the classification scheme. In order to ensure consistency in the way patterns from different sources are classified according to the categories of the Two Way classification scheme, a repeatable process is required for determining the category of the classification scheme that the patterns belong to.

This Chapter presented a process for analyzing agent-oriented patterns using level and category attributes defined in Chapter Four as the basis of the analysis. This pattern analysis process is referred to as attribute-based analysis. Attribute-based analysis is a repeatable process for analyzing agent-oriented patterns for classification, quality assessment and complexity assessment. The tools of attribute-based analysis, that is level attributes table and category attributes tables were described and used in this Chapter. This Chapter used attribute-based analysis
to analyze and classify the twenty eight existing agent-oriented patterns, out of ninety seven patterns studied, according to the categories of the Two Way classification scheme. The observations which bother on the quality and complexity of the patterns as noted in the process of analyzing these patterns were also documented in this Chapter. Thus, the usefulness of the categories and attributes of the Two Way classification scheme were demonstrated.

In the next Chapter, the categories and attributes of the Two Way classification scheme are used to define eight agent-oriented pattern templates.
Table 5.7: Identification of Level Attributes

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Agent-Oriented Analysis</th>
<th>MAS Architecture</th>
<th>Agent Internal Architecture</th>
<th>MAS Realization</th>
<th>Vertical Classification – Level of Abstraction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Roles/Multiple Roles</td>
<td>Organization of Roles</td>
<td>Roles and Agents</td>
<td>Multiple Agents</td>
<td>Arrangement of Agents</td>
</tr>
<tr>
<td>1. Structure-in-5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Joint Venture</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Pyramid</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4. Agent Society</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>5. Agent as Delegate</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6. Star Shaped</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Branching</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8. Itinerary</td>
<td>Y</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9. Contract Net Protocol</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>10. Embassy</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Monitor</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12. Wrapper</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13. Agent as Mediator</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14. Basic Negotiating Agent Architecture</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15. InteRRaP</td>
<td></td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16. Sense and Infer</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17. Ecological Recognizer</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18. Reactive Agent</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

187
This table is the Level Attributes Table, one of the tools of Attribute-based Analysis (section 5.2.2). It is used for carrying out the levels attributes analysis of agent-oriented patterns. All the twenty eight patterns analyzed in this chapter are passed through this table to identify the vertical classification of each pattern according to the Two Way classification scheme. The vertical classification of each pattern analyzed in this table (shown on the rightmost column) is used to determine which of the tables 5.7, 5.8, or 5.9 that the pattern will be passed through in order to identify the horizontal classification of each of the patterns according to the classification scheme.
This table, the first of the three Category Attributes Tables, is a tool of Attribute-based Analysis (section 5.2.2). It is used for determining the horizontal classification (category attributes analysis) of patterns whose vertical classification is the Agent-Oriented Analysis level of the Two Way classification scheme. Once the vertical classification of a pattern is identified as Agent-Oriented Analysis (result of table 5.7 above), the pattern is passed through this table to determine the horizontal classification of the pattern, that is Organizational or Interactional category of this level.

### Table 5.8: Pattern’s Category Attributes – Agent-Oriented Analysis Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Roles</th>
<th>Coordination/Control</th>
<th>Association/Relationship</th>
<th>Goals</th>
<th>Responsibility</th>
<th>Interaction Objective</th>
<th>Interaction Type</th>
<th>Interaction Roles</th>
<th>Domain Rules</th>
<th>Distribution</th>
<th>Accessibility</th>
<th>Horizontal Classification – Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Structure-in-5</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Organizational</td>
</tr>
<tr>
<td>2. Joint Venture</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Organizational</td>
</tr>
<tr>
<td>3. Pyramid</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Organizational</td>
</tr>
<tr>
<td>4. Agent Society</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Organizational</td>
</tr>
<tr>
<td>5. Agent as Mediator</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interactional</td>
</tr>
</tbody>
</table>
Table 5.9: Patterns' Category Attributes – Multiagent System Architecture Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Combination Rules</th>
<th>Division Rules</th>
<th>Goals</th>
<th>Responsibilities</th>
<th>Other Entities</th>
<th>Knowledge Repositories</th>
<th>Geometry</th>
<th>Bandwidth</th>
<th>Host Stability</th>
<th>Data Volume</th>
<th>Control Flow</th>
<th>Introduced Agents</th>
<th>Message Exchange Rules</th>
<th>Performatives/Message Format</th>
<th>Ontology</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Contract Net Protocol</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interactional</td>
</tr>
<tr>
<td>2. Embassy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interactional</td>
</tr>
<tr>
<td>3. Monitor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interactional</td>
</tr>
<tr>
<td>4. Wrapper</td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interactional</td>
</tr>
<tr>
<td>5. Star-shaped</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Structural</td>
</tr>
<tr>
<td>6. Branching</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Structural</td>
</tr>
<tr>
<td>7. Itinerary</td>
<td></td>
<td></td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Structural</td>
</tr>
<tr>
<td>8. Agent as Delegate</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Definitional</td>
</tr>
<tr>
<td>9. Agent as Mediator*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interactional</td>
</tr>
</tbody>
</table>

This table, the second of the three Category Attributes Tables, is a tool of Attribute-based Analysis (section 5.2.2). It is used for determining the horizontal classification (category attributes analysis) of patterns whose vertical classification is the MAS Architecture level of the Two Way classification scheme. Once the vertical classification of a pattern is identified as MAS Architecture (result of table 5.7 above), the pattern is passed through this table to determine its horizontal classification, that is Definitional, Structural or Interactional category of this level.
Table 5.10: Pattern’s Category Attributes – Agent Internal Architecture Level

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Structural</th>
<th>Interactional</th>
<th>Strategic</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge Component</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Structural/Strategic</td>
</tr>
<tr>
<td>Strategy Component</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Strategic</td>
</tr>
<tr>
<td>Interaction Component</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interaction</td>
</tr>
<tr>
<td>Environmental Interface</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interaction</td>
</tr>
<tr>
<td>Perception Handling</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interaction</td>
</tr>
<tr>
<td>Action Execution</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interaction</td>
</tr>
<tr>
<td>Environment</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Interaction</td>
</tr>
<tr>
<td>Autonomy</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Strategic</td>
</tr>
<tr>
<td>Reactivity</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Strategic</td>
</tr>
<tr>
<td>Deliberation</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Strategic</td>
</tr>
<tr>
<td>Knowledge Management</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Strategic</td>
</tr>
<tr>
<td>Adaptability</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Structural</td>
</tr>
</tbody>
</table>

This table, the third of the three Category Attributes Tables, is a tool of Attribute-based Analysis (section 5.2.2). It is used for determining the horizontal classification (category attributes analysis) of patterns whose vertical classification is the Agent Internal Architecture level of the Two Way classification scheme. Once the vertical classification of a pattern is identified as Agent Internal Architecture (result of table 5.7 above), the pattern is passed through this table to determine its horizontal classification, that is Structural, Interactional or Strategic category of this level.
Chapter 6

Agent-Oriented Pattern Templates

6.1 Introduction

Chapter Two discussed the potential of agent-oriented patterns for enhancing the adoption of agent technology. The issues that impact on the potential of agent-oriented patterns are also presented in this Chapter. Chapter Four addressed the issue of agent-oriented pattern classification by presenting a Two Way classification scheme and identifying attributes for the levels and categories of the Two Way classification scheme. Chapter Five analyzed and classified twenty eight (28) existing agent-oriented patterns according to the categories of the Two Way classification scheme.

This Chapter presents a template structure for describing agent-oriented patterns. Sane [112] states that “different forms may be appropriate for different uses, although needless originality should be avoided. The pattern form should be standard.” Though this statement was made about ten years ago, it is of utmost relevance to the description of agent-oriented patterns. The diversity of concepts of agent technology necessitates the use of different templates for describing different categories of agent-oriented patterns [41, 77, 83]. Avoiding needless originality suggests the need to create some boundaries to constrain the diversity of agent-oriented pattern templates.

The template structure presented in this Chapter is adaptable but consistent. It is adaptable as it can be used to derive customized description templates for the
different categories of agent-oriented patterns. However, the template structure is consistent across the different description templates derived from it. This template structure is designed to feature the concepts of agent technology in a description format that is familiar to a cross section of software industry practitioners thereby enhancing the communication and comprehension of agent technology as a software development paradigm. This Chapter shows how to customize the designed template structure for each of the categories at the Agent-Oriented Analysis, MAS Architecture, and Agent Internal Architecture levels of the Two Way classification scheme.

Section 6.2 discusses the desired properties of agent-oriented pattern templates. Section 6.3 presents the design of the template structure and describes how this template structure can be used to derive agent-oriented pattern templates for the different categories of the Two Way classification scheme. Section 6.4 derives the agent-oriented pattern templates from the template structure for the eight categories of the Two Way classification scheme. In section 6.5, eight out of the twenty-eight patterns analyzed and classified according to the categories of the Two Way classification scheme in Chapter Five, are described using the pattern templates derived in section 6.4. Section 6.6 presents the Chapter summary.

6.2 Desired Properties of Agent-Oriented Pattern Templates

Three desired properties of description templates for agent-oriented patterns are described in this section. These desired properties are identified based on the analyses of existing agent-oriented pattern templates as discussed in section 2.3.6 of Chapter Two and the statement of Sane in [112]. The three properties are described as follows.

1. Completeness: Agent-oriented pattern templates should reflect the concepts of agent technology in the description of agent-oriented patterns. The elements of agent-oriented pattern templates should be carefully defined to ensure that necessary concepts of agent technology are captured in the description of the
pattern [77, 41]. This is one instance of different forms being appropriate for different uses [112]. Pattern templates required for describing agent-oriented patterns are different from templates required for describing patterns in other software engineering paradigms such as Object Oriented patterns.

2. **Elimination of Redundancy:** Agent-oriented pattern templates should be appropriate for and relevant to the different categories of agent-oriented patterns. A single template is not adequate for all types of agent patterns because the concepts of agent technology are different at the different levels of abstractions in agent-oriented software engineering. Using a single template for all categories of agent-oriented patterns results in template elements that are redundant for some categories of patterns while applicable to others. This is another instance of different forms being appropriate for different uses [112]. Different pattern templates should be defined for describing the different categories of agent-oriented patterns due to the differences in the concepts of agent technology featured by these different categories [77, 83].

3. **Elimination of Ambiguity:** Agent-oriented pattern templates should be universally comprehensible. Even though agent-oriented patterns require different description templates, template elements for describing agent-oriented pattern should be understandable by a broad spectrum of software engineering practitioners. This is an instance of avoiding needless originality [112]. That is, there is no need for defining completely unique pattern templates for describing agent-oriented patterns.

The goal is to generate pattern templates for the effective and efficient communication of the different types of agent-oriented patterns that exist. To achieve this goal, the pattern template structure for generating pattern templates need to feature the desired properties of agent-oriented pattern templates. Hence, these three properties form the basis of the design of agent-oriented pattern template structure presented in the following section.
6.3 Agent-Oriented Pattern Template Structure

The previous section presented the desired properties of agent-oriented pattern templates. This section designs a pattern template structure according to these desired properties. A pattern template structure is the organization of the information that is contained in a pattern. The pattern template structure determines how the elements of a pattern template are arranged for communicating the essence of the pattern. As such two separate pattern templates can have the same pattern template structure. The template structure designed here is used as the basis for deriving agent-oriented pattern templates for the eight categories of the Two Way classification scheme in section 6.4. The activities involved in the design of the template structure are presented in this section.

The design of an agent-oriented pattern template structure involves the following activities:

- identifying the components of the template structure and designing the arrangement of these components;

- identifying the fixed segment elements of the template structure;

- determining the fixed segment elements that should have variable elements; and

- identifying the variable elements of agent-oriented pattern templates.

Each of these activities is addressed in turn in the following subsection.

6.3.1. Template Structure Components and their Arrangement

The identification of the components and design of the components’ arrangement in the template structure builds on two existing agent-oriented pattern template structures. The templates presented by [77] and [83] are analyzed and adapted for
the construction of the template structure. These template structures are chosen because they recognize the need for a more elaborate pattern template structure for describing agent-oriented patterns. Reviews of these templates have been presented in section 2.3.6 of Chapter Two.

Lind [77] presents a two part pattern description template, with a generic part and view-specific part. There are twelve elements of the generic part of the template which are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Aliases</th>
<th>Problem</th>
<th>Forces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entities</td>
<td>Dynamics</td>
<td>Dependencies</td>
<td>Example</td>
</tr>
<tr>
<td>Implementation</td>
<td>Known Uses</td>
<td>Consequences</td>
<td>See Also</td>
</tr>
</tbody>
</table>

There are eight elements of the view-specific part for the architecture view which include:

<table>
<thead>
<tr>
<th>Resource Limitations</th>
<th>Control flow</th>
<th>Knowledge handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasoning capabilities</td>
<td>Autonomy</td>
<td>User interaction</td>
</tr>
<tr>
<td>Temporal context</td>
<td>Decision making</td>
<td></td>
</tr>
</tbody>
</table>

Malyankar [83] proposes a pattern template for agent-oriented patterns by adding elements and sub categories to the thirteen element pattern template of the GoF [34]. The GoF pattern template elements are:

<table>
<thead>
<tr>
<th>Pattern Name and Classification</th>
<th>Intent</th>
<th>Also Known As</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motivation</td>
<td>Applicability</td>
<td>Structure</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Consequences</td>
<td>Implementation</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Related Patterns</td>
<td>Sample Code</td>
</tr>
</tbody>
</table>
The elements added to the GoF template by Malyankar include Dependencies and State Storage. The Dependencies element is subdivided into Dependencies managed and Dependencies affected. The Consequence element of the GoF template is subdivided into Scaling behaviour, Emergent effects and Pathology.

The analyses of these two templates in Chapter Two show that having two or more parts to the structure of a pattern template adds flexibility to the template structure. Flexibility is achieved by combining a stable part with a changeable part in the structure of the pattern template. The flexibility and extensibility of the template structure is subject to the strategy of combination of the stable and changeable parts of the template structure.

Based on these analyses, the pattern template structure presented here is designed to have two segments. The two segments are a fixed segment and a variable segment. The fixed segment of the template structure comprises of template elements considered to be fundamental to the description of software patterns and therefore critical to the universal comprehension of agent-oriented patterns. The identification of the elements of the fixed segment, that is fixed elements, is described in the next subsection. The variable segment of the template structure comprises of template elements that are essential to the communication of the concepts of agent technology. The variable segment adds flexibility to the template structure and facilitates the generation of different pattern templates from the same template structure.

In the design of the template structure, the elements of the variable segment, that is the variable elements, constitute sub elements to specific elements of the template structure’s fixed segment (table 6.1). Two pattern templates derived from this template structure differ only in their variable elements, their fixed elements are the same.

Table 6.1 shows the layout of the template structure and the relationship between the fixed segment and the variable segment of the template structure. This is the
layout of all pattern templates derived from this template structure. The layout in table 6.1 depicts that the variable elements are sub elements to the same set of fixed elements in all cases of pattern templates derived from this template structure. That is only fixed elements B, D and F can have variable elements as sub elements in all instances of pattern templates derived from this template structure. Variable elements are not assigned to the fixed elements A, C, E and N in any instance of pattern templates derived from this template structure.

Table 6.1: Pattern Template Structure showing Fixed and Variable Segments

<table>
<thead>
<tr>
<th>Fixed Segment</th>
<th>Variable Segment (sub elements to fixed elements B, D and F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed element A</td>
<td></td>
</tr>
<tr>
<td>Fixed element B</td>
<td>Variable element B1, Variable element B2, Variable element B3</td>
</tr>
<tr>
<td>Fixed element C</td>
<td></td>
</tr>
<tr>
<td>Fixed element D</td>
<td>Variable element D1, Variable element D2</td>
</tr>
<tr>
<td>Fixed element E</td>
<td></td>
</tr>
<tr>
<td>Fixed element F</td>
<td>Variable element F1, Variable element F2, Variable element F3, Variable element F1</td>
</tr>
<tr>
<td>Fixed element N</td>
<td></td>
</tr>
</tbody>
</table>

6.3.2. Fixed Elements of the Template Structure

The next activity in the design of the template structure is the identification of the fixed elements of the template structure. In this activity, the fundamental principles of a software pattern are reviewed to determine the fixed elements of the template structure.

According to the analyses of the various existing definitions of patterns in Chapter Two, the three key concepts of software patterns include the problem, the general solution and universal communication. These concepts are essential to any software pattern. Based on this analysis, the essential components of software pattern were identified as problem, context, forces, solution and resulting context (section 2.3.1
These five components therefore represent the fixed elements of the template structure. In addition to these five elements representing the fixed elements of the template structure, *name, classification, known uses, and related patterns* are identified as fixed elements of the template structure to improve its expressiveness. Table 6.2 shows the fixed elements of the template structure in the order that facilitates the communication of a pattern.

**Table 6.2: Fixed Elements of the Template Structure**

<table>
<thead>
<tr>
<th>Fixed Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Classification</td>
</tr>
<tr>
<td>Problem</td>
</tr>
<tr>
<td>Context</td>
</tr>
<tr>
<td>Forces</td>
</tr>
<tr>
<td>Solution</td>
</tr>
<tr>
<td>Resulting Context</td>
</tr>
<tr>
<td>Known Uses</td>
</tr>
<tr>
<td>Related Patterns</td>
</tr>
</tbody>
</table>

These fixed elements of the template structure satisfy the avoidance of needless originality [112] and therefore achieve the Elimination of Ambiguity property described in section 6.2. These elements are universally recognized and are not subject to individual interpretations.

The following are the descriptions of the fixed elements of the template structure.

**Name:** Describes a name for referring to the pattern and other names by which the pattern is described, if any. Pattern names usually indicate what the pattern is about, however, observations from the analyses of patterns in section 5.5 of Chapter 5 show instances of pattern names that do not reflect what the patterns are about. A pattern’s name facilitates discussions about the pattern among software
practitioners. The choice of a pattern’s name is important due to the number of agent-oriented patterns that exist.

**Classification:** Describes the position of the pattern in the Two Way classification scheme. Classification is included in the fixed elements of the template structure based on the work done in Chapters four and five. Classification is essential as it gives an indication of the applicability of an agent-oriented pattern in terms of level of abstraction and software development task.

**Problem:** Defines the recurring problem that the general solution is defined to solve. Problem is a fundamental component of a pattern as it is the reason for the pattern. The problem that a pattern addresses should therefore be stated as early as possible in the description of the pattern.

**Context:** Describes a particular setting for the problem, that is, a particular scenario in which the problem occurs. Context provides the basis for defining the factors constraining the solution to the problem at hand. Context is a fundamental component of a pattern. It provides an indication of the applicability of a pattern.

**Forces:** Describes the constraints that are relevant to a particular problem based on the context of the problem. Forces is an important fundamental component of a pattern as the system of forces is what determines why a solution is applicable to the problem in context and another equally valid solution is not applicable to the same problem. Forces determine the applicability of a pattern to other projects.

**Solution:** Describes the general solution that best resolves the forces identified. Solution is a fundamental component of a pattern. A pattern’s solution should be expressive enough to show how the solution resolves the system of forces. Solution should describe the core of a pattern’s solution, that is, the general principles applied to resolve the system of forces. Describing the core of a solution facilitates the adaptability of the pattern to other projects. The solution of a pattern captures the participating entities in the solutions (depending on level of abstraction and software development task), the collaborations among the participants, structure of the participants and the dynamics of the behaviour of the participants.

**Resulting context:** Describes the effects of applying the general solution to the problem in its initial context/forces. Resulting context is a fundamental component of a pattern which describes the unresolved system of forces when a particular
solution is applied to the problem. The resulting context also describes the advantages of the solution.

**Known Uses:** Describes existing applications of the pattern. Known uses describe examples that place patterns in the perspective of real software development projects. It helps potential pattern users gain a better understanding of how the pattern may be adapted and applied in real projects.

**Related Patterns:** Describes other agent-oriented patterns that are connected to the pattern being described in one way or another. Connected patterns include other patterns that lead to or those that follow from the pattern being described or patterns that are variants of or alternates to the pattern being described. For example, a pattern at the MAS Architecture level is related to another pattern that implements it at the MAS Realization level (see the observations on Branching and Master-Slave patterns in section 5.5. of Chapter 5).

### 6.3.3. Fixed Segment Elements with Variable Elements

In this activity, the fundamental components of a pattern are analyzed to determine the fixed elements of the template structure that variable elements should be attached to.

As noted in [95] a pattern describes a specific recurring problem in a context that defines a set of forces which are resolved by a general solution to create a resulting context.

A problem can recur in different scenarios such that a particular solution may be relevant in one scenario and irrelevant in others. Also, the scenarios determine the trade-offs that result by applying a solution. The particular scenario in which the solution is applicable to solving the recurring problem at hand is represented by the context and forces aspect of the pattern.

Where the applicability of a pattern to a designer’s specific problem is determined, by the context and forces, the pattern solution needs to be adapted since the
solution of a pattern is usually a general solution. A pattern has to be adapted to the specifics of the project to which the pattern is applied. Adapting a pattern's solution is necessary for integrating the pattern into other aspects of the design or combining it with other patterns to generate composite patterns. The description of the solution and resulting context of the pattern determines how adaptable the pattern solution is and how well it can be integrated into other aspects of design or combined with other patterns.

Hence, a software designer determines the applicability of a pattern to a specific design problem by the context and the forces of the pattern. The software designer determines the adaptability of a pattern to other aspects of the design by the solution structure and resulting context of the pattern. That is the applicability of a pattern is expressed by the context and forces components of the pattern and the adaptability is expressed by the solution and resulting context of the pattern.

It can be argued that if a software practitioner is able to determine the applicability of an agent-oriented pattern to a problem at hand, it shows that the software practitioner understands the concepts of agent technology as contained in the pattern. Also, it can be argued that if a software practitioner is able to adapt an agent-oriented pattern and integrate it into an existing design, it shows that the software practitioner has appreciable understanding of the concepts of agent technology.

The concepts encapsulated by the pattern are better understood if the applicability and adaptability of the pattern are adequately expressed. Therefore, the communication and comprehension of agent technology are most effective when the concepts of agent technology are applied towards describing the applicability and adaptability of an agent-oriented pattern.

In order to make the pattern template more applicable to agent-oriented patterns, variable elements are assigned to the Context and Forces fixed elements to make the applicability of agent-oriented patterns more expressive and to reflect the

203
notions of agent technology. Also, variable elements are assigned to the Solution and Resulting Context fixed elements to make the adaptability of agent-oriented patterns more expressive and to reflect the concepts of agent technology. Table 6.3 shows the fixed elements identified for the template structure as well as the fixed elements to which variable elements are assigned to.

Table 6.1 (section 6.3.1) is a generic layout of the template structure. Table 6.3 shows the nine fixed elements of the template structure and the four (context, forces, solution and resulting context) of the nine fixed elements to which variable elements are assigned to.

Table 6.3: Pattern Template Structure

<table>
<thead>
<tr>
<th>Fixed Segment</th>
<th>Variable Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Fixed elements)</td>
<td>(Sub elements to fixed elements - context, forces, solution, resulting context)</td>
</tr>
<tr>
<td>Name</td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
</tr>
<tr>
<td>Problem</td>
<td></td>
</tr>
</tbody>
</table>
| Context | Variable element context1  
Variable element context2  
... |
| Forces | Variable element forces1  
Variable element forces2  
... |
| Solution | Variable element solution1  
Variable element solution2  
... |
| Resulting Context | Variable element resulting context1  
Variable element resulting context2  
... |
| Known Uses | |
| Related Patterns | |

The variable elements that are assigned to these four fixed elements are determined by the particular pattern template that is derived from the template structure. The next activity describes the basis for the identification of the variable elements that are relevant to different agent-oriented pattern templates.
6.3.4. Variable Elements of the Template Structure

The activity in the previous subsection achieves the elimination of ambiguity property of agent-oriented pattern templates. The next activity achieves the completeness and elimination of redundancy properties by defining the concepts of agent technology as variable elements and determining the number and types of agent-oriented pattern templates to be derived from the designed template structure.

The Two Way classification scheme presented in Chapter Four defines eight categories of agent-oriented patterns each having distinct characteristics. The differences among these categories of patterns are described in Chapter Four. These categories represent the level of abstraction in agent-oriented software engineering and the software development tasks at each of these levels. Also described in Chapter Four are the level and category attributes of these categories. Level attributes are the distinguishing properties of the patterns that belong to the levels in the Two Way classification scheme. Category attributes are the distinguishing properties of the patterns that belong to the categories in the Two Way classification scheme. The category attributes define the characteristics of the categories in the Two Way classification scheme.

The number of agent-oriented pattern templates to be derived from the template structure is based on the categories of the Two Way classification scheme. Therefore, eight agent-oriented pattern templates are derived from the template structure, one for each of the categories of the Two Way classification scheme. The definition of eight agent-oriented pattern templates for the eight categories of the Two Way classification scheme eliminates redundancy and facilitates expressiveness of the agent-oriented patterns belonging to these categories.

The variable elements to be identified for each of the eight pattern templates are based on the level and category attributes of the Two Way classification scheme categories. The attributes define the distinct characteristics of the categories therefore they represent the essential information about the patterns in the category.
Defining the variable elements of a category’s pattern template on the basis of the level and category attributes ensures that the essential information that is relevant to the patterns in the category is captured in the pattern descriptions.

The next challenge is to determine which of the level and category attributes of a particular category represent the context, forces, solution or resulting context of the patterns in that category. The in depth analyses and classification of twenty eight existing agent-oriented patterns in Chapter Five was used to determine which of the attributes of a category inherently describe the forces and context as well as the attributes that inherently describe the solution and resulting context of a pattern that belongs to that category.

The attributes that represent the variable elements to be assigned to the four fixed elements context, forces, solution and resulting context are identified in section 6.4. The derived pattern templates for the eight categories of the Two Way classification scheme based on these attributes are presented in section 6.4.

### 6.4 Two Way Classification Scheme Pattern Templates

This section presents eight agent-oriented pattern templates for describing agent-oriented patterns in the eight categories of the Two Way classification scheme. The pattern templates are derived from the template structure designed in the preceding section.

Two templates are defined for the two categories of the Agent-Oriented Analysis level, three templates are defined for the three categories of the MAS Architecture level, and three templates are defined for the three categories of the Agent Internal Architecture level. No templates are defined for the MAS Realization level because no categories are defined for this level in this work as discussed in section 4.5.6.
The agent-oriented pattern templates presented in this section are arranged in the following subsections according to the levels of the Two Way classification scheme.

6.4.1. Template for Agent-Oriented Analysis Level Patterns

There are two agent-oriented pattern templates at this level of the Two Way classification scheme. They are agent-oriented pattern template for Organizational patterns and agent-oriented pattern template for Interactional patterns. Table 6.4 shows the elements of the fixed segment, which is common to the two templates, and the elements of the variable segments of each of the two templates that are assigned to the elements of the fixed segment.

<table>
<thead>
<tr>
<th>Fixed Elements</th>
<th>Variable Elements for Organizational Category</th>
<th>Variable Elements for Interactional Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td>Domain of application</td>
<td>Domain of application</td>
</tr>
<tr>
<td>Forces</td>
<td>Goals</td>
<td>Interaction objective</td>
</tr>
<tr>
<td></td>
<td>Coordination/Control</td>
<td>Domain Rules</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accessibility</td>
</tr>
<tr>
<td>Solution</td>
<td>Roles</td>
<td>Interaction type</td>
</tr>
<tr>
<td></td>
<td>Association/Relationships</td>
<td>Interaction roles</td>
</tr>
<tr>
<td></td>
<td>Coordination/Control</td>
<td>Distribution</td>
</tr>
<tr>
<td>Known Uses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resulting Context</td>
<td>Adaptation/Integration</td>
<td>Adaptation/Integration</td>
</tr>
<tr>
<td>Related Patterns</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The two templates at this level do not feature concepts such as agents and ontology in their variable elements due to the fact that these concepts are not relevant at this level. The absence of these non relevant concepts at this level illustrates the elimination of redundancy property of agent-oriented pattern templates (section 6.2).
The descriptions of the elements of each of the templates are presented in the following subsections. The fixed and variable segments of each pattern template are combined to give a complete description of each pattern template. In each template description, the variable elements are described as sub elements under the fixed elements that they are assigned to.

6.4.1.1. Agent-Oriented Analysis-Organizational Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the Two Way classification scheme.

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces. The following agent concept is introduced as a sub-element to the Context element.

*Domain of Application:* defines the nature of the domain of the system being modeled. This specifies the overall system structure, for instance if it is a system that functions over the internet or only a department of an organization.

**Forces:** describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub elements to the Forces element.

*Goals:* defines the goals, the sub goals and soft goals of the system being modeled. The goals and sub goals constrain the structure and relationships of the roles or agents in the system.
**Coordination/Control:** defines the constraints to how the groups or entities (roles or agents) in the organization can relate. It specifies constraints resulting from hierarchical structure, supervisory controls/limits and access to resources.

**Solution:** describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

**Roles:** describes the roles that are going to feature when carrying out the role relationship analysis of the system. It specifies both software and human roles that are required for the analysis of the system.

**Association/Relationship:** defines the relationship between the entities of the real life organization. There are two types of associations at the analysis level. One is the role-role relationship which is the definition of the associations among the roles in the organization. Role-role relationships describe the structure and arrangement of the roles within the system. Two is role-goal relationship which is the definition of the associations among the roles and the roles within the organization. The roles that are connected to a particular goal are the roles that have the responsibility to achieve the goal.

**Coordination/Control:** describes the coordination of resources in the role organizations described above. Control flow specifies the strategy for sharing limited resources, maximizing potentials and carrying out group tasks. Control flow also specifies whatever limits or boundaries to what individual entity in the system can do. This is different from the Coordination/Control variable element under the Forces section of the pattern. While Coordination/Control under Forces describes the constraints relating to coordination of the members of the organization to be modeled, Coordination/Control under Solution describes how the specific roles in the pattern are coordinated.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.
Adaptation/Integration: presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

Related Patterns: specifies patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

6.4.1.2. Agent-Oriented Analysis-Interactional Patterns

Name: describes a name for referring to the pattern and other names by which the pattern is described, if any.

Classification: describes the position of the pattern in the Two Way classification.

Problem: defines the recurring problem that the general solution is defined to solve.

Context: describes a particular setting for the problem which provides the basis for defining the relevant forces. The following agent concept is introduced as a sub-element to the Context element.

\textit{Domain of Application}: defines the nature of the domain in terms of the interaction needs of the system. The domain of application impacts on how interactions will play out in the system. This specifies the basis for the interaction constraints in the system.

Forces: describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub-elements to the Forces element.

\textit{Interaction objectives}: specifies the purpose of the interaction in question. The purpose of the interaction constrains the participants and the dynamics as well as expected outcome of the interaction. The purpose of the interaction will also introduce issues like safety or security requirements.
**Rules:** defines issues such as the conditions required for the interaction to commence, to continue and to conclude. It also specifies conditions under which agents are to participate in the interaction.

**Accessibility:** specifies whether each agent is aware of the other agents within the system and how to contact them. Agents in a closed system have greater accessibility to one another when compared with agents in an open system.

**Solution:** describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

**Interaction type:** describes the type of interaction that will most effectively resolve the system of forces identified for the interaction problem. Interaction types include negotiation, competition, or collaboration. Specifying the interaction type will address the forces of interaction objectives and rules. Interaction type will also determine certain design options further on in the development process.

**Interaction roles:** defines the participants in the interaction. It also specifies who initiates the interaction, who can conclude it as well as who can cancel the interaction.

**Distribution:** specifies the form of connection amongst the various roles in the interaction. That is, this aspect of the solution defines the One-Many, Many-Many relationships in the interaction. It addresses the accessibility constraints of the interaction.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.

**Adaptation/Integration:** presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.
Related Patterns: specifies patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

6.4.2. Template for MAS Architecture Level Patterns

There are three agent-oriented pattern templates at this level of the Two Way classification scheme. They are agent-oriented pattern template for Definitional patterns, agent-oriented pattern template for Structural patterns and agent-oriented pattern template for Interactional patterns. Table 6.5 shows the elements of the fixed segment, which is common to the three templates, and the elements of the variable segments of each of the three templates that are assigned to the elements of the fixed segment.

Table 6.5: Pattern Templates – Multiagent System Architecture level

<table>
<thead>
<tr>
<th>Fixed Elements</th>
<th>Variable Elements for Definitional Category</th>
<th>Variable Elements for Structural Category</th>
<th>Variable Elements for Interactional Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td></td>
<td>• Organization of Roles</td>
<td></td>
</tr>
<tr>
<td>Forces</td>
<td>• Goals</td>
<td>• Quality goals</td>
<td>• Interaction Objective</td>
</tr>
<tr>
<td></td>
<td>• Responsibilities</td>
<td>• Knowledge volume</td>
<td>• Adaptive Behaviour</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Knowledge source</td>
<td>• Security</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Infrastructure</td>
<td>• Language Constraint</td>
</tr>
<tr>
<td>Solution</td>
<td>• Agents</td>
<td>• Agents</td>
<td>• Interaction Strategy</td>
</tr>
<tr>
<td></td>
<td>• Role-Agents</td>
<td>• Other entities</td>
<td>• Introduced Entities</td>
</tr>
<tr>
<td></td>
<td>• Rationale</td>
<td>• Geometry</td>
<td>• Ontology</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Interaction mechanism</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Communication/Connectivity</td>
</tr>
<tr>
<td>Known Uses</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resulting context</td>
<td>Adaptation/Integration</td>
<td>Adaptation/Integration</td>
<td>Adaptation/Integration</td>
</tr>
<tr>
<td>Related Patterns</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other Comments</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

At this level, the variable elements of the templates feature concepts such as agents, agent types geometry (arrangement of agents), and concepts relating to agent interaction such as ontology, adaptive behaviour and interaction mechanism. These
concepts are consistent with the agent-oriented software engineering tasks at this level. The Agent-Oriented Analysis level concepts of organization of roles and interaction objective which feature in the templates at this level as context and forces depict how the solution of one level of abstraction can define the context of the problem at the next lower level of abstraction.

The descriptions of the elements of each of the templates are presented in the following subsections. The fixed and variable segments of each pattern template are combined to give a complete description of each pattern template. In each template description, the variable elements are described as sub elements under the fixed elements that they are assigned to.

### 6.4.2.1. Multiagent System Architecture-Definitional Pattern

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the Two Way classification.

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

**Forces:** describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub elements to the Forces element.

**Goals:** defines the goals of the roles that are to be translated to an agent or agents. It defines the degree of commitment to and ownership (shared or solely owned) of the goals.
**Responsibilities:** defines the responsibilities of the role or roles to be translated to agent or agents in order to ensure that all the responsibilities of the role or roles are captured by the corresponding agent.

**Solution:** describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

**Agents:** describes the agents that the roles are translated into. Specifies the types of agents that the roles are translated into based on the domain of application and the responsibilities of the roles. Types of agents include information agents, negotiating agents, assistant agents, and mobile agents.

**Role-Agents:** describes the mapping of the roles to the agents they are translated into. It introduces traceability to the pattern and therefore to the analysis that the pattern is applied to.

**Rationale:** describes the basis for combining or dividing roles into agent or agents by considering the goals and responsibilities of the roles. Rationale is an important aspect of the solution to this type of patterns as it is the basis for the adaptation of the principles used in the pattern for the translation of the roles to agents.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.

**Adaptation/Integration:** presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

**Related Patterns:** describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.
6.4.2.2. Multiagent System Architecture-Structural Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the Two Way classification.

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

- **Organization of Roles:** defines the model of the real life system for which the multiagent system is to be designed as organization of roles. Organizations or roles are defined at the Agent-Oriented Analysis level of the Two Way classification scheme and they serve as considerations in the design of the structural arrangement of the agents in the multiagent system.

**Forces:** Describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as subelements to the Forces element.

- **Quality goals:** describes the performance requirements of the agent based system that will implement the virtual organization being modeled. Quality goals represent the constraints on the design of the agent based system as a result of quality of operation required of the system.

- **Knowledge Volume:** describes the volume of knowledge that the agents in the multiagent system being designed will use. The volume of knowledge to be used by the individual agents will determine if the knowledge will be built into individual agents or some knowledge base will be required to implement the system.

- **Knowledge Source:** describes the existing sources of the knowledge that will be used by the agents in the system based on the domain of application. The
existing knowledge sources could be specific databases, the Internet, or interaction with some other multiagent system.

*Infrastructure:* describes the constraints on the hardware platform for implementing the multiagent system. Infrastructure describes constraints relating to network bandwidth, number of nodes on the network, as well as capacity of the nodes on the network.

**Solution:** describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

*Agents:* defines the agents that will feature in the multiagent system or the aspect of the multiagent system that the solution is addressing. Agents will specify the type of agents and the role they will be playing in the multiagent system.

*Other entities:* describes all the non agent entities that will feature in the design of the multiagent system. These are entities that need to be introduced into the multiagent system architecture to address soft goals and knowledge requirements of the system.

*Geometry:* describes the structural arrangement of the entities in the multiagent system. Geometry defines the arrangement of the entities in the multiagent system into units or groups as well as the relationships among these groups. The geometry is designed to address the forces posed by the context of the problem at hand and the intended goals of the system.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.

*Adaptation/Integration:* presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.
**Related Patterns:** describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

### 6.4.2.3. Multiagent System Architecture-Interactional Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the Two Way classification.

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

**Forces:** Describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub elements to the Forces element.

- **Interaction objective:** describes the purpose of the interaction in question. The purpose of the interaction constrains the participants and the dynamics as well as expected outcome of the interaction. The purpose of the interaction will also introduce issues like safety or security requirements.

- **Adaptive behaviour:** describes the challenges to the interaction as a result of adaptive behaviour requirements of the system. An example of a constraint posed on the interaction where adaptability is required is the interaction cannot be rigid or fixed over the life of the multiagent system. Interaction in an adaptive environment will need to evolve.

- **Security:** describes the sensitivity of information to be exchanged in the course of the interaction. Security specifies the need to protect the information to be exchanged or otherwise.
Language constraints: describes the uniformity or otherwise of concept representation by the different parties involved in the interaction. This is most applicable where an agent needs to interact with heterogeneous agents.

Agent type: describes the nature of the agents that will be involved in the interaction. The agent type for this purpose could be heavy weight agents with a lot of deliberation, largely reactive agents or a combination of both. The type of agents will affect the kind of interaction protocol that will be suitable for the interaction to be achieved.

Error sensitivity: defines what level of error is tolerable in the outcome of the interaction in question. This will raise issues such as message integrity.

Time: defines how time critical the interaction is. Real time systems for instance will require time sensitive interactions in order to achieve the interaction’s objectives.

Solution: describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

Interaction Strategy: describes the type of interaction. It could be a simple type such as negotiation, collaboration or a composite type such as competitive-negotiation. It also specifies the policies guiding the conduct of the interaction. For instance, inclusiveness considerations can be used to handle the issue of malicious agents.

Introduced entities: defines the new agents or other entities introduced to the system for the sole purpose of facilitating the interaction.

Ontology: specifies strategy for representation of the real world for the purpose of the interaction where the participants have different representations of the real world. E.g. use of a neutral representation of the real world that the participating agents must interpret

Interaction Mechanism: describes the rules of high level message exchanges amongst interacting agents. For example, English Auction rules, the format of the messages to be exchanged as well as the performatives, and messaging mode such as synchronous versus asynchronous, bilateral versus multilateral, and multicast, broadcast.
Communication/Connectivity: describes the communication policies and services that enable flow of data across physical media. This element describes low level data transmission solutions, and networking solutions.

Known uses: describes existing applications of the pattern.

Resulting context: describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.

Adaptation/Integration: presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

Related Patterns: describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

6.4.3. Template for Agent Internal Architecture Level Patterns

There are three agent-oriented pattern templates at this level of the Two Way classification scheme. They are agent-oriented pattern template for Structural patterns, agent-oriented pattern template for Interactional patterns and agent-oriented pattern template for Strategic patterns. Table 6.6 shows the elements of the fixed segment, which is common to the three templates, and the elements of the variable segments of each of the three templates that are assigned to the elements of the fixed segment.

The concepts of agent technology that are reflected in the variable elements of the templates at this level include autonomy, reactivity, deliberation, interaction processing, and intelligence. These are concepts that relate to the internal architecture of an agent and are therefore consistent with the agent-oriented software engineering tasks at this level. The featuring of these concepts at this level achieves the completeness property of agent-oriented pattern templates (section
The fact that these concepts do not appear in any of the two templates at the higher levels of abstractions demonstrates the elimination of redundancy in these pattern templates.

It is worthy of note that concepts such as goal, interaction type and agent type are included in the variable elements of the templates at this level. This shows the connection among the different levels and how the concepts at the different levels influence one another.

The descriptions of the elements of each of the templates are presented in the following subsections. The fixed and variable segments of each pattern template are combined to give a complete description of each pattern template. In each template description, the variable elements are described as sub elements under the fixed elements that they are assigned to.

<table>
<thead>
<tr>
<th>Fixed Elements</th>
<th>Variable Elements for Structural Category</th>
<th>Variable Elements for Interactional Category</th>
<th>Variable Elements for Strategic Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classification</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Problem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forces</td>
<td>Autonomy</td>
<td>Interaction type</td>
<td>Goal</td>
</tr>
<tr>
<td></td>
<td>Social Ability</td>
<td>Environment</td>
<td>Flow of control</td>
</tr>
<tr>
<td></td>
<td>Environment</td>
<td>Other agents</td>
<td>Other agents</td>
</tr>
<tr>
<td></td>
<td>Adaptive Behaviour</td>
<td>Control Structure</td>
<td>Agent type</td>
</tr>
<tr>
<td></td>
<td>Intelligence</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Decision and action</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Solution</td>
<td>Components</td>
<td>Communication</td>
<td>Control</td>
</tr>
<tr>
<td></td>
<td>Geometry</td>
<td>Interaction Processing</td>
<td>Autonomy</td>
</tr>
<tr>
<td></td>
<td>Dynamics</td>
<td>Interaction Management</td>
<td>Reactivity</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Deliberation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Knowledge Management</td>
</tr>
<tr>
<td>Known Uses</td>
<td>Adaptation/Integration</td>
<td>Adaptation/Integration</td>
<td>Adaptation/Integration</td>
</tr>
<tr>
<td>Resulting context</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Related Patterns</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.4.3.1. Agent Internal Architecture-Structural Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the two way classification according to [94].

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

**Forces:** describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent technology concepts are introduced as variable elements to the Forces fixed element.

  - **Autonomy:** defines the degree of dependence on client for final decisions; and degree of control over own services
  - **Social ability:** defines the interaction related constraints. Agent interaction may require negotiation strategies; secured messages for instance in military applications.
  - **Environment:** defines the nature of the agent’s environment.
  - **Adaptive behaviour:** defines the need to be able to sense changes in agent’s usage trends and behaviour of the environment and modify behaviour accordingly.
  - **Intelligence:** defines the constraints that relate to the amount of knowledge to be used in decision making.
  - **Decision and action:** defines the level of accuracy and urgency that is required in decision making. Also, it defines constraints related to action utilities and costs.
**Solution:** describes the general solution that best resolves the forces identified. The following variable elements are introduced to capture the notions of agent technology in the template:

*Components:* describes the individual components of the agent’s internal architecture. This element defines how elaborate each of the components should be in the agent’s internal architecture. The components include Knowledge Component, Strategy Component, Interaction Component, and Environmental Interface Component.

The following sub elements have been used in existing pattern templates; however, they are classified as sub elements under Solution to maintain the pattern template structure.

*Structure:* defines the arrangement of the component parts of the architecture. It is usually presented as a diagram.

*Dynamics:* describes how the component parts relate in order to realize the Strategy of the architecture.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. I introduce one sub element as follows.

*Adaptation/Integration:* presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

**Related Patterns:** describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

See Table 1 for a summary of this template.
6.4.3.2. Agent Internal Architecture-Interactional Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the Two Way classification.

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

**Forces:** describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub elements to the Forces element.

*Interaction type:* describes the complexity of the interaction. The complexity of an interaction will determine how the interaction management of the agent should be designed. For instance, argumentative interaction requires more reasoning about messages received when compared with every day trading interaction.

*Environment:* describes issues like perception, action, need for intention recognition in the interaction being modeled

*Other agents:* describes the nature of the other agents that will be involved in the interaction. It states whether the other agents in the interaction are heterogeneous or homogeneous, rational or malicious agents, the number of them, and the relationship with them in the virtual organization.

*Control structure:* describes the flow of control in the agent architecture. The interaction management of agent architecture is impacted by the module that controls the activities of the agent. For instance, in InteRRaP, control flows from the Cooperation layer which is part of the interaction module of the architecture, while in the ADEPT architecture, control flows from the Situation
Assessment Module. The differences in the interaction management structure of these two architectures are partly attributable to the differences in their control flows.

**Solution:** describes the general solution that best resolves the forces identified. The following agent concepts are introduced as sub-elements to the Solutions element.

- **Communication:** describes the agent’s approach for receiving incoming messages or perception handling and sending outgoing messages or action execution. It specifies the messaging mode, the agent communication language and connection with the communication mechanism of the multiagent system.

- **Interaction Processing:** defines the agent’s strategy for generating messages to be sent to other agents, interpreting messages received when necessary, deciding on which agent to interact with in the environment, as well as the strategies for evolving the message construct in an adaptive system.

- **Interaction Management:** describes the agent’s handling of interaction issues like strategy for interaction (negotiation or collaboration or argumentation); evaluating messages received from other agents, and interaction protocol management.

**Known uses:** describes existing applications of the pattern.

**Resulting context:** describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. One sub-element is introduced as follows.

- **Adaptation/Integration:** presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

**Related Patterns:** describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.
6.4.3.3. Agent Internal Architecture-Strategic Patterns

**Name:** describes a name for referring to the pattern and other names by which the pattern is described, if any.

**Classification:** describes the position of the pattern in the two way classification according to [94].

**Problem:** defines the recurring problem that the general solution is defined to solve.

**Context:** describes a particular setting for the problem which provides the basis for defining the relevant forces.

**Forces:** describes the constraints that are relevant to a particular problem based on the context of the problem. The following agent concepts are introduced as sub elements to the Forces element.

- **Goal:** defines the goals that the agent is set to achieve. The goal of an agent will determine if it is to be deliberative and to what degree it is to be deliberative, and to what degree it should be reactive.

- **Flow of control:** defines the flow of control in the virtual organization that is modeled by the multiagent system to which the agent belongs. Flow of control affects autonomy in that the flow of control in the organization determines what an agent can do and the degree of independence it has. A subordinate agent has a less degree of autonomy than a supervising agent.

- **Other agents:** defines the characters and nature of other agents in the multiagent system that the agent has to interact with. If there are malicious agents in the environment, then a tighter control on the internal operations of an agent is necessary. This determines the degree of autonomy of the agent.

- **Agent type:** defines the agent’s type which may include information agents, negotiating agents, assistant agents, and mobile agents. The agent type determines the degree of emphasis on some of the characteristics of an agent in the agent’s architecture. For instance, an automated assembly line agent
requires a higher degree of autonomy when compared with the degree of autonomy required by an assistant agent. Also, a negotiating agent requires higher degree of deliberation when compared with an information agent.

**Solution:** Describes the general solution that best resolves the forces identified. It should be noted that the solution of the patterns in this category will provide the forces for the patterns in the Agent Architecture-Structural category. The following agent concepts are introduced as variable elements to reflect the concepts of agent technology in the description of the patterns Solution:

- **Control:** coordinates the other components of the architecture. It may stand alone or be part of another component. Control handles issues like choice of reactive over deliberative actions (or vice versa). It also handles the autonomy of the agent and goal definition strategy, that is, how the agent decides the goals to commit to.

- **Autonomy:** defines the agent’s degree of control over its internal state and its actions. Different agents in different multiagent system require different degrees of autonomy depending on factors such as the nature of other agents in the multiagent system, the agent type, and flow of control.

- **Reactivity:** defines the agent’s requirement and approach for responding to events in the environment that impact on the achievement of its goal. Different approaches are required for responding to the environment depending on the goal of the agent and its type.

- **Deliberation:** defines the agent’s requirement and strategy for reasoning. The goal of the agent and its type determine to what extent the agent should be deliberative. Reasoning strategies include deduction, and planning/scheduling.

- **Knowledge management:** defines the amount and structure (ontology) of knowledge that the agent should have access to, store or manage. The goal and type of the agent will determine its knowledge management requirements.

**Known uses:** describes existing applications of the pattern.
Resulting context: describes the effects of applying the general solution to the initial context/forces, the advantages of the solution and unresolved forces. I introduce one sub element as follows.

Adaptation/Integration: presents suggestions on how to adapt the pattern to differing projects and how to combine it with other patterns to generate agent pattern languages.

Related Patterns: describes patterns that lead to this or those that follow from the application of this or patterns that are alternates to this pattern.

6.5 Examples of Pattern Descriptions with Pattern Templates

This section illustrates the use of the pattern templates presented in the previous section by describing eight patterns, one from each of the categories of the Two Way classification scheme. The examples presented in this section use the pattern templates defined in the preceding section to structure and organize the information in the patterns’ original descriptions. The agent-oriented pattern examples are presented with the variable elements described as sub elements under the fixed elements that they are assigned to. The patterns described in this section are selected from the twenty eight existing agent-oriented patterns analyzed and classified into the Two Way classification scheme in Chapter Five of this thesis. The eight described patterns according to the categories of the Two Way classification scheme are Structure-in-5, Agent as Mediator, Agent as Delegate, Star-Shaped Movement, Contract Net Protocol, InteRRaP, Ecological Recognizer, and BDI. No MAS Realization pattern is described here as no template was defined for patterns at this level of the Two Way classification scheme (see section 6.4).

6.5.1. Description of Agent-Oriented Analysis Patterns

Two existing agent-oriented patterns are described here to represent the Organizational and Interactional categories of this level.
6.5.1.1. Agent-Oriented Analysis-Organizational Pattern

The Organizational pattern described here is Structure-in-5. It is described using the agent-oriented pattern template presented in section 6.4.1 for this category of patterns. Three other agent-oriented patterns were classified into this category in Chapter Five.

**Name:** Structure-in-5 [65, 66]

**Classification:** Agent-Oriented Analysis-Organizational

**Problem:** How should an existing system be analyzed in order to define an agent based virtual organization, that is, a role-goal structure which can be readily converted to multiagent system architecture?

**Context:** A multiagent system is to be developed for a social organization that features hierarchical relationships, different degrees of control, responsibilities, resources and operations for achieving a set of goals. A social organization whose members work together to achieve corporate goals.

*Domain of Application:* e-commerce applications

**Forces:**

*Goals:* the goals of the organization include maximization of profit on ordinary activities. Sub goals include optimizing the business process of the organization and optimizing the use of resources.

*Coordination/Control:* the structure of a system from which multiagent system is to be developed has middle management with a high degree of autonomy and minimal control from top management. Top management basically provides strategic direction to the middle management. The middle management coordinates the activities of the lower level roles in the system in order to achieve the strategic goals of the system.
**Solution:** Structure-in-5 provides an organizational model with three levels. These levels are the Apex level, Middle management level and the Operational core level.

**Roles:** The roles at the apex level are the Strategic Executive roles. The roles at the middle management level are the Coordination, Middle Agent technology and Support roles. The roles at the operational core level are the Operational Core roles.

**Association/Relationship:** The strategic executive roles which constitute the top of the organization form the apex of the structure. The goal of this apex is strategic management. The roles at the middle level are the Coordination, Middle Agent technology and Support roles. At the lowest level are the Operational core roles, that is, the roles that will carry out the basic tasks and operations that are required to achieve the overall goal of the system. The middle level roles are related to one another by the goal dependencies of Control and Logistics while the operational core roles are related to the middle level roles by task dependencies. See figure 6.1 for the representation of the associations among the roles and goals as well as resources in the Structure-in-5 pattern

**Coordination/Control:** the Strategic Executive roles provide strategic objectives and direction of the organization to the Coordination, Middle Agent technology and Support roles. However, the generation of more detailed plans towards achieving the strategic objectives is left to the Coordination, Middle Agent technology and Support roles. The Coordination, Middle Agent technology and Support roles in turn give directions regarding day to day execution of the plans generated to the Operational Core roles.

**Known Uses:** E-Media, which is a business-to-consumer application that allows customers to buy different kinds of media items on-line [35].

**Resulting Context:** There is clear separation of control and the structure is easily adaptable.
Adaptation/Integration:
- The middle management level can be extended to have more than three roles
- The operational core roles can be split up into departments for better coordination

Related Patterns: Joint Venture, Pyramid [65, 66]

6.5.1.2. Agent-Oriented Analysis-Interactional Pattern
The Interactional pattern described here is Agent as Mediator. It is described using the agent-oriented pattern template presented in section 6.4.1 for this category of patterns. No other agent-oriented patterns were classified into this category in Chapter Five.

This pattern is of a general nature and does present the experience gained from particular projects therefore, some of the variable elements of this category of patterns cannot be described for this pattern.
Name: Agent as Mediator [127]

Classification: Agent-Oriented Analysis-Interactional

Problem: How should the interaction among roles that are not aware of themselves be facilitated?

Context: An organization where a role requires the services of one or more other roles in order to carry out a task or achieve a goal.

Domain of Application: the pattern is of a general nature. The domain of application is not specified (section 5.4.2 of Chapter 5).

Forces:

Interaction objectives: this is not specified as the pattern is of a general nature.

Rules: this is not specified as the pattern is of a general nature

Accessibility: A role in an organization willing to interact does not have information about other roles within the organization with which it can interact. The role does not know which roles exist or are available per time, does not know the capabilities of the roles, does not know the nature and rationality of the roles, does not know the interaction protocol and ontology of the other roles and desires to have safe and secure interaction with some of them.

Solution: A Mediator role holds information about the different roles in the system. It holds information relating to their availability, capabilities, and rationality/credibility. The Mediator role may just serve to connect roles that are willing to interact or it may serve to coordinate the interaction between two roles for example where there are differences in the roles representation of concepts or in order to ensure security of information involved in the interactions.

Interaction type: the interaction type is mediation. An intermediary within the organization connects unfamiliar roles to one another.

Interaction roles: A mediator role is introduced into the modeling of the organization to facilitate interaction among roles that are not aware of
themselves. Analogies of mediator include translators, directories, market makers, rating services. Other interaction roles defined by this pattern are Task, Mediator and Resource.

**Distribution**: The pattern describes different possible distributions of these roles such as Task-Mediator-Task and Task-Mediator-Resource.

**Known uses**: this is not specified as the pattern is of a general nature

**Resulting context**:

**Adaptation/Integration**: the mediation concept can be adapted to different contexts as seen from the analogies of translators, directories, market makers, rating services.

**Related Patterns**: Broker and Embassy patterns [40]

### 6.5.2. Description of MAS Architecture Patterns

Three existing agent-oriented patterns are described here to represent the Definitional, Structural and Interactional categories of this level.

#### 6.5.2.1. Multiagent System Architecture-Definitional Pattern

The Definitional pattern described here is Agent as Delegate. It is described using the agent-oriented pattern template presented in section 6.4.2 for this category of patterns. No other agent-oriented patterns were classified into this category in Chapter Five.

This pattern is of a general nature and does present the experience gained from particular projects therefore, some of the variable elements of this category of patterns cannot be described for this pattern.
Name: Agent as Delegate

Classification: Multiagent System Architecture-Definitional

Problem: How should the role of a user be converted to an agent or agents in an agent based system while maintaining confidentiality of user information?

Context: a user role carries out activities in a system where confidentiality of user information is critical.

Forces:

Goals: to achieve optimum performance and maximize gains by taking decisions based on outcome of activities carried out.

Responsibilities: the responsibilities of this role involve carrying out both non trivial operational tasks and making concluding decisions based on the execution of the tasks carried out. User specific information is used in making the decisions. However, the user information should not be included in the execution of the operational tasks for security and confidentiality reasons.

Solution: this pattern describes an approach for translating a role into agents. It prescribes translating a complex user with sensitive data into two types of agents which are User Agent and Task Agents. The pattern specifies the relationship and control that should exist between these two types of agents.

Agents: this pattern prescribes translating a role with sensitive data with non trivial responsibilities into an Administration Agent and an Operations Agent. Figure 6.2 shows the relationship between the user and the task agents in this pattern.

Role-Agents: this pattern features one role and two agents. The User role is mapped to the Administration (user or assistant) agent and the Operations (task) agent.

Rationale: Divide the role of the user into a User agent and a Task agent. The User agent is the delegating agent while the Task agent is the executing agent.
and will represent the user in different task contexts. The User agent holds the information about the user, user preferences and goals. That is the User agent builds a user profile. The User agent also, updates the user profile as it learns about changes to the user’s preferences. The User agent delegates authority to the Task agent based on the user’s preferences and goals per time and controls the Task agent’s degree of autonomy. The User agent also controls how much information about the user is made available to the Task agent. The number of Task agents that the user will have depends on the complexity and diversity of the tasks to be carried out.

**Resulting context:** the interaction between the assistant agent and the task agents has to be analyzed, modeled and implemented.

**Adaptation/Integration:** a user role can be translated into more than one assistant agents depending on the complexity and volume of the user information and decision making process.

**Related Patterns:** Other patterns by the same author include Agent as Mediator [128].

![Figure 6.2: Agent as Delegate Pattern](image)

**6.5.2.2. Multiagent System Architecture-Structural Pattern**

The Structural pattern described here is Star-Shaped Movement. It is described using the agent-oriented pattern template presented in section 6.4.2 for this category of patterns. Two other agent-oriented patterns were classified into this category in Chapter Five.
Name: Star-Shaped Movement

Classification: Multiagent System Architecture-Structural

Problem: How should a mobile agent efficiently navigate among the different hosts in the multiagent system to achieve its goal?

Context: a system of interconnected hosts with the resources required by an agent to achieve its goals distributed across the network.

Organization of Roles: Not applicable to this problem. The problem relates to a single mobile agent willing to optimize its navigation strategy.

Forces:

Quality goals: quality goals are not stated in the description of the pattern

Knowledge Volume: the amount of knowledge required for the mobile agent to achieve its result is relatively large.

Knowledge Source: the sources of knowledge in the multiagent system are the different hosts on the network.

Infrastructure: the infrastructure of the multiagent system is a network of hosts. The hosts with the resources are predetermined and are relatively stable. The network bandwidth is also relatively broad.

Solution: In the Star-Shaped Movement pattern, the mobile agent moves from the base host to a destination host holding some of the resources, executes its task at the destination host and returns to the base host. It moves to another destination host and returns to the base host after executing its task there. It navigates all the required hosts on the network by the base host to destination host and back to base host movement until the goal is achieved. Figure 6.3 is a representation of the Star-Shaped Movement pattern.

Agents: the only agent in this pattern is the mobile agent

Other entities: the other entities in this pattern are the hosts on the network that hold the resources required by the agent to achieve its goal
**Geometry**: this pattern is not about the arrangement of agents to form an architectural model. Geometry is therefore not a part of the solution of this pattern.

**Known uses**: Not mentioned in the pattern description

**Resulting context**: this pattern allows the user of the system to monitor the developments to the resources at the hosts after each return of the mobile agent to the base host

**Adaptation/Integration**: in a system where the mobile agent has to visit many hosts, this pattern could be adapted such that it visits two hosts and executes its tasks there before returning to the base host.

**Related Patterns**: Branching and Itinerary patterns [126]

6.5.2.3. **Multiagent System Architecture-Interactional Pattern**

The Interactional pattern described here is Contract Net Protocol. It is described here using the agent-oriented pattern template presented in section 6.4.2 for this category of patterns. Four other agent-oriented patterns were classified into this category in Chapter Five.

**Name**: Contract Net Protocol [122, 32]

**Classification**: Multiagent System Architecture-Interactional

**Problem**: How should an agent execute a task that requires other agents to perform?

**Context**: A multi agent system where an agent requires the services of one or more other agents to carry out a task.

**Forces**:
**Interaction Objective:** the objective of this interaction is to optimize a function (e.g. price) that characterizes an agent’s task.

**Agent Types:** the agents are non-malicious agents; there is a degree of competition amongst the service provider agents; the client agent has knowledge of addresses of the provider agents.

**Adaptive Behaviour:** there may be changes in individual agents’ internal business rules over time. Such changes when they occur may change the agents’ evaluation of proposals or offers.

**Security:** the interaction in the multiagent system does not significantly impact on the security of the system. **Time:** time is usually not a critical consideration in this interaction. However, there are deadlines at points in the interaction when proposals should be presented.

**Language Constraint:** there are no language constraints. The agents in the multiagent system have the same representation of the real world as it relates to the task to be carried out.

**Error Sensitivity:** it is essential that the content of the interaction be accurate as decisions to accept or decline an offer is based on the details of each proposal.

**Solution:** the Contract Net pattern approaches this problem by assigning the role of a manager to the agent that owns the task to be carried out. The manager agent sends a request to the potential contractors (other agents that can carry out the task). The proposal from the contractors are assessed by the manager and the task is awarded to one (or more) of them that best optimizes the characteristic function of the task. The contractor agent(s) then send the result (or no result) of the task execution to the manager agent and the interaction is ended.

**Interaction Strategy:** the Participants in Contract Net include one Manager Agent (task owner) and a known number of Contractor agents (probable task executors). The manager agent has direct interaction with each of the contractor agents. The contractor agents do not interact with one another.

**Introduced Entities:** Contract Net does not introduce agents into the multiagent system in order to solve the interaction needs.
**Ontology:** the agents involved are assumed to have the same representation of the real world as it relates to the task they want to perform. Therefore, Contract Net does not have ontological specifications.

**Interaction Mechanism:** Contract Net is a negotiation type interaction. The time for the presentation of proposals is deadline controlled. The participants must respond to every message sent. Contractor agents must commit to executing the task awarded to them. The manager agent sends out a call for proposal to the probable contractor agents. Each contractor agent then sends a proposal for the task execution to the manager agent within a stipulated time frame. The manager agent responds to all the proposals by either accepting or rejecting them. Each contractor agent whose proposal is accepted sends the result of the task execution to the manager client and the interaction ends. Figure 6.3 shows the dynamics of the Contract Net Protocol.

**Communication/Connectivity:** this pattern is not about low level data transmission, hence there are no communication specifications in the pattern.

![Figure 6.3: The FIPA Contract Net Protocol](image)

**Known Uses:** Intelligent Manufacturing [99, 36], Electronic Marketplace [21]
**Resulting Context:** This solution does not cater for capturing the adaptive nature of the multiagent system as it affects interaction; there is no provision for handling language constraints if it exists among the participating agents.

**Adaptation/Integration:** The relative simplicity of Contract Net Protocol allows for easy adaptation. Examples of adaptations of Contract Net include the Iterated-Contract Net [32]. Provisional Agreement Protocol (PAP) [106]. Contract Net could be integrated with other patterns e.g. Embassy pattern [40] in a case where manager and contractor agents have different representations of the world.

**Related Patterns:** Broker [40], Request [32]

### 6.5.3. Description of Agent Internal Architecture Patterns

Three existing agent-oriented patterns are described here to represent the Structural, Interactional and Strategic categories of this level.

#### 6.5.3.1. Agent Internal Architecture-Structural Pattern

The Structural pattern described here is InteRRaP. It is described using the agent-oriented pattern template presented in section 6.4.3 for this category of patterns. One other agent-oriented pattern was classified into this category in Chapter Five.

As discussed in section 5.4.3 of Chapter Five, the InteRRaP has features that qualify it to belong to the Structural and Strategic categories of this level. Therefore, this description of the pattern presents information that is relevant to the structural aspect of the pattern. The information that relates to the strategic aspect of the pattern such as how it achieves the right blend of reactive and deliberative behaviour and the knowledge management mechanism are not included in this description of the pattern.
**Name:** InteRRaP Architecture [87, 88]

**Classification:** Agent Internal Architecture-Structural

**Problem:** How to build agent architectures that combine the advantages of reactive and deliberative architectures?

**Context:** Designing agent architectures for a process environment with distributed control and joint planning (e.g. flexible production systems).

**Forces:**

- **Autonomy:** the agent is required to make timely decisions based on changes in operating conditions.
- **Social Ability:** the agent requires extensive collaboration with other agents for cooperative planning. Interactions are flexible and generalized.
- **Environment:** the agent’s environment is relatively closed and observable. The environment is relatively deterministic and non-episodic, that is, there is a connection between different scenarios.
- **Adaptive Behaviour:** the agent has to operate in a self organizing system in response to different environmental changes.
- **Intelligence:** the agent requires high degree of flexibility in interaction and processing. Optimization of time and resources is critical.
- **Decision and action:** the agent’s decisions and actions should be accurate.

**Solution:** The InteRRaP architecture is composed of a world interface, a knowledge base and a control unit.

- **Strategy Component:** The flow of control in InteRRaP is vertical. It is a 2-pass vertically layered architecture with three layers of control which are Behaviour-based Layer (BBL), Local Planning Layer (LPL) and the Cooperative Planning Layer (CPL). CPL and LPL are the deliberative layers while the BBL is the reactive layer.
**Knowledge Component:** This component of InteRRaP is made up of three layers which are the World Model, Mental Model and Social Model. These layers hold information that corresponds to the BBL, LPL and CPL respectively.

**Interaction Management:** Interaction management is centered on joint planning.

**Environmental Interface:** The World Interface component is used for all interaction with the agent’s environment. Only the BBL and world model layers have direct connection with the world interface component. All necessary client interface and communication with other agents are implemented in this component.

**Geometry:** The InteRRaP components are structurally arranged as presented in figure 6.4.

**Dynamics:** Information from the environment is received into the world model layer of the knowledge base through the world interface component. The world model is updated with the received information using the belief revision function. The information is then passed on to the mental model and the social model layers. The updated world model is also passed to the SG function of the BBL which uses it to generate a new set of goals (if necessary). The new set of goals is sent to the PS function of the BBL and appropriate reactor patterns of behaviours (PoB) are activated. If the new set of goals cannot be handled by the PS function of the BBL, bottom-up activation is used to transfer control to the LPL (and then to the CPL). The LPL will then use top down execution to pass appropriate plans (probably from the CPL) to the BBL for execution through appropriate procedure PoB.

**Known Uses:** These include FORKS [28], a system for simulation of an automated loading dock. FLIP [55] (FLEXible Inter Processing).

**Resulting Context**
Planning: the planning capability is a major feature of the InteRRaP architecture.
Environment: it may not function optimally in an open environment with unpredictable and uncontrolled exposure to changes in the environment.

Competition: it may not function optimally in a competitive environment with possible malicious agents.

**Adaptation/Integration:** the separation of the knowledge base from the control unit facilitates ease of adaptation. A control layer with its corresponding knowledge base could be separated and adapted. The BBL and world model are separated and adapted in the application of InteRRaP to the FLIP system [55].

**Related Patterns:** BDI architecture [109]

---

**Figure 6.4: The InteRRaP Agent Architecture**

### 6.5.3.2. Agent Internal Architecture-Interactional Pattern

The Interactional pattern described here is Ecological Recognizer. It is described using the agent-oriented pattern template presented in section 6.4.3 for this category of patterns. One other agent-oriented pattern was classified into this category in Chapter Five.

**Name:** Ecological Recognizer [41]

**Classification:** Agent Internal Architecture-Interactional

**Problem:** How should the architecture of an agent be structured to recognize the intentions of other agents in its environment?
Context: An intelligent agent (recognizing agent) must recognize the intention of other agents (intending agents) but has access only to the state data. Furthermore, the state data is complex and the intentions to be recognized are difficult to describe by sets of rules.

Forces:

Interaction type: the type of interaction is intention recognition. Direct interaction is not feasible as the agents are competitors (for example, agents of two separate armies in a military context).

Environment: there is no access to data other than state data and there is a lot of state data. The environment is a military context with situational dynamism, uncertain knowledge and identification problems. The recognizing agent senses the actions of the intending agent which manifest in the environment as state data.

Other agents: the other agents involved in this interaction are heterogeneous and they are adversaries. The other agents are rational but dangerous and will not engage in direct interaction.

Control structure: Control flows from the Reasoning module of the architecture. Therefore, the descriptions of the recognized intentions have to be passed to the Reasoning module.

Solution: the basic architecture of a recognizing agent is made up of three components which are perception, reasoning and action. To address this problem, include a recognition module called Pattern Matcher in the architecture of the recognizing agent. This pattern matcher is however a part of the perception component of the agent and it maps the pattern of incoming data stream onto learned, stored or otherwise supplied examples of the recognized intention (figure 6.5).

Communication: the perception mechanism of this agent is sensing the state data in order to capture the actions of the intending agent. The action execution mechanism of the agent is not mentioned in this pattern as it is about intention recognition.
**Interaction Processing:** not applicable to this kind of context where there are no direct exchanges of messages

**Interaction Management:** the Pattern Matcher module is used to evaluate the sensed data and to map it to examples of recognized intentions in the agent’s knowledge base. The recognized intentions are then forwarded to the Reasoning module of the recognizing agent.

**Known uses:** this pattern is used in air combat simulation [42, 104]

**Resulting context:** the application of this pattern produces the benefits of modularity, ontological purity, fuzziness and technological suitability. However, modularity creates the need for modeling and implementing interactions between the reasoning and intention recognition processes of the agent. Other disadvantages include technological complexity, visibility and performance trade offs.

**Adaptation/Integration:** the pattern is simplified by changing the information received from the environment from sensed data to intending agent’s action state

**Related Patterns:** Assisted Ecological Recognizer, Hybrid Recognizer and Sense and Infer [41]

---

**Figure 6.5: Ecological Recogniser Pattern**
6.5.3.3. Agent Internal Architecture-Strategic Pattern

The Strategic pattern described here is BDI. It is described here using the agent-oriented pattern template presented in section 6.4.3 for this category of patterns. Six other agent-oriented patterns were classified into this category in Chapter Five.

**Name:** BDI Agent Architecture [109, 107, 8]

**Classification:** Agent Internal Architecture-Strategic

**Problem:** How should agents that need to reason in response to external events while pursuing specific goals be designed?

**Context:** Designing an architecture for agents that reason about alternative objectives and actions. The environment of the agent is dynamic. There are different courses of actions for each objective chosen.

**Forces:**

*Goal:* An agent’s goals are structured in a hierarchy of goals and sub goals. To achieve a goal, the agent has to make a choice among alternative sub goals of the goal that is to be achieved. Changes in the state of the environment may render a chosen sub goal incapable of achieving the parent goal.

*Flow of control:* the control structure is such that the agent has appreciable degree of autonomy

*Other agents:* there is the possibility of ‘malicious’ agents in the environment, hence information security and control over internal state is critical to safety and success of the agent.

*Agent type:* possible agent types include information agents, negotiating agents, assistant agents, mobile agents

**Solution:** BDI models a rational agent using the mental attitudes of beliefs, desires and intentions. The agent’s knowledge is captured in beliefs such as the information about the agent’s current environment. Desires represent the agent’s
current possible courses of actions i.e. the different goals the agent could pursue. An intention could be described as the goal or sub goal that the agent commits to at a particular point in time. The BDI architecture takes a perceptual input from the environment and feeds this into the belief revision function for necessary updates to the belief base. Updates to the beliefs are passed to the option generation and filter functions. The updates passed to these two functions are combined with the existing state of the intentions to update both the desires and intentions as necessary. Current intentions are fed into the action selection function to define an appropriate plan of action for achieving these intentions. Figure 6.6 shows the reasoning process of BDI.

![Figure 6.6: The BDI Architecture](image)

**Control:** BDI attempts to balance both reactive and deliberative behaviour by having pre-determined plans to achieve goals, with triggers as to when these plans are applicable

**Autonomy:** BDI achieves autonomy by the interaction of its mental attitudes. It regularly updates its belief base and uses that to modify its desires. The intentions are determined by the current state of the desires. Therefore, the actions the agent commits to at any point in time are determined by the interplay of its beliefs and desires.
**Reactivity:** Action Execution Strategy: Means end reasoning is carried out by the action selection function. This function is designed to generate alternate plans to achieve a given intention in case a plan fails while the intention persists.

**Deliberation:** Goal Definition Strategy: BDI architecture is based on practical reasoning. Practical reasoning involves deliberation and means end reasoning. Deliberation in the BDI model is iterative as a decision has to be made to either remain committed to an intention or to drop it. BDI uses the filter function and input from the knowledge bases to generate new intentions.

**Knowledge management:** The knowledge used by the BDI model includes information about the environment (beliefs) and the set of alternative sub goals (desires). BDI uses four of its seven units to handle knowledge. One is the informative unit which holds the information about the current state of the environment i.e. the current set of beliefs. Two is the belief revision function which updates the informative component. The belief revision function decides on the need to update the informative component by comparing current perceptual input with existing information in the informative component. Three is the option generation function which determines the agent’s current set of desires. It uses the agent’s current beliefs and intentions to determine the current set of desires. And four is the desires unit which holds the agent’s current set of desires.

**Known uses:** The OASIS air traffic management system [78]; the IRTNMS network management system [108]

**Resulting context:** Commitment to intentions: requires maintaining a balance between adequate and excessive reconsideration of an intention that the agent has committed to.

Structure: provides a good functional decomposition of the architecture into data structures and functions. This makes it easy to map the model to implementation.

Interaction/Interface: the BDI architecture does not describe the interaction component. Also, the structure of the interface with the environment is not captured
in the BDI architecture. BDI only mentions perceptual input and action output. It gives no suggestion regarding the structure of these components.

**Adaptation/Integration:** The functional decomposition of the architecture makes it readily adaptable. The BDI model needs to be integrated with patterns that deal with the Control, Interaction Management and Environmental Interface components of the general agent architecture.

**Related Patterns:** InteRRaP [87]

### 6.6 Chapter Summary

The pattern template is the vehicle for conveying the experiences that patterns are meant to share. According to the definition of a pattern, a good pattern form should explicitly feature a specific recurring *problem* in a *context* that defines a set of *forces* which are resolved by a general *solution* to create a *resulting context*. On the other hand, the description of a pattern should contain only required but adequate information of the experience to be shared. The pattern template should be structured to feature the appropriate elements for describing just enough information about the experience to be shared. That is the pattern template should be concise but sufficient. Striking the balance between what is concise and what is sufficient is a challenge for the pattern template structure. Since patterns are media of communication, it is important that the pattern template be structured and designed for a broad range of practitioners to identify with. That is, the pattern template should be universal in appeal. An inappropriate or inadequate pattern template can adversely affect the usefulness of patterns described with it.

This Chapter designed an agent-oriented pattern template structure. The template structure features two segments which are a fixed segment and a variable segment. The Fixed segment ensures universal communication of agent-oriented patterns as it comprises of elements that are fundamental components of software patterns. The variable segment comprises of elements that are concepts of agent technology and therefore enhance the comprehension and adoption of agent technology. Agent-
oriented pattern templates were derived for the eight categories of the Two Way classification scheme in order to achieve relevance of description templates to the different categories of patterns. Eight agent-oriented patterns were described using these description templates to illustrate the expressiveness and relevance of the description templates.
Chapter 7

Evaluation and Conclusions

7.1. Evaluation

The Two Way classification scheme and the Attribute-based Analysis process were presented at a seminar run by the agent association in Victoria, Agents-Vic\(^4\). The seminar audience was made up of more than 20 agent technology researchers from different university research groups and industry practitioners.

After the presentation, the attendees were divided into seven groups with approximately three people in each group for the purpose of participating in an interactive session. The interactive session’s task required each group to analyze and classify an agent-oriented pattern according to the categories of the Two Way classification scheme using attribute-based analysis. Six agent-oriented patterns by different authors were used in the exercise. The patterns are Branching [126], Ecological Recognizer [41], Embassy [40], Basic Negotiating Agent [4], Pyramid [65], and Master-Slave [2]. Each group was provided with a description of one of the patterns, blank level and category attributes tables, the Two Way classification scheme table and the eight description templates. Each group read the description of the pattern given to it then, the members collaborated to follow the attribute-based analysis for analyzing and classifying the pattern.

The results were quite impressive as the classifications of the patterns by the attendees are close to the results presented in table 5.6 of chapter 5. The attendees’

\(^4\) Agents-Vic is an organization that was formed to foster research and development among agent technology research groups, industry and government institutions in Melbourne, Victoria. [http://www.agents.org.au/](http://www.agents.org.au/)
vertical classifications of these patterns are same as in table 5.6 in almost all cases while those that analyzed the Branching pattern classified it into the same vertical and horizontal classifications as in table 5.6.

After the interactive session, the attendees gave feedback on the comprehensibility and applicability of the Two Way classification scheme by responding to a set of six questions. Three of the questions relate to classification of agent-oriented patterns and the attribute-based analysis process. The other three questions relate to the description templates. The set of six questions and a summary of the attendees’ responses (Table 7.1) are presented as follows:

1. Do you understand the Two Way classification scheme?
   - Yes (Y)  - No (N)  - Unsure (U)

2. Do you believe the Two Way classification scheme will be helpful for classifying agent-oriented patterns?
   - Yes (Y)  - No (N)  - Unsure (U)

3. Is the attributes-based analysis intuitive to you?
   - Yes (Y)  - No (N)  - Unsure (U)

4. Is the description template structure readily comprehensible?
   - Yes (Y)  - No (N)  - Unsure (U)

5. Do you believe the description templates will improve the comprehension of the information presented by the pattern?
   - Yes (Y)  - No (N)  - Unsure (U)

6. Can the description templates be applied to any aspect of your work?
   - Yes (Y)  - No (N)  - Unsure (U)
Eighteen (18) attendees responded to the feedback questions. 44% of the attendees stated that they understood the Two Way classification scheme, 50% believe the classification scheme will be helpful for classifying agent-oriented patterns and 44% consider the attributes-based analysis intuitive. 33% of the attendees consider the description template structure readily comprehensible, 56% believe the description templates will improve the comprehension of information presented in patterns. 28% of the attendees suggest the description template can be applied to some aspect of their work.

These responses show that after only a short introduction of about 20 minutes to the Two Way classification scheme and the attribute-based analysis, an appreciable number of practitioners (based on the number of practitioners that took part in the activity) were able to understand and apply them. The responses also show that a good percentage of practitioners were able to appreciate the potentials of the description templates in improving the communication of agent-oriented patterns. This evaluation outcome therefore demonstrates the intuitive nature of the Two Way classification scheme, the attribute-based analysis process and the description templates. Also, based on this evaluation results, it can be safely concluded that given adequate exposure to the concepts and contents of the Two Way classification scheme, the attribute-based analysis process and the description templates, software practitioners will have increased comprehension of agent-oriented patterns and agent technology concepts and potentials.
7.2. Conclusion

This thesis has addressed the challenges facing agent-oriented software engineering and the need to improve the adoption of agent technology by the wider software engineering community. Two areas explored in detail are interaction analysis and experience sharing through the use of software patterns. Each of these areas is reviewed in turn.

7.2.1. Interaction Analysis

Interaction is a common characteristic of multiagent systems, regardless of the complexities or sophistication or simplicity of the individual agents in the multiagent systems. The interactions among the components of a system provide a basis for analyzing and understanding the behaviour of the system. Hence, interaction analysis has the potential to deal with agent-oriented software engineering challenges relating to modeling the behaviour of the system. Analyzing multiagent systems by the interactions within such systems provides a universal platform for reasoning about the concepts and applicability of agent technology and hence improves the appreciation of the technology.

An agent interaction protocol conceptualizes, models, and implements the interactions in agent-based systems. Agent Interaction Protocols give context and direction to the interactions in multiagent systems. Agent Interaction Protocols constitute a crucial aspect of the development, the implementation and the operation of multiagent systems because interactions drive the overall behaviour of the multiagent systems and influence the decision processes of the agents in the system.

By their function, agent interaction protocols represent some of the peculiar characteristics and behaviour of a multiagent system’s application domain. Hence, the applicability of multiagent systems to different application domains is affected by the development of the agent interaction protocols in the system.
Development of agent interaction protocols should be based on the characteristics of the application domain of the multiagent system that the protocol is designed for. This is in recognition of the influence of the properties of individual application domains on the effectiveness and applicability of the agent interaction protocol being designed.

Also, development of an agent interaction protocol requires a structured definition of the components of an agent interaction protocol and their relationships. Defining a structure for agent interaction protocols facilitates reusability as it makes it easy to customize a protocol for different applications.

To this end, a dedicated approach for developing agent interaction protocols is presented. Also, an agent interaction protocol structure (protocol structure) and an agent interaction protocol property suite (protocol property suite) were presented. The protocol structure and the protocol property suite are the end products of the dedicated approach for developing agent interaction protocols.

The dedicated approach for developing agent interaction protocols involves two phases. The phases are the Domain-directed Analysis phase and the Design/Verification phase. This approach provides a well defined process for the analysis and design of agent interaction protocols which facilitates productive reusability based on the agent interaction protocol structure and the protocol property suite.

7.2.2. Experience Sharing through Software Patterns

Agent-oriented patterns by their structure depict the connection between software development problems and the concepts of agent technology that are applied in solving such problems. Agent-oriented patterns describe examples of agent-oriented software engineering solutions that bring the concepts of agent technology to light. Patterns don’t just describe processes or solutions, the problem that necessitates these solutions or processes and the contextual characteristics of the
particular problem that justifies the solution are also described. This description of what necessitates or justifies a particular solution in software development enhances the comprehension of the concepts of agent technology used in the solution.

The application of software patterns to agent technology allows the presentation of the concepts and principles of agent technology to cut across a wider community of software practitioners.

Reflecting the notions of agent technology appropriately in agent-oriented patterns helps software practitioners who come in contact with such patterns to comprehend the capabilities and concepts of agent technology as well as the issues in engineering multiagent systems.

The notions of agent technology can be reflected in the classification and description of agent-oriented patterns. A classification scheme provides the platform for classifying, analyzing and describing agent-oriented patterns. Agent-oriented pattern classification scheme should incorporate the concepts of agent technology into the categorization, analysis and description of agent-oriented patterns.

To this end a Two Way classification scheme is presented to provide the platform for classifying, analyzing and describing agent-oriented patterns. The levels of abstractions in agent-oriented software engineering provide the basis for the design of the Two Way classification scheme. This classification scheme comprises of two dimensions, the vertical dimension and the horizontal dimension. The vertical dimension is based on the levels of abstractions in agent-oriented software engineering while the horizontal dimension is defined by the software development tasks at each of the levels of abstractions.

Each category in the Two Way classification scheme is therefore defined by the level of abstraction and a development task at that level of abstraction. This
definition of the categories ensures that agent-oriented patterns are classified according to the concepts they convey and the levels of abstractions they relate to. Also, this definition of the categories provides a guide for the identification of the properties of the categories.

Categorizing agent-oriented patterns in the Two Way classification scheme, like any other classification scheme, is based on the premise that patterns that possess similar properties are placed in the same category. The properties that are common to the patterns in a category are the attributes of the category. In other words, the defining characteristics of a category of the Two Way classification scheme are the set of the common attributes of the patterns that belong to that category.

The attributes of the categories of the Two Way classification scheme present the criteria for the analysis of agent-oriented patterns and the basis for the design of agent-oriented pattern templates.

To this end, the properties of the levels and categories of the Two Way classification scheme were defined. Level attributes are the attributes of the patterns that belong to a particular level of the Two Way classification scheme. Category attributes are the attributes of the patterns that belong to a particular category of the Two Way classification scheme.

The usefulness of the Two Way classification scheme can be demonstrated by classifying agent-oriented patterns from different sources according to the categories of the classification scheme. In order to ensure consistency in the way patterns from different sources are classified according to the categories of the Two Way classification scheme, a repeatable process is required for determining the category of the classification scheme that the patterns belong to.

A process for analyzing agent-oriented patterns using level and category attributes as the basis of the analysis was presented. This pattern analysis process is referred to as attribute-based analysis.
Attribute-based analysis is the examination of an agent-oriented pattern to determine its attributes and use of those attributes to evaluate the pattern for different purposes. Attribute-based analysis is useful for assessing the versatility of an agent-oriented pattern’s solution, assessing the quality of an agent-oriented pattern, and classifying existing agent-oriented patterns according to the categories of the Two Way classification scheme.

Level attributes table and category attributes tables which are the tools of attribute-based analysis were described. Twenty eight existing agent-oriented patterns, out of ninety seven patterns studied were analyzed and classified according to the categories of the Two Way classification scheme using attribute-based analysis. The observations noted in the process of analyzing these patterns were documented. These observations relate to the quality and complexity of the patterns analyzed.

The pattern template is the vehicle for conveying the experiences that patterns are meant to share. According to the definition of a pattern, a good pattern form should explicitly feature a specific recurring problem in a context that defines a set of forces which are resolved by a general solution to create a resulting context. On the other hand, the description of a pattern should contain only required but adequate information of the experience to be shared.

Agent-oriented patterns should be clear and complete and expressed with minimal complexities. That is, agent-oriented patterns should present clear presentation of the applicability of the patterns such that software practitioners can understand the contexts to which the solutions and concepts of agent technology are applicable. Also, agent-oriented patterns should present clear description of the concepts in the solution to facilitate comprehension by non agent software practitioners. To this end, a template structure for describing agent-oriented patterns is presented. This template structure is adaptable but consistent. It is adaptable as it can be used to derive customized description templates for the different categories of agent-
oriented patterns. However, the template structure is consistent across the different description templates derived from it. This template structure is designed to feature the concepts of agent technology in a description format that is familiar to the wider software engineering community thereby enhancing the communication and comprehension of agent technology.

The template structure features two segments which are a fixed segment and a variable segment. The Fixed segment ensures universal communication of agent-oriented patterns as it comprises of elements that are fundamental components of software patterns. The variable segment comprises of elements that are concepts of agent technology and therefore enhance the comprehension and adoption of agent technology. Agent-oriented pattern templates were derived for the eight categories of the Two Way classification scheme in order to achieve relevance of description templates to the different categories of patterns.

In summary, the contributions of this thesis are in the areas of multiagent systems interactions and agent-oriented patterns.

The contributions in the area of multiagent system interactions include a structured definition of agent interaction protocols and a dedicated approach for analyzing multiagent system interactions and developing agent interaction protocols.

The contributions in the area of agent-oriented patterns include a Two Way classification scheme for classifying agent-oriented patterns; the definition of agent-oriented pattern attributes according to the levels and categories of the Two Way classification scheme; an attribute-based analysis process for analyzing agent-oriented patterns; and a template structure as well as templates for describing agent-oriented patterns.

It is believed that these contributions can enhance the comprehension of agent technology and hence, the adoption of the technology by the wider community of software practitioners.
Bibliography


Foundation for intelligent physical agents, FIPA. *Agent communication language.* In http://www.fipa.org


37 Guerin F. Specifying Agent Communication Languages. PhD 2002, Imperial College of Science, Technology and Medicine University of London.


48 Jennings N. R. (1999) "Agent-Oriented Software Engineering" Proc. 12th Int Conf on Industrial and Engineering Applications of AI, Cairo, Egypt, 4-10. (Invited paper) [Also appearing in Proc. 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW-99), Valencia, Spain 1-7 (Invited paper)]


International Conference on Autonomous Agents, pp. 92-9, Minneapolis, Minnesota, USA, May 1998.


71 Koning J.L.. Designing and testing negotiation protocols for electronic commerce applications. 34-60 Electronic Edition (Springer LINK)


75 Lesser, V.R., Cooperative multiagent systems: a personal view of the state of the art Knowledge and Data Engineering, IEEE Transactions on Volume 11, Issue 1, Jan.-Feb. 1999 Page(s):133 – 142


Sauvage S., MAS Development: Reusing through Agent-Oriented Design Patterns, Eight World Multi-Conference on Systemics, Cybernetics and Informatics, SCI'04, Orlando, 18-21 July 2004.


Weiss, M., *Patterns for Motivating an Agent-Based Approach*, *Conceptual Modeling for Novel Application Domains (AOIS@ER)*, LNCS 2814, 229-240, 2003


Zanella G., 2004. *"Social Interactions and Economic Behavior"* Department of Economics, University of Siena 441,
Zhao X., Chan K., Li M. *Applying agent technology to software process modeling and process-centered software engineering environment.* SAC 2005: 1529-1533
Author/s:
Oluyomi, Ayodele O.

Title:
Patterns and protocols for agent-oriented software development

Date:
2006-11

Citation:

Publication Status:
Unpublished

Persistent Link:
http://hdl.handle.net/11343/39172

File Description:
Patterns and protocols for agent-oriented software development

Terms and Conditions:
Terms and Conditions: Copyright in works deposited in Minerva Access is retained by the copyright owner. The work may not be altered without permission from the copyright owner. Readers may only download, print and save electronic copies of whole works for their own personal non-commercial use. Any use that exceeds these limits requires permission from the copyright owner. Attribution is essential when quoting or paraphrasing from these works.