A Novel Development Methodology for Cooperative, Distributed Multi-agent Systems

Ebrahim Fahad Al-Hashel

Master of Science in Computer Science (University of Sheffield – UK)

FACULTY OF INFORMATION SCIENCES AND ENGINEERING

UNIVERSITY OF CANBERRA
AUSTRALIA

This dissertation is submitted in fulfilment of the requirements for the award of the degree doctor of philosophy

November 2010

Abstract

The multi-agent systems (MaS) typology can be divided into “independent” and “cooperative” or closed and open respectively. Independent MaS embraces a set of agents linked together by predefined protocols that operate in a closed central control system. The closed system does not include or perform MaS dynamic behaviour; rather, it mainly performs agent team formation processes and promotes agents’ cooperation at runtime. This type of MaS is appropriate for application within fixed procedures that operate in one system boundary. In contrast, the cooperative MaS is an emergent system that has the potential to dynamically, at runtime, search in an open distributed computational environment and subsequently form a team of appropriate agents to achieve the defined goal. The agent cooperation behaviour is a key strength of MaS, which is characterised by agents’ autonomy.

This thesis investigates existing multi-agent system development methodologies: Prometheus, Gaia, MaSE, PASSI and Tropos. The results indicate that these methodologies are engineering an independent MaS focusing on the agent’s internal structure or system architecture through interaction protocols. However, the cooperative MaS development processes are minimally realised in these methodologies and the agent cooperation process is not implicitly addressed.

Further, the research aims to enhance MaS development methodology by proposing a novel development methodology for multi-agent systems (DMMAS) that can guide software practitioners in developing cooperative MaS with the ability to function in large-scale, open, distributed, incremental, heterogeneous systems. It is motivated by distributed architecture for problem solving in domains including military logistics, healthcare, transportation and travel agency systems. The research attempts to transition existing MaS from independent concepts to cooperative concepts.

To model agent autonomous behaviour, the research proposes a new organisational multi-agent systems architecture supported by an ontology-based search model and the agent cooperation, through dynamic team formation process is built on agent

adoptive strategy and *Share Plan* cooperation theory as an important characteristic of DMMAS.

The research has been conducted using design science in information system research method, and the case study research approach. For proof of the concept the research applied DMMAS development methodology on a real world case study "Travel Agency System (TAS)" which served as the motivating problem for the research work. The results are evaluated using a benchmark approach to compare DMMAS performance with the five existing MaS development methodologies.

This thesis makes four main contributions: first, it enhances the agent-based system by providing a new development methodology with an attempt to develop the multi-agent systems current state of the art from independent to cooperative. Secondly, the research presents new multi-agent systems architecture and a methodology on how to design and develop open distributed multi-agent systems. Thirdly, the research proposes how ontology analysis and design can be incorporated in software engineering practice. The research explains how ontology concepts, objects and relationships are identified to build the agent systems domain. Finally, the research introduces a new agent functionality ontology schema for a search to replace the agent name keyword based conventional search. The functionality based ontology approach utilises descriptor based semantics.

The proposed DMMAS design methodology is evaluated against software engineering principles and its strengths and inadequacies investigated. The research achievements are summarised and emerging research questions are outlined for future work.

Acknowledgement

To H.E. Major General Dr. Sheikh **/ *Mohammed bin Abdulla Al Khalifa***, Minister of State for Defence Affairs. You are an extra ordinary person; your leadership has guided me to achieve this work.

To my supervisor Professor **/ *Dharmendra Sharma***, thanks for being so kind and so supportive. His guidance is crucial to this thesis, and the will to drive the research forward.

To my brother ***Yousif Al-Hashel***, thanks for the support and continues asking.

To my brother in life Brigadier General / ***Ali bin Sager Al-Nuaimi***, I appreciate the efforts and the support you offered to me to make this work possible.

To ***Jennifer Bradley***, thanks for the effort you contributed to edit this thesis.

To Dr. / ***Masoud Mohammadian***, Mr. / ***Robert Cox***, Dr. / ***Bala Balachandran***, Dr. / ***Wanli Ma***, thanks for the encouragements.

To my best friends in Australia ***Sisira Adikari***, and ***Patrick Collin***, stay in contact I enjoy yours friendships.

Dedication

To my family:

My wife: Anissa Al-Rowaie

My sons: Ahmed and Sufyan

Yours truly, love and patience, yours constant encouragement and support provided a solid result. Being away from you was a tough time, all what I have done was for a better life.

To my parents:

Fahad Al-Hashel and Amina Al-Thawadi

Table of Contents

List of Figures

List of Tables

List of Acronyms

The following acronyms and abbreviations of standards phrases are used throughout the thesis:

ARPA	Advance Research Project Agency
AbSE	Agent-based Software Engineering
AbS	Agent-based System
AOSE	Agent-Oriented Software Engineering
AOS	Agent-Oriented Software
AI	Artificial Intelligence
BDI	Belief, Desire, and Intention agent architecture
DARPA	Defence Advance Research Projects Agency
DFI	Design Fabricator Interpreter
DMMAS	Development Methodology for Multi-agent Systems
DAI	Distributed Artificial Intelligence
DPS	Distributed Problem Solving
DPS	Distributed Problem Solving
ERD	Entity Relationship Diagram
EVO	Evolutionary Delivery
SXML	Extensible Markup Language Schema
XML	Extensible Markup Language
EP	Extreme Programming
FIPA	Foundation for intelligent physical agents (FIPA)
HERM	High Entity Relationship Diagram
IS	Information System
JVM	Java Virtual Machine
JAD	Joint Application Development
JAD	Joint Application Development
KIF	Knowledge Interchange Formalism
KQML	Knowledge Query Manipulation Language
KSE	Knowledge Sharing Effort
KSE	Knowledge Sharing Effort
MaS	Multi-agent Systems

NII	National Information Infrastructure
OMT	Object Modelling Technique
OOSE	Object-oriented Software Engineering
OWL	Ontology Web Language
PTA	Planed Team Activity
PASSI	Process for Agent Societies Specification and Implementation
Pa	Professional-agent
PDT	Prometheus Development Tools
RAD	Rapid Application Development
RAD	Rapid Application Development
RDFS	Resources Definition Framework Schema
RDF	Resources Definition Framework
SP	Shared Plans Theory
Sa	Skill-agent
SDM	Software Development Methodology
SAT	Speech Act Theory
SQL	Structured Query Language
SDLC	System Development Life Cycle
TFP	Team Formation Process
TAS	Travel Agency System
DOD	United States Department of Defence
W3C	World Wide Web

Chapter 1 Introduction

1.1 Background and Motivation

> "*Since the mid 1980s, software agents and multi-agent systems have grown into a very active area of research and also commercial development activity. One of the limiting factors in industry take up of agent technology, however, is the lack of adequate software engineering support, and knowledge in this area*" from; Agent Oriented Software Engineering (AOSE) 11th. International Workshop (2010).

The multi-agent systems (MaS) evolve from Artificial Intelligence (AI), Knowledge-based System (KbS), Object-Oriented System (OOS), and Distributed Artificial System (DAI). The *First Workshop on Foundations of Multi-Agent Systems committee* (Doran et al., 1996b) delivered the multi-agent systems typology which classified MaS into two types "independent" and "cooperative" (close and open respectively). The independent MaS embrace a set of agents linked by predefined (hardwired) protocols which operate in a closed, central control system. Therefore the system does not allow for dynamic activities. This type of MaS appropriates applications defined by fixed, predefined procedures operating in one system boundary. In contrast, the cooperative MaS is a dynamic emergent system, characterised by cooperative behaviour. It has the potential to dynamically (at system runtime) search in an open distributed computational environment then form a team of appropriate agents to achieve the user goal. The agent cooperation feature is the key strength of multi-agent systems over all the other software paradigms (Davidson et al., 2009).

A survey of the existing agent-based and multi-agent systems development methodologies, mainly Gaia, Prometheus, MaSE, PASSI, and Tropos reveals that these methodologies do not address the modelling and techniques for the cooperative multi-agent systems. The survey confirmed that all these methodologies built toward modelling an independent multi-agent system where each followed a particular approach pursuant to particular MaS issues. In addition, although these methodologies established some grounding for further enhancement, all there evaluated as incomplete, designed for specific projects and difficult to deploy (AlHashel et al.,

2009a, Stum and Shehory, 2004, Tveit, 2001, Wood and Deloach, 2001, Yogesh et al., 2008). One of the main obstacles to the large scale take-up of multi-agent systems and agent technology is the lack of adequate software development methodology that can generalize the MaS domains and model the system structure into abstractions (Iglesias et al., 1999, Luck et al., 2004).

"However, agent-oriented methodologies have not received much acceptance in industrial environments, which can be partially explained by drawbacks in current agent-oriented methodologies, mainly in terms of applicability and comprehensiveness" (Gonzalez-Palacios and Luck, 2008).

The multi-agent systems cooperative problem solving techniques is the core requirement of multi-agent systems over all the other software development paradigms and without it, MaS lose their concept strength. Deloach stated "Much of the current research related to agents system has focused on the capabilities and structure of individual agents. However, in order to solve complex problems, these agents must work cooperatively with other agents in a heterogeneous environment. This is the domain of Multi-agent Systems" (Deloach, 1999). This research is motivated by the observation that a gap exists in the multi-agent systems development practice where the existing multi-agent systems development methodologies aim to engineer independent (closed) multi-agent systems only. This limitation impedes the multi-agent system playing its key advantage. This research investigates how to evolve the multi-agent systems from the independent type to the cooperative type. To achieve this objective the research proposes a new multi-agent development methodology (DMMAS) that can be used by the software practitioner to design cooperative multi-agent systems that can function in open, distributed, heterogeneous software environment. DMMAS development cycle follows, software engineering lifecycle starting from an early requirement analysis, system analysis, architecture design, and the detailed design phase. Throughout these phases the DMMAS model cooperative MaS main components, the agent autonomous behaviour, agents team formation process and the goal execution plan. The DMMAS is structured around a cooperative agent architecture (ACM) proposed by this research (AlHashel, 2008). The DMMAS incorporates an ontology approach to represent the agent's functionality

which is an innovative technique arising from this research to represent the agent autonomous behaviour.

The basic notion of DMMAS is to build a system that has an emergent organisational structure consisting of two main levels. At the top level are the professional-agents - responsible for executing a set of goal in particular domain. At the bottom level are the skill-agents - expert in particular skills for the goal achievement. Every goal in the system is supported by a dedicated execution plan to guide the professional-agent in achieving the goal in coordination with the skill-agents team. At every goal the professional-agent will read the goal execution plan recipe then search the ontology schema to find the skill-agents that provide the necessary skills to achieve the goal in hand. Therefore, the professional-agent will send a request to invite these skill-agents (to form a team of agents) then adopt these agents accordingly as they execute their skills within the plan. The system has a dynamic structure in response to the goal. Whenever the goal changes, the system will change the skill-agents team according to the new goal requirements. For every goal, there will be a set of services that are formed dynamically by the team maker. The team maker has the master plan based on a predefined database and a master ontology-based services catalogue. For the ontology implementation the research uses Web ontology tools OWL, RDF, RDF-S, XML, and XML-S to represent each skill-agent's functionality. The DMMAS team maker and agent cooperation concept design is based on "*SharePlan*" cooperation theory proposed by Grosz and Kraus (1996) without the agent's intentional model.

1.2 Why a New Methodology

Based on the previous section background this section deduces that the existing MaS methodologies' design objective is different from DMMAS design objective. As stated in the previous section, existing design methodologies do not address or build MaS dynamic cooperation features. Instead those methodologies are applicable for building independent MaS suitable for applications operating locally and developed under one uniformed, coherent system. For example, Gaia focuses on a predefined fixed organisation structure (Gonzalez-Palacios and Luck, 2008). Prometheus author Padgham (2004) stated that "*Prometheus as described in this book does not address agent teamwork or mobile agent*". MaSE does not support the construction of plan-

based agents and it does not contain any dynamic behaviour or temporal or environment interaction (Badr et al., 2008). PASSI has been driven by work on robotics applications, which may impose constraints on its suitability for other domains (Luck et al., 2004). Tropos is the most complete and tool supported methodology. However, it does not support the team formation process and cooperation or agent dynamic grouping (AlHashel et al., 2009a).

A cooperative multi-agent system possesses unique features such as agent autonomous behaviour, team formation process, dynamic organisational structure and agent adoption strategy. In addition it must be able to operate in an open, distributed computational environment. Combining these features in one application introduces new software design challenges that entail sophisticated techniques because these features are not available within the existing software engineering paradigms. In general, multi-agent systems features have changed the traditional software development methods used in software engineering (Luck et al., 2004).

> "*There is a fundamental mismatch between the concepts used by object-oriented developers (and indeed, by other mainstream software engineering paradigms) and the agent-oriented view. In particular, extant approaches fail to adequately capture an agent's flexible, autonomous problem-solving behaviour, the richness of an agent's interactions, and the complexity of an agent system's organisational structures*" (Wooldridge et al., 2000).

For these reasons, using the object-oriented or other software paradigms are not options and a new customised multi-agent systems development methodology is required. However, the existing methodologies provide a useful grounding to start from. In addition, the existing software practice mainly; foundation for intelligent physical agent (FIPA) standard (2000), UML notations Jacobson et al. (2000), and Agent UML Odell et al. (2003) contributes useful inputs into the DMMAS development process.

1.3 Research Aims and Objectives

Despite a great deal of research in the area of agent and multi-agent systems, a number of research challenges must still be addressed before making agent-based computing a widely accepted paradigm in software engineering practice. The main objective of this thesis is to improve the multi-agent systems development practice to transfer the development process from designing an independent system to a cooperative system. This objective addresses the existing gap in the multi-agent software development practice and improves the MaS development process.

To achieve the research objective the following aims are set:

- Extend the existing multi-agent systems software engineering practice by providing a new multi-agent systems development methodology (DMMAS). The DMMAS work as a development tool for the practitioner to analyse and design cooperative multi-agent systems characterised by autonomous agents, and agents' adoptive concept to forming emergent organisational structures based on professional agents and skill agents.
- Inform the software engineering practice by the experience gained in designing a novel software development proof of concept for:
    - autonomous software entity,
    - emergent system structure,
    - MaS team formation process including ontology based agent selection process.
- Propose a new multi-agent system architecture characterised by agent levels and professional agent and skill agent.
- Improve the software engineering practice by incorporating ontology in software engineering development process.
- This contribution introduces new diagrammatic notations and language for representing the agent-based system behaviour attempting to standardise agent-based system design. In addition, it incorporates the ontology approach in the agent-based system analysis and design phases to construct a robust software agent architecture.

1.4 Research Questions

This research focuses on the multi-agent systems software engineering, specifically the analysis and design processes for building cooperative MaS. The research takes into account the complexity of the MaS internal structure and components in addition to the operating computational environment and its openness. In a research area where some work has been done, the thesis addresses these questions:

1. What are the components and how are they interconnected to form the cooperative multi-agent systems architecture that has emergent organisational structure and embraces agents team formation process?

2. What is the mechanism that has the ability to represent the agent's autonomous behaviour and allow it to function in an open, distributed computational software environment?

3. Ontology paradigm has been used in many software projects without a standard software development methodologies or dedicated graphical diagrams or notations. How and ontology be incorporated into software development methodology?

4. How to engineer a cooperative multi-agent system that embodies: autonomous agent behaviour; team formation process; agent's adaptive strategy; and emergent structure?

1.5 Hypothesis

- The existing multi-agent systems development methodologies are designing independent (closed) systems and fail to design cooperative (open) multi-agent systems characterised by an agent autonomous behaviour with the potential to function in an open, distributed, heterogeneous software environment then dynamically form a team of agents to achieve a common goal.

- This thesis proposes a new development methodology with the potential to analysis and designs the cooperative multi-agent systems.

- The ontology approach is sufficient to represent the agent functionality and it is possible to incorporate the ontology engineering within software engineering using UML class diagram with new additional graphical notation and diagrams to represent the ontology concept, classes, attributes, properties, and relationships.

1.6 Research Methodology

Selecting or agreeing on a particular research methodology is a critical decision that has direct impact on achieving the research objective and aims with the best possible precision. Moreover, the research methodology must be able to address the approach and guidelines that will lead to answering the research question(s) and validate and evaluate the research outcomes and verify the hypothesis. Selecting a research methodology is part of the research design process. In fact this process enables the researcher to understand his/her research in more detail.

Many research methodologies are available and selecting one that matches the research will be arguable because there are differences in opinion over the nature of the truth, and how we can discover that truth using scientific investigation. In normal situations there are more than one approach to achieve the same result and the decision to choose a particular approach is influenced by the philosophical stance adopted by the researcher or research members (Easterbrook et al., 2007). Despite the decision made on method selections, it is often necessary to use a combination of methods to fully understand the problem (Easterbrook et al., 2007). However, there are no standard procedures or rules that can be applied for designing the research, but there are a set of consideration that can be used as guidelines to align the research aims with the research methodologies. The classification of the research question(s), the data collection method, the nature of the problem statement, research subjects, and the available resources are criteria forming important inputs to the research design process (Morse and Field, 1995).
The research plan consists of three main phases as follows:

1. Study the existing agent-based software development methodologies and related software engineering backgrounds including the software development tools.
2. Design a development methodology (artefacts and process) that has the potential to design agent-based systems.
3. Test the development methodology, as proof of concept applying to a real world case study that is a travel agency system (TAS).

The proposed research is classified as qualitative research, and seamlessly runs under design science in an information system research framework, and for validation and testing; the research applies a case study research approach. Figure 1.1 illustrates the research framework using Hevner et.al, (2004) model for an information science framework.

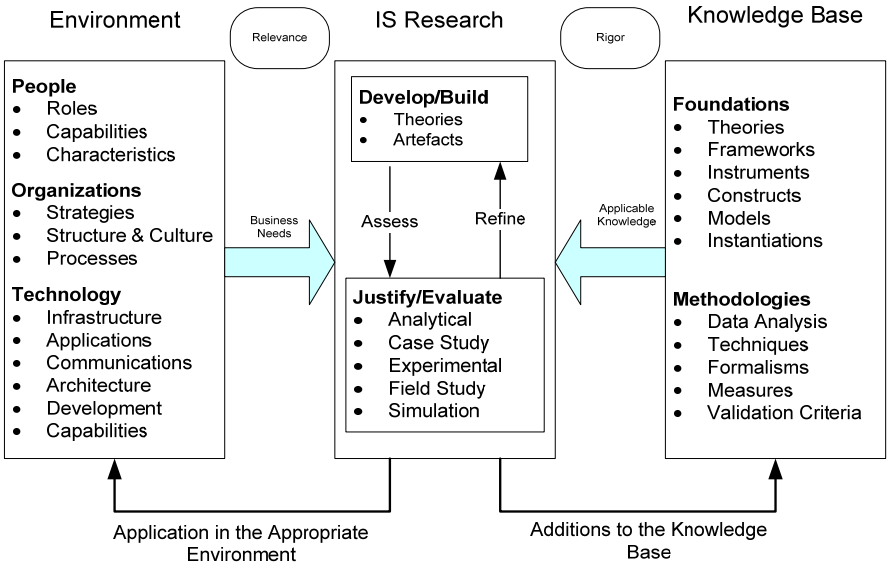

Figure 1.1: Information system research framework (Hevner et al., 2004).

Hevner et.al framework (depicted in Figure 1.1) is based on two complementary phases. Design science addresses research through the building and evaluation of artefacts design to meet the identified requirement, and behaviour science addresses research through the development and justification of identified requirements. The IS research develops and builds theory and artefact that may be under development or undiscovered then justifies or evaluates the products using assessment methodology for example, analytical, case study, experimental, field study, or simulation. There are two sources in the development process business needs or requirements justified by

people, organisation, and technologies. The other resource is the applicable knowledge facilitated by foundations and methodologies of the available knowledge base. Hevner et.al, IS framework defines the artefacts development process as an iterative cycle; develop, evaluate, amend then at the end impact the environment and the knowledge base by the outcome usefulness.

In their discussions of Hevner information system design science framework and Dewayne and Alexander (Dewayne and Alexander, 1992) have addressed seven guidelines for research design to be considered, summarised as:

1. Systematic problem solving which identifies variables and tests relationships between them.
2. Logical, so procedures can be duplicated or understood by others.
3. Empirical, so decisions are based on data collected.
4. Reductive, so it investigates a small sample which can be generalized to a larger population.
5. Replicable, so others may test the findings by repeating it.
6. Design as a search process, so research must be conducted with knowledge of other, competing approaches and should approach the process as a cyclical problem solving process.
7. Communication of the research, presentation of the result needs to address both the rigour requirements of the academic audience and the relevance requirements of the professional audience.

Hevner's IS research framework

Mapping the research plan into the information system design science research methodology framework is an important step to illustrate how the research will be carried out. Figure 1.2 reconfigures Hevner's model for design science in information science to adapt to this research requirements and objective. It also illustrates the proposed research development process, and the research development structure. The people under the environment representation are related to the practitioners who will use the product of the research, but in the development process the people are related to the research developer. The research development process is based on continuous progressive iterative delivery techniques. The DMMAS consist of two main phases,

analysis and design. Each phase is divided into main steps and each main step has a number of sub-steps. Whenever a step has been developed there will be a test to evaluate its applicability and refinement if necessary. There will also be a main testing and evaluation of the phases and main steps which will be carried out through the development of the case study. Furthermore, in case there are any unsatisfactory results the required modification will occur and undergo to the testing process again. This process will continue until the complete product is built. The knowledge base representation is another input into the research development process. It is related to the architecture that the research will model. This architecture works as a design guideline for DMMAS modelling. DMMAS should be able to design the required multi-agent systems around its architecture.

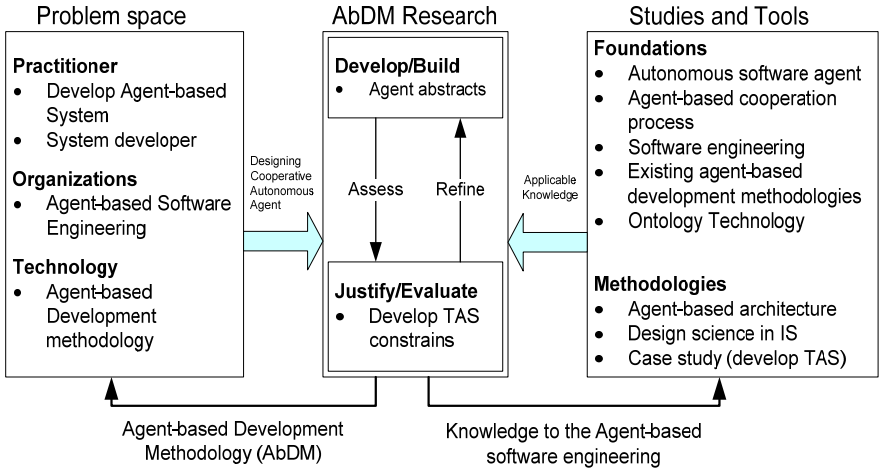

Figure 1.2: Research methodology diagram in context of Hevner's model.

To achieve the aims, the proposed research follows a course of processes summarized as:

1. Investigate the software engineering development techniques; the purpose of this activity is to extend the research understanding of the software building strategy and utilise suitable software concepts and practice in the research. To achieve this objective, the research considers the following software concepts:
   a. Software development methods.
   b. Components based software design.
   c. Formal language specification, Z specification, first predicates calculus.
   d. A model-based approach for software engineering.
   e. Multitasking embedded systems techniques.

2. Study and evaluate the existing, most commonly used agent-based development methodologies to discover their strength and weaknesses. The main agent-based development methodologies that are under the research focus are: Gaia, Tropos, Prometheus, PASSI, and MaSE. The evaluation process for these methodologies applies each method on the TAS case study as a prototype model then uses Onn and Arnon's (2001) evaluation framework to present the results. The research also proposes its own evaluation model to enhance the proposed assessment framework and to test the abilities of these methodologies based on the agent cooperation process.
3. Study the main agent-based cooperation theories: Joint Intention, Share Plans, and Planed Team Activity. In this process the research analyses the theories then identifies the cooperation principle used, the cooperation type, depth, input elements, resources, process, and domain type. Out of this study the research derives the agent-based cooperation abstraction and decides how these abstractions are represented and connected together. The next step conceptual agent cooperation framework is developed, with consideration to the best engineering agent cooperation model.
4. Convert the conceptual agent-based cooperation model of step three to architecture then develop the operation cycle of the architecture. The operation cycle will be used to guide the process of the development methodology building blocks.
5. Develop the structure of the DMMAS development method, and identify both the analysis and the design phases. Start from the analysis phase and build the TAS case study application analysis phase. While the research is processing and generates artefacts (agent abstractions), each generated artefact goes through the test cycle. The research follows Simon's (1996) nature of the design process as a Generate/Test Cycle (Figure 3).

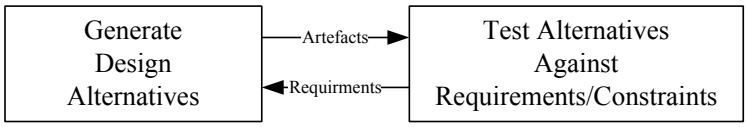

Figure 1.3: The generate/test cycle (Simon, 1996).

6. Until the last artefact is produced and a complete model delivered, the research continues processing in the following loop:
    i. Identify the concept of the proposed agent-based method.
    ii. Identify the methodology (DMMAS) notations.
    iii. Identify the syntax and semantics used.
    iv. Analyse each phase and find the input and output over the entire model.
    v. Identify the method final product.

1.7 Thesis Outline

Chapter 2 surveys the literature of agent-based systems and argues on the paradigm development tools limitation. The second part of the chapter surveys the multi-agent system current state including the development issue, theory, software tools, and agent architectures.

Chapter 3 discusses the problem's background and related works then presents the effort made in the field. The second part of the chapter relies on a practical experiment, a Travel Agency System (TAS) development using three main agent-based development methodologies (Prometheus, MaSE, and Gaia) then reveals their strength and weaknesses.

Chapter 4 works as a user manual for DMMAS deployment. The first part of the chapter proposes a new multi-agent systems architecture and explains its main components and connectivity. In the second part, it presents the new development methodology for multi-agent systems. The chapter explains the processes of DMMAS from the early requirements to the detailed design phase including the graphical diagrams and notations used.

Chapter 5 proposes a proof of concept. This chapter deploys the DMMAS analysis and design processes to develop a real world case study, Travel Agency System (TAS). The chapter discusses the deployment techniques and provide a proof for DMMAS's analysis and design.

Chapter 6 evaluates and assesses DMMAS modelling and techniques. The evaluation follows a benchmarking approach and shows the advantages of DMMAS in comparison with the existing five multi-agent systems development methodologies previously examined, i.e. Gaia, Prometheus, MaSE, PASSI, and Tropos.

Chapter 7 outlines the conclusions of the research. The chapter discusses the findings then presents the research limitations and concludes with recommendations for future enhancements.

Appendix A contains goals XML schema selected code for the TAS case study and "*Achieving PhD*" example. Appendix B contains select code ontology definitions for the TAS case study skill-agent functionality. Appendix C provides an example of a goal execution plan database relationship and standard query language including the results presented in screen image. Appendix D explains AUML implementation including graphical diagrams and notations. Appendix E lists publications produced from this research.

Chapter 2 Agents and Software Engineering: A Review

2.1 Introduction

As has been stated in the previous chapter, an agent is generalized and gains diversity of understanding and definitions. The first part of this chapter establishes a background for agent-oriented software (AOS) concepts and definitions then it explains how AOS differs from object-oriented software. This background is followed by exploring the literature of multi-agent systems theories and practice including software agents cooperation theories. This part also investigates the current state of art in software agent paradigms from a development perspective, including concepts, approaches, tools, and paradigms.

The second part of this chapter focuses on surveying related existing MaS development methodologies and other current software practices. Then it argues their limitations for engineer cooperative (open) MaS. Finally the chapter justifies the research direction and explains the context of the research objective and aim.

2.2 Agents

Agent-oriented software relies on its basic unit behaviour called "*agent*". The term *agent* has many definitions and yet no consensus has been reached on what is meant by the term, '*agent*' (Mullar et al., 1997), (Luck, 1999). An agent is a software unit that possesses autonomous behaviour toward its own services and relationships with other agents. An agent also operates homogeneously with its environment and responds to the changes in that environment (Weyns et al., 2007). This definition also confirms that "agents form a synergy with their operational environment and react to changes in the environment. Agents also exploit the environment to share information and coordinate their behaviour" (Bergenti and Huhns, 2004). However, the most common definition in the software agent community is stated by Wooldridge and Jennings (1995); an agent is software computer program that enjoys the following properties:

- *autonomy:* agents operate without direct intervention of humans or others, and have some kind of control over their actions and internal state,

- *social ability:* agents interact with other agents (and possibly humans) via some kind of control over their actions and internal state,
- *reactivity*: agents perceive their environment (which may be the physical world, a user via a graphical user interface, a collection of other agents, the internet, or perhaps all of these combined) and respond in timely fashion to changes that occur in it,
- *pro-activeness*: agents do not simply act in response to their environment; they are able to exhibit goal-oriented goal-directed behaviour by taking the initiative.

In addition, Michael Genesereth (1994) defines an agent: "*agents are software components that communicate with their peers by exchanging messages in an expressive agent communication language* ". Genesereth's definition views an agent as software components communicating with each other using a standard messaging system independent of their internal data structures. The messages used are in the form of request rather than invocation. However, Prometheus developed a book store system base without standard protocols or semantic language. Odell (2002) also defines agent from the autonomy behaviour perspective that distinguishes agent behaviour from all other software paradigms " *Since a key feature of agents is their autonomy, agents are capable of initiating action independent of any other entity*". This definition is close to the research point of view.

Each of these definitions focuses on particular agency features or properties, and yet none of these definitions qualified the agent that operates cooperatively in open multi-agent systems that design for business application domain. For this type of system environment, the research envisages an agent with autonomous and cooperative behaviour, with unique capability (services); then defines it "*an agent is an independent software system or subsystem that has unique capabilities (skill) and can offer its skill towards achieving a common goal base on an accept or reject request*". This definition is the same understanding stream of both Zambonelli et.al (2003), and Giorgini (2005).

2.2.1 Agent Typology

As explained in the previous section, an agent has several definitions and there are no agreements on one specific definition. This results in a multiplicity of agent types being proposed, including agent sensor agent, interface agent, reactive agent, coordination agent, and the list is too long. The researchers have described agents in a diversity of classification. Nwana (1996) proposed an agent's topology model that governs the key properties of the ideal agent type.

Figure 2.1 depicts three main agent's properties that result in having four types of agents. The first property classified as agent "*autonomous*" and indicates that agent possesses its own thread of control on its action and is capable of mobility across different networks and platforms. The second property is "*cooperative*". This property can be examined when an agent's goal is beyond the capacity of one agent's capability and in this situation; an agent operates in a group, then cooperatively performs the assigned goal. The third property by which an agent can be classified is learning ability properties "*learnable*". This property enables an agent to act in a deliberative manner against its peer agents and it must perceive its environment and react to the changes in this environment. This property entails knowledge repository to develop experience model.

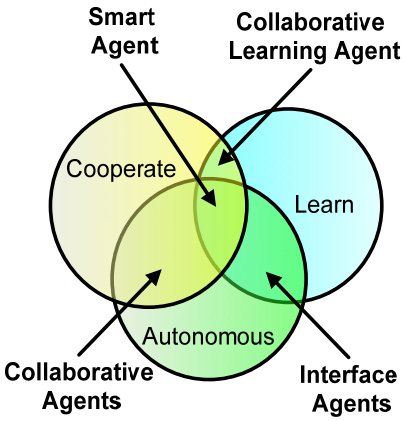

Figure 2.1: Part view of an agent typology (Nwana, 1996).

The three properties Cooperative, Autonomous, and Learnability characterise four types of agent as shown in Figure 2.1; collaborative-agent that combines both

autonomous and cooperative properties, smart-agent that combines autonomous and learning properties. The collaborative-learning-agent is the result of combination between cooperation and learning properties; the smart-agent is the ideal type as it combines all three properties.

The Nwana's agent typology defined the strength of the AOS concept and provided a classification for the agent types. Furthermore, any agent-oriented application must be classified under one of these four categories and demonstrate at least one of these properties. This research targets the collaborative agents which combine both autonomous and cooperation properties where the learnability is unfeasible to DMMAS application domain. Thus this topology is not absolute typology for all agent-oriented software application domains. For example, the deliberative (learnable agent) agent is not necessarily cooperative. However, the research proposes that the Nwana's typology will be more comprehensive if it extended by the communicative properties. This is to accomplish the coordination agent type which totally relies on the agent communication abilities.

2.2.2 Agent versus Object

Software agent is unique in its characteristics, in addition it is a recent concept delivered over the existing software paradigms. In comparison, the closest computational software concept to agent-oriented software is object-oriented software (OOS). In addition, object-oriented software is the ideal example for successful software techniques and technology. This success stimulates many developers to build agents on top of objects technology.

The purpose of this section is;

- to provide evidence for the research direction in building DMMAS as dedicated MaS methodology from scratch,
- to demonstrate the research understanding for the agent-oriented system concept,
- used the differences list to illustrate the weaknesses of the object based existing multi-agent systems development methodologies.

While there are some basic similarities between agent-based software and object-based software, there are fundamental differences between them at the conceptual level, problem domains, internal architectures, implementation techniques, and development approach. These are summarised as:

Conceptual differences: Agent is an independent software entity which reacts autonomously to the changes in its environment without human intervention, to pursue its design objective; in other words, an agent owns its own thread of control over its action. The agent's autonomy property becomes possible because agents interact with each other using standard messaging protocols (ACL, KIF, or KQML) independently from the agent internal data structure. In this interaction, an agent responds to another's request as an accept or reject choice, thus there is uncertainty in their respond to the call. In contrast an object does not own its thread of control; it links with each other's using a predefined signature or call method. In this linking an object operates under invocation condition to perform its design objective. If this condition is not met then an exceptional system error may occur.

Object does not have an open interaction instead it integrated with others and their communication is predefined routine, implemented part of the object internal data structure. Odell (2002) illustrates the agent's autonomy in comparison with object behaviour using graphical example, see Figure 2.2. The autonomy comparison illustrated from activity base on (passive/active) and predictions (predictable/unpredictable) with clock, and ant colony as an example. The graph reflects that object is a passive and predicted entity with no choice of self-control where an agent operate in an unpredicted environment and has its own choice of control that can say (Yes/No) or (Stay/Go).

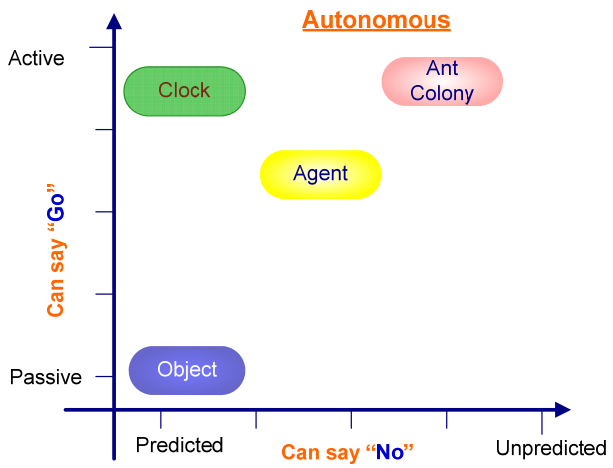

Figure 2.2: Two aspect of autonomy (Odell, 2002).

a. When the goal is beyond one agent's capabilities an agent enters in commitment with other agents to offer its services as part of a goal solution then collectively performs the assigned goal in a cooperative style. The agent's cooperation process can be defined by the ability to dynamically form a team of agents towards achieving a common goal. The agent's cooperation problem solving technique is a power tool that distinguishes a software agent from all the other software paradigms as it allow agents to operate in an open, distributed computational environment. However, software object-oriented relies on hardwired predefined links subsequently create rigged objects relationships.

b. An agent is intelligent, is facilitated by an intelligent capability to support its decision criteria, and then exercises the most appropriate actions that match the tasks given. Software agent is equipped with intelligence to develop and support its past experience repository to develop its own learning course. Most software agent-oriented definitions relate the intelligent behaviour of agent as its superiority over object specification (Rudowsky, 2004), (Odell, 2002), (Wooldridge and Jennings, 1995). However, the research argues that software intelligence is not proscribed on the non-agent software, in fact there are no conceptual reasons not to implement intelligent software object. In the literature, several object-oriented algorithm and programming techniques have been developed to support object intelligence behaviour, for example, Jess is a rule-based software engine developed in Sun's Java to manipulate and reason about

Java objects (Friedman-Hill, 2003). The difference between the two paradigms (agent and object) in the intelligence aspect is that in the software agent the intelligence comes as part of the agent definitions while in software object it remains implicit. Generally, it is all about the system requirements and the nature of the system application domain.

c. An agent is goal oriented; it persists on its goal achievement. The research does not agree on this concept as uniqueness to the agent rather it has been stated in the agent definition. In fact, the software object exerts this concept with no indications to its specification, for example, software object can be set in a loop of time limit or counts to perform its task; in addition each object has its goal or design objective to pursue. Finally, there is no significant difference in this particular concept.

Implementation differences: Software agent required specific runtime environment; it is different from software object runtime. In software object, the compilation and runtime are done within the development programming language resources and are conditional to successful compilation (object code). In contrast, software agents entail runtime services environment. This runtime services must be able to manage the system messages, agent identification locally and globally, services registration, running container, and system manipulation (add/delete) agents. For example, JADE, Cougaar, Aglet, and JACK are all agent runtime environments where each platform is dedicated to a particular agent system approach.

Architecture differences: Software agent allows distributed computing, where software objects are centrally organized, more integrated. In this regard software agents are loosely coupled, adding or removing agent from the system will not break the system runtime.

a. In the software object architecture, you must predefine the object interface in order to support the design and development process. In contrast, in the software agents architecture, the interfaces employ several techniques to identify its interface.

b. The software agent structure is scalable. The system is open and adding or removing agents can be done at system runtime. This scalable structure also embodies the agent's cooperation activity so while the system goal change

accordingly, the team of agents will change. Therefore the system is emergent, open changeable architecture. The architecture of the software object does not exert dynamic object changes, because the object-oriented system is built around predefined components and does not expect adding or removing objects at runtime.

There are fundamental differences between the agent and the object, both pointed to different application domain and designed to tackle different type of problems. While the agent-oriented system focuses on open, distributed, environment and is characterised by dynamic, changeable activities at runtime, the object-oriented system is structured around predefined components to pursue constant system goals and designed to serve in defined system boundary.

2.2.3 Agent Architecture

In the early stages agent architectures were established around two agent properties, reactive behaviour that operates under sensors/response style and deliberative act that reason about their actions such as those based on believe, desire, and intention (BDI) architectures (Langley, 2008). The prospective is to model hybrid combinations of those two architectures and adopt the best of each approach. However, there are four main groups of agent architectures; Belief desire and intention (BDI), Logic-based, Reactive, and Layered architecture (Bellifemine et al., 2007b). Each of these architectures has its own techniques and strategy to structure an intelligent agent-based system as a set of interrelated components. For example both Brook (1991) and Eyal et al. (2004), argue that intelligence can be represented by a set of interrelated decomposed components of a system or intelligence can emerge from systems with complicated components structure.

Belief Desire Intention model: BDI agent is designed around a logical mantel state or deliberative foundation in forms of belief, desire, and intention attitudes then each of these attitudes is represented within the BDI agent implementation. The belief is represented by the information an agent has about its environment, regardless of the accuracy of this information. The desire is represented by the goal or the task allocated, therefore, it represents the objective that agent intended to achieve. Finally,

the intentions specify a course of actions the agent performs to pursue its goal. BDI agent architectures have been developed and implemented in many agent applications domain. One of the most similar models to the BDI concepts is implemented by Georgeff and Lansky (1987). The model is also supported by a number of implementation and runtime software platforms for example, JACK, JADEX, Cougger.

Figure 2.3 depicts the elements of BDI agent architecture, the interpreter is the logical mechanism that runs the entire model and provides interaction with the environment including the external systems, and updates the model domain knowledge that is the belief (database). The belief is in dynamic changes parallel to the changes in the environment, subsequently, the course of actions (desire) change with the new belief these techniques provide the plan in-time factor as happens in the real world. According to Georgeff and Ingrand (1989) "*In real-world domains, information about how best to achieve some given goal can often only be acquired after executing some initial part of that plan"*.

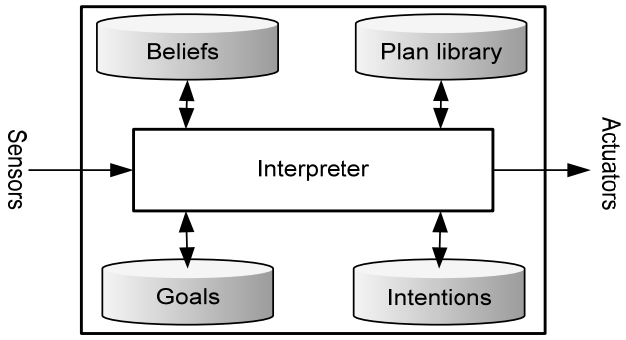

Figure 2.3: The BDI architecture.

Layer Architectures: In the Layer Architecture the system is partitioned into multi layers hierarchal structure and each layer represents a unit of logical behaviour. There are two types of layer architectures, horizontal and vertical, each with its own approach and internal interactions techniques. In the horizontal architecture (see Figure 2.4), the layer is set in sequential hierarchy (linear) style and the input sensor directly connected to each layer. The sensor passes input to and receives the output from each layer; that is each layer is treated as individual agent. If the layer that

received the input is not related to that input request then it does not respond and this is translated as irrelevant to the layer behaviour.

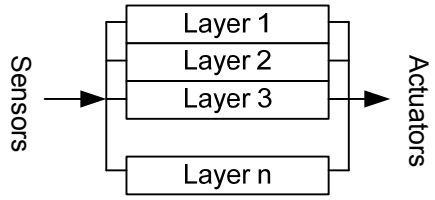

Figure 2.4: Horizontal layer.

The vertical layer architecture (Figure 2.5), is the advanced version over the horizontal model its main approach relay on reducing the interaction overhead in addition, it is set to be a fault tolerance. The layers are organized in a hierarchal structure similar to the horizontal model but the sensor pass is subdivided into two techniques, one-pass and two-pass (see Figure 2.5). In one-pass the sensor pass input sequentially starts from top to bottom (vertically), and every layer performs its related role and the last layer performs the output action. For the two-pass technique the sensor input passes first top layer, to the bottom layer then bottom to top and receives the output action from the top layer, in other words the model is treated as one agent.

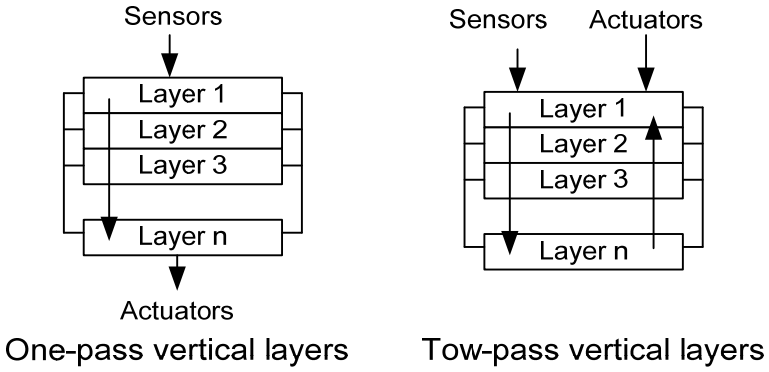

Figure 2.5: One-pass and two-pass layer.

Logical Architecture: The idea of the logical architecture is knowledge-based system techniques based on the ability to perceive the environment (real world) and respond according to the changes in the environment. The environment is the main reason to take an action about the task given in hand. The environment in the logical

architecture is represented symbolically, for example, a set of first-order logic. The logical architecture equipped by a logical reasoning mechanism that runs over a sequence of processes to achieve one full reasoning cycle then generate the output action or decision. This set of reasoning is kept for the system developer to design. However, it is difficult to translate the real world accurately into a symbolic representation and it is time consuming.

Reactive Architecture: In decision making architecture consists of sequential integrated logical layers each with its own functionality and able to interact with each other in hierarchal method. The Reactive Architecture is developed by deconstructing the system into tasks or behaviours then restructuring those tasks into hierarchal top to bottom integrated logical layers (units). Each layer receives its input from the previous layer then performs its logical unit cycle, then generates output to the next layer, the operation continues until the bottom layer produces the action and the loop continues. A well defined reactive architecture system has been developed by Rodney Brooks (1986), the system used to control a robot that wanders the office areas laboratory. Figure 2.6 illustrates a set of sequential tasks that modelled Brooks' robot control system.

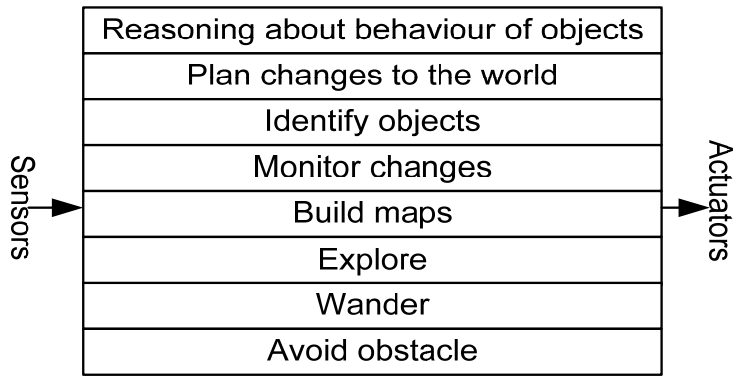

Figure 2.6: Subsumption architecture for robot navigation (Rodney, 1986).

The research focuses on open multi-agent systems that rely on agents' cooperation concepts. However, the agent cooperation features are not designed or considered in any of the available agent architecture, while agent cooperation is the key advantage of the software agent paradigm. This shortcoming could be related to the view that agent cooperation is emerging of agent behaviour at the runtime managed by the

runtime platform, or these architectures are built toward robotics and particular software projects and are not generalised. Even though the architecture must demonstrate either the independence of agents, or accept or reject components as part of the agent architecture. In fact, there is no behaviour without built-in components to support it. The other argument is that the agent standard body (FIPA) agent architecture is also not generalized; therefore it is not compatible with all application domains. Overall, the agent architectures are task based, some specific tasks need a specific architecture, therefore, the research deals with the service agent as independent from any architecture. This is to generalize the DMMAS techniques.

2.3 Multi-agent Systems

The multi-agent systems is a modern approach that has emerged from distributed artificial intelligence (DAI). The Cooperative Distributed Problem-Solver (CDPS) techniques used in DAI have been replaced by an autonomous agent and a dynamic team formation process to create an automated resilience system that can reconfigure itself according to the changes in the environment, or the user's goals. Sycara (1998), describes MaS as an agent system with the following characteristics:(1) each agent has incomplete information or capabilities for solving the problem and, thus, has a limited viewpoint; (2) there is no global system control; (3) data are decentralised; and computation is asynchronous.

Given that agents are autonomous in their behaviour, the relationship between agents or agent-to-agent interaction also becomes autonomous. This creates a social relationship between agents. Social behaviour recognizes that agents interact in a request form (accept or reject) to take on a problem solving role. In other words, agents achieve a common goal through a cooperative method (Li Changhong, 2002). Solving a common problem using cooperative techniques is the key element of the multi-agent system approach. In order to establish the cooperation process, a dynamic team formation process is essential. Without a dynamic team formation process, the multi-agent system will lose its fundamental value (cooperation), then will result in poorly designed systems (Doran et al., 1996b).

The cooperation techniques in the multi-agent systems can be seen as the potential to dynamically form a team of agents for achieving a common goal. The potential of dynamically forming a team of agents is an internal cooperation structure that sustains the idea of designing a collaborative system. The team formation process has been designed by adopting different approaches, including agent motivation, execution plan, organization structure, built-in goal (Sycara and Sukthankar, 2006), (Barbara, 1996), (Kinny et al., 1992), (Fan and Yen, 2004).

This research developed from the intensive literature of multi-agent systems types and building approaches, and a particular interest to research a specific foundation to the cooperative multi-agent systems structure and behaviour. Based on this objective the research studies the multi-agent systems from cooperation perspective.

Multi-agent Systems Types: Multi-agent systems are categorised by the levels of cooperation that take place between the agents of one system. Cooperation is the central behaviour of multi-agent systems and the type of cooperation determines the type of multi-agent system. Figure 2.7 shows the multi-agent system typology based on the panel discussion at the *First Workshop on Foundations of Multi-Agent Systems committee* (Doran et al., 1996a).

Multi-agent systems are classified as independent, if the system does not include dynamic cooperation between their agents. Independent systems are specified when their agents link together in predefined mechanisms and the system does not behave dynamically. The independent MaS classified as discreet, and emergent. The discreet MaS does not contain any cooperation, for example the book store system (BDI agent approach) in Prometheus. The emergent MaS is characterised by predefined constant cooperation instructions and does not exercise any dynamic behaviour in addition, their agents having their own objective isolated from each others, for example, the organisational structure of Gaia development methodology.

The other branch of MaS is cooperative, which is characterised by distributed autonomous agents linked to gather dynamically at runtime and act in cooperative modes using particular coordination mechanisms. The cooperative MaS is classified

into two classes, communicative, and non-communicative. The communicative MaS can be deliberative, if it follows a plan and action strategy or it can be negotiative if it uses a negotiation strategy. The non-communicative MaS is cooperative based on observing and reacting to the other agents' behaviours without any messages or protocol, for example, Observes and Act (OA) technique, and Brook's robotic model.

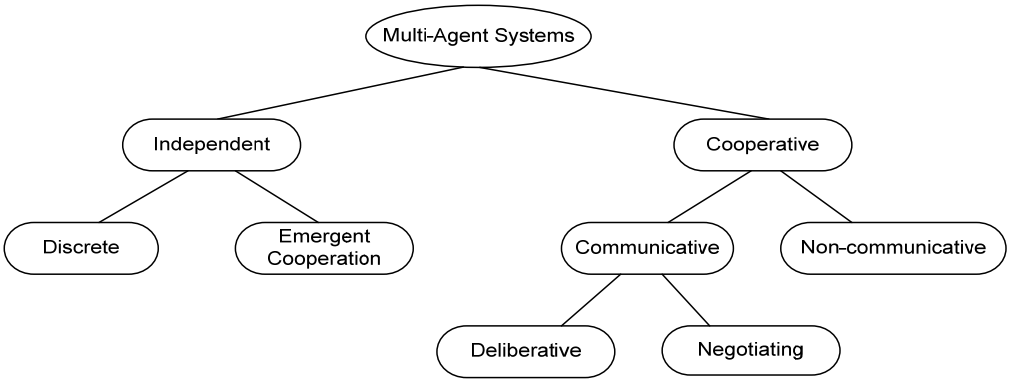

Figure 2.7: Cooperation typology (Doran et al., 1996b).

The research relies on this typology and intends to transfer the multi-agent systems from the independent type to the cooperative type. The research's main argument is that the existing multi-agent systems are designed independent MaS using a particular design approach. However, the research focuses on the communicative and specifically on the plan and actions type. This type of multi-agent systems can be derived with less communication and messages exchange and relies more on the plan and actions strategy. As the study goes further, it will clarify that the plan and action strategy can be developed based on the SharePlan agent cooperation theory for Grosz and Sidner (1990).

2.4 Agent Cooperation Concept and Theories

There have been many attempts to formulate multi-agent social behaviour "*collaboration*". Those attempts result in different documentation, theories, research reports, and conference papers, all centred on the agents team formation process for achieving a common goal (Tambe, 1997), (Bulka et al., 2007b). The team formation process is designed according to different approaches that rely on the motivation or initiation of collaboration between agents. For example, Joint Intention introduced

shared beliefs. Another theme, Shared Plan, is based on sharing the execution plan, Planned Team Activity, is based on individual BDI and predefined plans within agent internal states. Wooldridge and Dunne (2006), also present a model based on the desire to achieve one of a set of goals. This set of goals is linked with the coalition choice which leads to corresponding cooperation. There are also some attempts which rely totally on the capability of agent interactions using standard communication protocols. For example, Vieira et al. (2007), have developed knowledge based semantics to be incorporated into agents programming language known as AgentSpeak (Rao, 1996). These semantics expand AgentSpeak logic to recognise agent communication messages and transform them into knowledge and subsequent action based on this knowledge.

This research studies the following agent's cooperation theories; Joint Intention, Shared Plan, and Planned Team Activity. The research focuses on these theories for their originality and acceptance by the multi-agent community (Wilsker, 1996).

Joint Intention (JI): proposed by Levesque et al. (1990), and extended by Cohen and Levesque, (1991). Joint intention theory for agents' team formation process is designed base on complicated mathematical model that works as agents' intentional mental attitude toward agents team commitment for solving a problem in particular domain. Each agent is equipped with one mental model that also shares with the team, a common mental model with a particular mathematical value to show its belief about goal achievement. The mental model will guide each agent to observe the other agent commitment toward the goal achievement belief. Subsequently it sets its mental state and its belief in the team common model. Therefore each agent acts intentionally based on their belief about the goal achievement then advertises their commitment in the team common model.

The commitments are motivated when the agent possesses a shared belief with other agent team members about the common goal. But to enable an agent to create its team, it must advertise its own goal belief and share its private goal with the other agents and agree to communicate, regardless of its implicit belief. The agent with the goal has the responsibility to make sharing agent belief and agent private goal mutual. This responsibility will create a joint commitment between agents to achievement of

the common goal. An agent cannot arbitrarily drop a goal (including the achievement plan), unless the agent develops a new private belief that the goal has been achieved, is unachievable, or is irrelevant.

Joint Intention theory is coherent when agent behaviour occurs in a rational and deliberative manner (Hoek et al., 2007). This agent behaviour introduces complex agent design and implementation. In addition, it is more appropriate for a deliberative type of cooperation.

Shared Plans (SP) (Grosz and Sidner, 1990): The share plan written as "SharePlan" theory extends the Pollack mental state model of plans (Pollack, 1986, Pollack, 1990). Shared Plan theory is based on building different level plans to meet different levels of action. The suitable plan will be allocated to the agent team depending on the problem solving strategy; whether the actions will be executed in an individual or in a group style; and whether the plan will be implemented partially or fully. The problem solving strategy will be based on the extent to which a particular combination of plans is appropriate to the situation. There are two plan types for each level; on the individual levels there are full individual plans (FIPA), and partial individual plans (PIP) and for the group level, there are full shared plans (FSP), and partial shared plans (PSP).

The plan considers the individual or group intention, and mutual belief as the initial motivation to the collaborative action. A group of agents must have a collaborative action plan that includes the following:

a. Mutual belief in a (partial) recipe.
b. Individual intentions towards the action.
c. Individual intentions towards the success of the collaborative sub-actions
d. Individual or collaborative plans for the sub-actions.

Individual actions: When a common goal needs to be achieved by a group of agents, but executed by individual agents each having its own action, each agent needs to use its own plan, provided that there are no conflicts between these plans. For example, two agents might need to use the same resources at the same time. To avoid conflicts, agents must know each other's actions towards the common goal.

Group actions: When a common goal needs to be achieved by a group of agents, but executed by a sub-group of agents, each sub-group having its own actions, the plan of group activities comprises firstly the plan of individual agents and secondly, the plans of the groups. As a result, the sub-action will be decided by the group or sub-group. This means there is a shared plan between groups of agents to perform one action (collaborative plan for the action).

In both individual and group plans, the execution can be achieved in two ways:

a. Partial plan. In this case agents can select the action of interest without starting it immediately. The next step is to select the method of achieving it. This can be done by accessing a "how to do it" source, or by asking another agent.
b. Full plan. Agents completely determine the way in which they will perform an action.

Shared Plan (SP) theory closes the gap in the Joint Intention considering the environmental dynamic changes while the agent is the executing the actions. In addition, SP provides full plan concepts to enable the collaborative agent to take action in case of failure. The theory constructs a deliberative type of collaboration which requires rational agent behaviour, for example, RESTINA software framework. SP needs agents that possess intention and belief at an individual level, a group level and a goal level. The approach is of a domain-nature and requires high level systems design effort.

Planned Team Activity (Kinny et al., 1992): PTA introduces a simple effective collaboration approach which rests on team formation. The process of team formation initiated by the agent intends to achieve a goal but in the case that the goal is larger than its capacity, this agent will assemble a team, installing itself as a team leader. The team leader and team leader communication with potential team members is a key factor in the PTA approach. The team members will be supplied by the goal execution plans prior to their joint decision. In this case agent behaviour is predictable, and can respond immediately in a dynamic environment.

Planned team activity encompasses a set of agents cooperating with each other. Those agents presume that they possess a belief, a goal, a plan, and an intention on an individual level. On the team level also, agents must possess: (a) mutual beliefs about the operational environment and about each others actions; (b) common goals that need to be executed; (c) a team member joint plan that is capable of achieving the common goal; (d) joint plans (committed or joint intentions), accepted as a response to the common goal or an external event (Wilsker, 1996).

The PTA team formation process starts when an agent "A" decides to achieve a goal but the goal is exceeding its capabilities. In this situation the goal becomes the joint goal and agent "A" becomes a team leader and takes responsibilities for goal achievement. Agent "A" will announce the joint goal, the joint plan and the individual roles to the potential team members. The team members adopt the plans if and only if:

- the team has the necessary skills between them,
- the team does not already have a joint goal,
- the preconditions of the plan are already believed by the team,
- the joint goal and the current goals of the team members are compatible,
- the joint plan and the role plan are compatible with the team member intentions.

There are two team formation strategies within PTA. The first is the Commit and Cancel (C&C). The second is Agree and Execute (A&E). Both formations share the fact that the joint goal, the joint plan, and individual beliefs are known prior to the agents' commitment and subsequent team formation.

Commit and Cancel: The team leader sends a request to ask each participant to "commit" to the plan package. If the reply says commit, then the execution begins immediately. If any one of the participants does not commit within a particular time limit, then the team leader sends a "Cancel" message to all the participants.

Agree and Execute: The team leader will send a message requesting all the participants "to agree". If all reply affirmatively, then he explicit requests all the team members to execute the plan, otherwise the commitment process will be cancelled automatically without the need to confirm.

The planned team activity model is more compatible with BDI agents (Kinny et al., 1996). This approach has less implementation complexity and requires fewer communication resources. The PTA is designed to function in unpredictable circumstances. At the same time it is assumed that the execution plan and the joint team plan are predefined by the agent designer prior to the system implementation. This creates reliance on the design process to construct a comprehensive prediction plan tree.

2.5 Other Collaboration Models

There are many other cooperation models and approaches which attempt to formalise collaboration, based on different definitions of cooperation. Cooperation is the end product of collaboration that characterises multi-agent systems.

Matsubayshi and Tokora (1993), for instance, discuss a contracting type of collaboration when one agent delegates its responsibilities (agents goal exchange) to another agent. This situation arises from overlapping between agents' team goal. The advantage of overlapping delegation is a reduction in overall execution cost.

Tidhar (1992), defined the union between the agents' task in achieving a goal as a team work process towards achieving that goal jointly. The team will be formed by identifying the intersection between the agents' plans. The plans library is predefined and built in within the agent design in accordance with the goal or sub-goal. Each agent will share its plan for one given goal or part of a goal (sub-goals). The intersection between those plans then represents the team plan.

Castelfranchi (1995), relates agent collaboration to two principles, firstly the notion of commitment, including agent mantel state, and secondly agent relationships. Commitment comes in three types: internal (I-commitment), social (S-commitment), and collective (C-commitment). I-commitment is what individual agents intend in regard to an action. S-commitment occurs when agent α has intention to adopt the agent β goal, in this case, a social relationship is created between the two agents. C-

commitment emerges from the development of S-commitment but at the level of a group possessing internal intention.

Cavedon Tidhar (1995), concept is to generalise Belief, Desire, Intention (BDI) architecture of individual agents on the team member level using top-down or bottom-up (member to team) approach. First, the BDI will be selected from the library and then this will be used as team guidance. This team formation technique does not provide joint intention so the members are not aware of a team sharing attitude.

All three theories have contributed to the agent cooperation process following different motivation approaches and methodology. The study shows that the Joint Intention theory is focused on the agent mantel attitude and it assumes that agents deliberatively or intentionally share with others to offer it services. However the theory is difficult to implement and entails overhead effort, in addition it is appropriate for intelligent application domains. The plan team activity required agents to build mutual beliefs and an active plan then develop a commitment but the theory does not explain the belief structure over the entire system and does not provide the type of the agent intelligent model. Generally the theory does contribute to research objective. The other approaches are incomplete and difficult to understand. The share plan is applicable for implementation but without the active or dynamic rebuilt processes. The research has used the basic concept of the SharePlan to support DMMAS execution plan model.

2.6 Team Formation Process

The reason behind incorporating the dynamic team formation process within MaS is to equip the system with a mechanism to solve a distributed problem or a problem that has networking characteristics. The team formation process can be explained by the ability of the system and dynamically (at runtime) form a team of agents to achieve a common goal. The American Heritage Dictionary defines teamwork as *cooperative effort by the member of a team to achieve a common goal*.

The agent team formation process originated and discussed in the distributed artificial intelligence (DAI), then diverted to become multi-agent systems. However, designing

TFP opens a new AI research stream and results in many different proposals on how to form a team of agents. The cooperation theory mentioned in the previous section (agent cooperation theories) is an example of those attempts of which most focus on the mantel attitudes as a key element to forming a team of agent. There are also other attempts focusing directly on the process of making the team itself; Durfee (1989), explained the cooperative distributed problem solving strategy (CDPS), and explained the process's main elements. Hartonas (2003), deployed formal language to develop agent plan (behaviour) then proposed agent signature to provide information on the agent's skills, capabilities and domain ontology to establish an agent team. In the same stream Tuomela (2001) proposed the *collective intentionality* framework which derived from social action philosophy. The studies show how a group belief plays a main role to accommodate the team attitude, furthermore, he propose the use of the belief-desire-intention conceptual model at a group level will lead to a mutual belief, subsequently providing shared attitude calling it "we-attitude".

A coordination based approach to model agent communities is suggested by Jennings (1996). Jennings argues that commitments (pledges to undertake a specific course of action) and conventions (means of monitoring commitment in changing circumstances) are the foundations of coordination in all distributed artificial intelligent systems. Another approach for building FTP suggested by Matthew and Marie (2005), emphasises agent networking structure called "agent-organized network" (AON). The approach focused on the explicit interactions, or interdependence, between the agents in the system, then designed local network adoption strategies for individual agents that give rise to dynamic, adoptable organizational structures. Additional philosophers visions of team formation processes are described by (Cohen et al., 1997b). The vision links both theories, the speech act and the joint actions, then focuses on the reaction of illocutionary acts, for example, requests, promises then shows how these subjects can affect the process of the team in terms of establishing and monitoring. In this vision Cohen et al. mentioned that the joint actions are different from the collective individual actions, even if there is coordination.

Analysing, designing, and building ideal team formation processes embrace a high level of complexity. For example agent commitment, conflict resolution,

communication, coordination, cooperation, are not yet approachable in the field of agent-based software development. Furthermore, the software engineering techniques have not addressed any solution to such constraints at the level of the existing agent-based development methodologies; FTP is ignored and not considered in any form of presentation. For example, GaiaROADMAP and MaSE do not design for agent cooperation (Alhashel et al., 2007). On examining Prometheus, its methodology does not address the agent teamwork at all; "*There are a number of types of systems that it does not yet handle well. For example, Prometheus as described in this book does not address agent teamwork or mobile agents.*" (Padgham and Michael, 2004).

However, the team formation process is the main process in multi-agent systems concepts. The research implements the concept of TFP in the DMMAS structure with the following considerations:

- In the design considerations, the research focuses on designing autonomous agent. For this purpose the research translates TFP into the use of ontology approach in the development process of DMMAS.
- The research purposely avoids the complication in DMMAS user manual and tries to provide an affordable user friendly development methodology that can be used and understood by the practitioner.
- The research views the TFP from the information system not from other computational aspects, for example robotics or simulations.

2.7 Agent Negotiation

The concept of agent-oriented software applies to broad types of problem domains including those applications needing coordination between agents to achieve the system objective, or software applications that represent humans (working in the users behalf). For example, networking problem solving, bargaining between seller and buyer over the web, auctioning in e-commerce. In such as environment agents need to negotiate between themselves or between agent and human agents. Negotiation is a stage in the coordination process between parties to reach agreements on the subject of interest. According to the Oxford Dictionary of English "negotiation" definition is "*discussion aimed at reaching an agreement*". There are also various informal definitions for negotiation, from software agents perspective, for example that stated

by Bussmann and Müller (1992) "…*negotiation is the communication process of a group of agents in order to reach a mutually accepted agreement on some matter*".

To enable software agents to negotiate automatically, protocols and strategies must be predefined to them (Sierra et al., 1998). Protocols define the negotiation rules for example, valid negotiation conditions, the participants, language used. The strategies define the decisions under particular conditions, for example, when to bid and when to accept or reject. In general, negotiation is divided into two types: competitive and cooperative, where each employs a particular course of negotiation status whether self-beneficially (maximise own utility) or share-beneficially (maximise group utility) (Lawley et al., 2004).

For the implementation of software agent negotiation, several frameworks, approaches and theories have been proposed, where each model has its own techniques designed to operate in a particular application domain. Thus there is no single common strategy for all possible types of negotiation, resulting in different objectives for particular negotiation models. A survey on negotiation can be found in Bichler (2000) and Jennings et al. (2004). However, in the theoretical techniques, negotiations are classified in three broad categories:

- Game theory-based negotiation.
- Plan-based negotiation.
- Human-inspired and miscellaneous AI-base negotiation approaches.

1. **Game theory-based negotiation**: Game theory is a branch of applied mathematics concerned with studying the participants behaviours then analysing the situation to predict or reason about actions; thus the theory can be applied to negotiation. Game theory assumes a win/lose scenario. Solomon applied the game theory to develop agent negotiation to model a coordination between computer systems (Jeffrey Solomon, 1986).

The game theory-based negotiation embraces a set of concepts they are; *utility* functions, a space of *deals*, strategies**,** and negotiation *protocols*. The utility is defined as the difference between worth of achieving a goal and price paid in

achieving it. Deals are the actions an agent can take which has an attached utility and protocols specifying the kind of deals and the sequence of offers that one allowed in the negotiation course. The negotiation process proceeds as follows; there will be a utility value for each outcome of some interaction for each agent built into a pay-off matrix, which is common knowledge to both parties involved in the negotiation. The negotiation process involves an interactive process of offers and counter-offers in which each agent chooses a deal which maximise its expected utility value. There is an assumption in this technique that each agent in the negotiation:

- is an expected utility maxi miser
- evaluates the other's offer in terms of its own negotiation strategy.

2. **The plan-based negotiation:** Mark et al. (1989), consider an assumption that agents need information from others to function effectively and efficiently, furthermore, planning and negotiation are tightly related, so they propose that agents are provided with planning knowledge. Kreifelt and Martial (1990) proposed negotiation strategy based on two steps. Their first step, agents plan their activities separately, then in the second step, coordinate their plans. The master coordination plans then organized by one dedicated coordination agent, this role also can be performed by any agent. The negotiation will be developed using a protocol based on agents' states, message types, and conversation rules between agents.

3. **Human-inspired and miscellaneous AI-based negotiation approach:** The human negotiation strategies inspire most of the agent-based software negotiation approaches. Sycara (1989) developed a negotiation framework based on a concept that human negotiation draws from past negotiation experiences to guide present and future ones. In the absence of past cases, a preference analysis based on multi-attribute theory will be implemented. Sycara also strongly recommended that to enable agents to negotiate, they must; 1) represent and maintain belief model, 2) reason about other agents' beliefs, 3) influence other agents' intentions and beliefs. Sycara developed a negotiation system called PERSUADER to resolve conflicts between two practising negotiator. Agents can use PERSUADER to modify others' agents Belief, Intention then behaviour.

There are other approaches in modelling negotiation and negotiation protocols, for example, Kuwabara and Lesser (1988) implemented the multistage negotiation strategy. Alessio R. et al. (2001), proposed schema for negotiation in e-commerce, and Iyard et al. (2003) proposed a negotiation conceptual framework based on agent-arguments developed by agents motivations to increase the likelihood and quality of an agreement exchanging to influence each other's states. Designer Fabricator Interpreter (DFI) developed by Werkman (1990), is also a knowledge-based model using incremental forms. The DFI is designed to behave in more deliberative manner using a close approach to human negotiation concept. The model is based on a shared-knowledge representation strategy through shareable agent perspectives which are used by the agent for a negotiation course to share a common domain background. The DFI relay on the blackboard techniques supported by speech-act theory communication protocols, with partitions for the proposals (request, reject, accepts) in addition to the communication and share knowledge partitions.

This section demonstrates and discusses different agent negotiation techniques and modelling in the literature for the purpose of understanding the topics related to software agent engineering and subsequently to develop a background then justify why agent negotiations and not considered in the proposed methodology as presented in chapter 4.

2.8 Agent Coordination

To enable a group of agents working together (cooperate) toward achieving a common goal, or sharing the available resources, they must be able to coordinate between themselves in order to prevent any conflicts. Considering an agent operates autonomously in open distributed environments and acts in a cooperative manner, coordination is major activity. In practice, most of the coordination approaches are built around negotiation as a type of coordination, but identifying the border between the two processes requires some definitions. Holt, informally defines coordination as "*a kind of dynamic glue that binds together into a larger meaningful whole*" (Holt, 1988). Malone and Crowston (1990), focus on an operational perspective: "*The act of managing interdependencies between activities performed to achieve a goal*".

For the purpose of this research, coordination is understood as a process to allow a set of agents to achieve the assigned goal following a goal execution plan. For the DMMAS analysis and design, the research follows a predefined coordination strategy to avoid further complexity. Considering the technical overhead, predefined plan based strategy is a practical technique. In contrast the dynamic coordination entailed sophisticated design and implementation tools. However, many researcher dictate that coordination is constituted by communication and negotiation. This is not an absolute assumption as there is argument that coordination may not need communication provided that agents possess each others' behaviour models, whereby the coordination could be achieved by organization hierarchy (Nwana et al., 1996). Isik et al. (2007), applied implicit coordination model as a type of coordination that does not need negotiation, instead it rely on knowledge of the states of the others agent (beliefs and desire); "*Once the beliefs and desires of the cooperating party are known, we simply image what we would do in that situation*".

However, there are several coordination models developed for real-world software problems domain, each having its plan and each has its advantages and disadvantages. For specific details investigate Jennings (1996), Rabinovich et al. (2008), Jennings et al. (2000), and Faratin et al. (2000). Generally, and according to Nwana et al. (1996) the available software agent-based coordination approaches are classified in the following four types of categories:

1. Organisational structuring
2. Contract
3. Multi-agent planning
4. Negotiation

To broaden our understanding further, and to identify the type of coordination incorporated in the proposed research, we further study the nature of application problems that entail the coordination concept. These include:

1. An agent exercises its behaviour in a decentralised fashion and has a relationship with the other agency in the environment. This relationship needs to be coordinated to avoid conflicts.

2. Meeting global constraints – agent works within a given specification (resources) in this context it needs to stay within those resources.
3. Distributed experiences – agent needs to gather multiple expertise to satisfy particular tasks for example, agent may need to coordinate between doctors, surgeons, ambulance, and nurses to work in response to accident.
4. Dependencies between agents' actions – to enable an agent α to achieve its goal it may require agent β to finish its task prior then agent α initiate. To accomplish this type of procedure agents need to co-ordinate order to succeed.
5. Efficiency – exchanging the information discourse of one agent can be of sufficient use to another agent that both agents can solve the problem faster.

Aligning the five software application cases with the research application problem domain indicates that the coordination process is more applicable to software that is different from the traditional information system or business application system. Coordination is more involved in the system that has an agent with independent actions for example, robotics, game, simulations and networking.

However, for the purpose of the research the proposed methodology modelling introduced in Chapter 4 focuses on pre plan coordination to avoid complicated modelling that entailed syntax and semantics for the communication or mathematical modelling. In addition coordination in the current status developed as ad hoc or stand-alone architecture for example, Jonker and Treur (2001), and Jen-Hsiang et al. (2002) coordination models. Generally, coordination modelling as a standalone concept has not been incorporated into the existing software engineering methodology (refer to Chapter 3), instead existing development methodologies designed agent coordination in a form of interaction using UML sequence diagram similar to object-oriented software engineering.

2.9 Ontologies

This research plans to utilise the ontology technology to represent the agent functionality description as enhancement over the conventional agent name keyword search. This approach is the key concept in the proposed development methodology, furthermore it is an innovative direction toward the agent autonomous behaviour

modelling. The research plans to use Ontologies to define agents' functionality following a schema designed for this purpose then use search code to identify the agents based on their functionality that appropriated to the goal. In this way the agents are fully autonomous and known to the system through its functionalities representation only.

Ontology has been used in the software agent web-based agent search for example JADE platform, Tropos development tools AgentTool. In 1991, the DARPA Knowledge Sharing Effort project proposed three main components to build intelligent knowledge-based systems, i.e. declarative knowledge (ontology), problem-solving techniques, and reasoning services. Accordingly, a large number of projects have been tried to model computer understanding Ontologies (Gomez-Perez et al., 2004). However, Ontology in software engineering is attempts to define the objects through their properties and relationships with other objects for creating a logical meaning to these objects. The vision in using ontology in this context is to transfer the existing information from human interpretation (understanding) to machine (computer understanding) for example the semantic web project: "*The semantic web promises to make information understandable to computer*" (Keller et al., 2004).

However, agent-based software is characterised by its agent autonomous behaviour. To enable a software agent to exercise its autonomous behaviour the agent must be designed with the ability to reason about its action, establish a mutual understanding, and understand its environment. Furthermore, the agent-based system is associated with the semantic web where ontology is its main foundation (Blake and Gomaa, 2005, Kim, 2006, Klusch, 2008, Nyunt and Thein, 2005). Therefore using the ontology software approach to design agent-based system autonomous concepts provides advantage to this research outcome.

2.10 Ontologies in Software Engineering

This section focuses on Ontology from a tools and implementation perspective; in fact, the challenges are located in the Ontology development technologies. Currently, there are many research activities in the field of AI aiming to transfer the Ontology from theories to computer software implementations (Teller et al., 2007).

The modern trend in software is to construct a standard logic-ware reusable software component that possesses a reason properties into its processing (Gruber, 1993). This trend aims to create a global knowledge-based application domain that has capabilities to integrate the Internet entire information infrastructure and make it understandable by the existing computer programs. The primary aim is to changing the existing information from human readable (text-based) to computer understandable (knowledge-based), indeed this is the one of the agent-based paradigm application areas.

For the reasons stated in the previous section and for enhanced implementation of software agent systems a development methodology that can engineer Ontology as a part of the required agent-based system is a prerequisite. Several attempts investigated Ontology design methods and several approaches proposed, for example, Eric and Galina (2009) top-down and bottom-up approached. Formal methods have been proposed by many researchers and for comprehensive survey refer to Matteo and Roberta (2005). Furthermore, comparisons between various ontology approaches can be found in Gómez-Pérez and Manzano-Macho (2003). There are also some attempts to utilise the existing software development methodologies in designing Ontology. Exploits UML class diagrams for representing the ontology relationships between concepts and their attributes, but for axiom an Object Constraint Language (OCL) is used and attached to the concept notes since UML lacks formal semantics (Gomez-Perez et al., 2004, Siricharoen, 2007). However, UML does not own the full potential to model the Ontologies domain model and for this reason it requires further enhancement or to use UML with extending informal or formal model (Baclawski et al., 2002).

Another development process diagram used to model ontology is by extending the database design techniques, the entity-relationship diagram (ERD) (Fankam et al., 2008). Asuncion et al. demonstrate the utilisation of ERD, then mention the use of the model extension using HERM (high-order entity-relationship model) to adds complex attribute types (key constraints, generalisation and specialization relationships, etc.) (Thalheim, 2000). However, the main drawback in using extended ERD is that heavyweight ontology cannot be modelled. In general, according to comparisons

between the process of Ontologies and software engineering conducted by Wolfgang, Ontology engineering processes are different from SE processes (Hesse, 2005).

There are currently, a set of Ontology development tools available with the focus on the widely used tools, specifically those based on eXtensible Markup Language (XML). For example, Ontology Web Language (OWL), (http://www.w3.org/TR/owl-features/ and 01/06/2009) and its extension Resources Description Framework (RDFS). OWL provides more advance inference to answer queries not necessary predefined promptly. This is made possible because OWL has advance generalization properties then RDFS. OWL is part of the growing stack of W3C recommendations related to the Semantic Web. Below is more illustration of the use of the semantic copied from http://www.w3.org/TR/owl-features (01/06/2009):

- XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents.
- XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes.
- RDF is a data model for objects ("resources") and relations between them, provides a simple semantics for this data model, and these data models can be represented in an XML syntax.
- RDF Schema is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes.
- OWL adds more vocabulary for describing properties and classes: among others, relations between classes (e.g. disjointness), cardinality (e.g. "exactly one"), equality, richer typing of properties, characteristics of properties (e.g. symmetry), and enumerated classes.

There are also other ontology implementation programming languages emerging from knowledge-based systems and intelligent system engineering that support non XML based or traditional applications. These languages have been designed to align with specific domain ontology, for example, *Loom* (Loom, 1997), *Epikit* (Narinder and Michael, 1991), *Express* (Ogata and Collier, 2004) , and *KIF* (Gruber, 1992). Loom is equipped with a powerful tool called classifier. The classifier helps to rebuild the knowledge network relationship tree. Loom is efficient in the conceptual design of ontology and can also be used to manage knowledge. Epikit is useful for exploratory

reasoning; it provides a suite of powerful deductive engines. Express is the standard language for PDES information models, which are logical database designs for shareable product description data. KIF is used to create a common ontology in canonical form such as model libraries and experimental data set, to represent the public-domain knowledge bases. Nevertheless, there are many other ontology's tools and languages; further study and evaluation can be found in Corcho (2003), and Xiaomeng and Lars (2002).

This research takes the initiative to present a unique search technique for software agents based on their functionality (services). The research intended to incorporate ontology analysis and design within the proposed methodology modelling processes. Incorporated ontology in software development methodology is a unique design approach where the existing multi-agent systems development methodologies were not attempted.

2.11 Software Engineering Development Methodology

Software engineering is the "application of a systematic, disciplined, quantifiable approach to the development and maintenance of computer software" (Abran and Moore, 2004). The style of particular collection of processes and procedures that intend to build a software system that meets the requirements then leads to successful implementation is called software development methodology (SDM). All development methodologies have to follow software development life cycle (SDLC). A software development life cycle is the course of processes that software developments go through and consists of: system specification, analysis, design, implementation, testing, maintenance, and delivery of the system (Baird, 2003). It is not necessary to apply the software lifecycle in its traditional form (sequentially) subsequently the development process can be performed following any appropriate approach. There are several approaches to how to implement the software life cycle. For example, Boehm-Waterfall, Agile, Spiral, Extreme Programming, Booch method (Grady et al., 2007), Unified Software Development Process (Jacobson et al., 2000), The Object Modelling Technique (OMT), The V-Model, joint application development (JAD), rapid application development (RAD), and other project specific

or plan-driven approaches, however, all these approaches are categorised as either Sequential (waterfall) or Incremental (iterative).

2.11.1 Waterfall Approach

The waterfall approach was developed in the 60s by the U.S. Navy then gradually enhanced by many software development stockholders. The approach is often considered as the classical development life cycle. The basic concept of the waterfall approach is that it is a sequence of phases, and software developers should clearly delineate the phases: system specification, analysis, design, implementation, testing, maintenance, and end with system deployments, see Figure 2.8. Whenever one phase of development is completed, the development proceeds to the next phase and there is no turning back, furthermore, no jump over or skip and back.

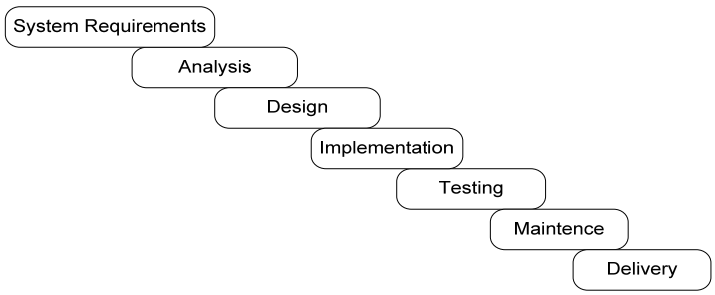

Figure 2.8: Waterfall approach lifecycle.

The advantage of waterfall development is that it allows for departmentalization and managerial control. A delivery schedule can be set with deadlines for each stage of development and a product can proceed through the development process like a production line in a factory, and theoretically, be delivered on time. Development moves from concept, through design, implementation, testing, installation, troubleshooting and ends up at operation and maintenance. Each phase has a goal and is well defined so it is easy to understand. The approach does not consume additional consideration for the system inter-components integration because the system is developed as one entire unit in each phase. The waterfall approach is scalable thus it is suitable for any system size.

The disadvantage of waterfall development is that it does not allow for early reflection or revision. Once an application is in the testing stage, it is difficult to go

back and do some changes that because all the steps are interconnected. The system testing and validation are carried out at late stage of the system development; in addition, the early errors are reflected and flow from the previous to the next steps. Furthermore, the system delivers at once (one-unit at one-time) to the user which creates a sudden impact particularly large scale system difficult to manage.

2.11.2 Incremental Approach

The incremental development approach is a strategy that develops the required system in stages; at each stage there is a deliverable of workable module or subsystem to the user. At each stage the development process will go through a complete software life cycle, analysis, design, build and test. The development process will continue repeating (iteration) in a structured manner until the entire system delivered then finally builds the entire required system. Figure 2.9 illustrates the incremental approach or what is describe by "evolutionary delivery (EVO)" (Gilb, 1985). Gilb state the three concepts for EVO are:

- deliver something to a real end-user;
- measure the added-value to the user in all critical dimensions; and
- adjust both design and objectives based on observed realities.

The principle behind the incremental approach is to derive the project toward a least risk framework. This objective is achieved through an early stage system delivery to the user. The user then tests the fragment or module of the system and provides feedback for any operational failure (coding bugs) or any mismatching between the requirements and the system functionalities (Gilb, 1988).

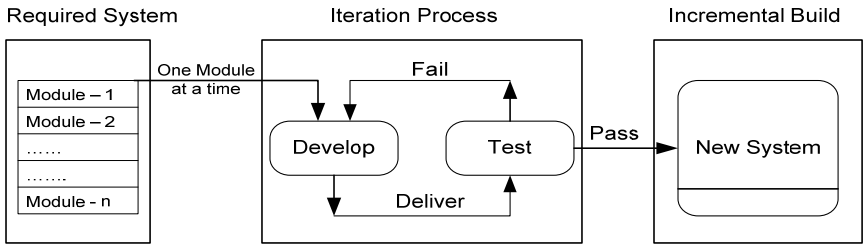

Figure 2.9: Evolutionary delivery process.

The main advantage of the incremental approach is that at each development stage it produces a valuable, usable, testable, workable unit of subsystem. The approach provides flexibility for finding then fixing the errors at an early development stage. The iterative process techniques leads to testing each system's component by the requirement expert (the user) which gradually leads to robust testing. Applying the incremental approach on a software development project creates confidence between the developer and the user which leads to clarify the project development plan, for example, the project visibility, the timelines, project cost.

On the other side, the incremental approach has some disadvantages mainly that the developer has to fragment the system into functional components, where the situation in some types of system makes it difficult to draw the fragmentation lines. In addition, building the system's components integrity in relation to the quality of the entire system operations is not a seamlessly task. Refitting the system components (modules) with each other at each new release requires a substantial configuration effort. However, many software development houses apply the incremental approach successfully, specifically when there is an enhancement of the existing systems (Greer and Bustard, 1996).

Following software development methodology is an essential tool for designing risk free high quality software. Software engineering techniques are a main prerequisite of running a successful software project. Applying a methodical approach results in fewer defects, ultimately provides qualities, better value software. However, the most frequent criticism of applying software development methodologies is that they are bureaucratic and restricted in their operational manuals. The deployment process is lengthy and contains difficult applications which result in slow down of the development process. Generally, for which development approach is better, it is necessary to analyse the nature of the system requirements and the size of the application.

2.11.3 Formal Specification Method

Many software crises have been occurred because the system requirement's specifications are misunderstood by the developer. It is not easy for the software

developer to understand the software requirements, thus the flowchart or any other diagrammatic notation including the normal descriptive languages, all have limitation in describing the subject meaning precisely; furthermore, the normal natural language is expressive, informal language and unspecific which dictates lengthy explanation to describe the same subject.

The formal specification language is a logical structured mathematical notation that embraces semantics and syntax for describing object and their behaviours. Several formal languages have been proposed where each has its focus and type of domain. For example, *Z* specification (Spivey, 1989), and *VDM* (Jones, 1986), are used to write system specifications based on first predicated calculus and formal semantic definitions to provide clear high quality system specification definitions or compliable specifications in some system development cases.

Formal specifications prove that it is a more efficient method for specifying a precise system specification that increases the understanding of the system requirements. Subsequently it works as clear system specification guidelines. Using formal methods needs accurate understanding of the system requirements which entailed to refine the system's requirements document several times. The accuracy and the expressiveness of the formal method guide the developer to match the requirements and design; in addition, it can verify and test the system specification mathematically even before the system development process stars.

The implementation of the formal specification language requires understanding of the formal language and ability to construct a logical model using mathematically. However, there are many advantages for using formal language for precise software system specification:

- formal specifications provide insights into and an understanding of the software requirement and design,
- formal languages are modular and mathematical rule-based characterised by a set of defined command and notation that embrace syntax and semantics. This structure can be used for developing auto transformation tools, from a formal language to a computer software programming language,

- the formal specification is mathematical based therefore it can analyse to prove the system specification consistence and completeness.

Despite the expression techniques of the formal method it is not widely used in software development processes " *Formal specifications and methods have not been widely used in software development.*" (Sommerville, 1992). However, many software development institutions do not consider formal language in their development processes for several reasons, mainly:

- formal specification requires a skill and background in discreet mathematics and logic. Inexperience of these techniques makes the development process difficult and hard to define a system specification particularly when the system is vast or complex,
- some classes of software system are hard to define
- no development supportive tools,
- easy to misunderstand, thus no one schema to define an object,
- no automated compiler to check the semantics,
- long and tedious process.

Formal method is a sufficient tool to document the system detail specification. However, formal method has not been widely used in software development engineering as many software developers consider it as an additional skills. The research excluded the formal method from the proposed methodology for providing more simple development procedures which can be applied by any developers.

2.12 Existing Agent-based Development Methodologies

The aim of this section is to review the existing agent-based development methodologies in the literature, emphasising strengths and weaknesses of the each method. The section focuses on widely used AOS development methodologies namely, Prometheus (Padgham and Michael, 2004), MaSE (Deloach et al., 2001), AUML (Odell et al., 2003), Gaia (Wooldridge et al., 2000) then extended by ROADMAP (Zambonelli et al., 2003) and Tropos (Bresciani et al., 2004).There are three primary reasons for selecting these methods. First, they are well documented in the literature. Secondly, each of these methods has its own software development

tools. Thirdly, these methodologies' artefacts construct agents abstractions, further illustration can be found in the survey in AOSE development methodologies conducted by Dam and Winikoff (2005). The other agent-based development methodologies for example, MESSAGE (Caire et al., 2001), INGENIAS (Pavón et al., 2005), ADELFE (Bernon et al., 2002), AOR Modeling (Gerd and Kuldar, 2004) and MAS-CommonKADS are not relevant because they do not provide any extra agent abstraction and do not cover the software development life cycle. In addition, they were developed for specific application domains. (There is a useful blog and experience documentation on applying those methodologies on software projects, which can be found on; *http://sharon.cselt.it/pipermail/jade-develop*/2003q4/003567.html.

An agent paradigm is characterised by a high degree of complexity with features such as autonomous behaviour, situation awareness, high degree of distribution, and intelligent actions (Bauer and Odell, 2005b, Sycara et al., 2003). Fundamentally, and agent-oriented system features belong to three software engineering practices: knowledge engineering, object-oriented and artificial intelligence. But, there is no particular development methodology from these software practices directly suitable to engineer multi-agent systems. The existing non-agent software development methodologies does not possess the capabilities to design agent abstractions or address the software agent abstraction (Padgham and Michael, 2004, Zambonelli et al., 2003). Because agent software system has unique abstraction and for the success of its analysis, design and implementation a tailor-made development process is a prerequisite requirement (Bauer and Odell, 2005a).

Luck et al., (2004), surveyed the existing agent-based development methodologies and classified them based on their approach, scope basis, phases they cover, syntax and semantics application area, and type of agency support. The diagram in Figure 2.10 outlines classification of the existing agent-oriented software engineering methodologies.

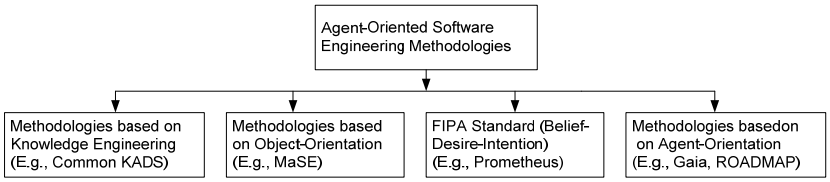

Figure 2.10: Categorisation of AOSE methodologies.

2.12.1 Agent-UML

The unified modelling language is an insufficient basis for modelling agents and agents-based systems (Marc-Philippe, 2003). For this reason the foundation for intelligent physical agent (FIPA) group suggested to extend the unified modelling language (UML) to create a standard modelling schema for agent-based system parallel to what OMG has done to object-oriented systems, which resulted in Agent Unified Modelling Language (AUML) (http://www.auml.org/), (Odell et al., 2003). According to the AUML Web site the general philosophy for using AUML is "*when it makes sense to reuse portions of UML, then do it; when it does not make sense to use UML, use something else or create something new*" (http://www.auml.org/). The AUML notations and diagrams have been tested successfully in modelling agent development activities, agent interaction protocols, and role specification (Marc-Philippe and Odell, 2005). Furthermore, for using AUML efficiently Cabac and Moldt introduce semantics for AUML agent interaction protocol diagram using Petri net code structures, the work is presented in (Cabac and Moldt, 2005).

Despite the semantics of AUML diagrams and FIPA recognition the model minimally incorporated in the existing agent development methodologies. The reasons could be different publications date or not matching with approach used. The research has considered AUML for agent's interaction diagram to follow the FIPA standard and to utilise the powerful representation of its diagrams and notations.

2.12.2 Prometheus

The Prometheus methodology is a detailed process for specifying, designing, and implementing multi-agent systems (Padgham and Michael, 2004). Prometheus consists of three phases; system specification, architectural design and detailed design. The first phase, system specification, deals with determining the system's environment and establishing its goals and functionalities, as shown in Figure 2.11. The environment is defined in terms of percept and actions while the system's functionality is identified in terms of goals and plans. The outcomes of this phase are system goals, scenario and functionality descriptions. The second phase, architecture design, processes the system specification's artefacts further, then develops a high–

level designed agent system using data coupling and functionalities grouping techniques. The outcomes of this phase are agent types that will be used by the application, the interactions between agents, and overall system structure. The third phase is the detailed design, in which phase three main artefacts will be delivered: agent capabilities, plans, and events. The implementation process is kept open for the developer to choose a preferred platform. But the use of the JACK development environment is strongly recommended because both Prometheus and JACK are based on BDI architecture. Prometheus development tool PDT provided an option to the developer to automatically generate JACK skeleton java code.

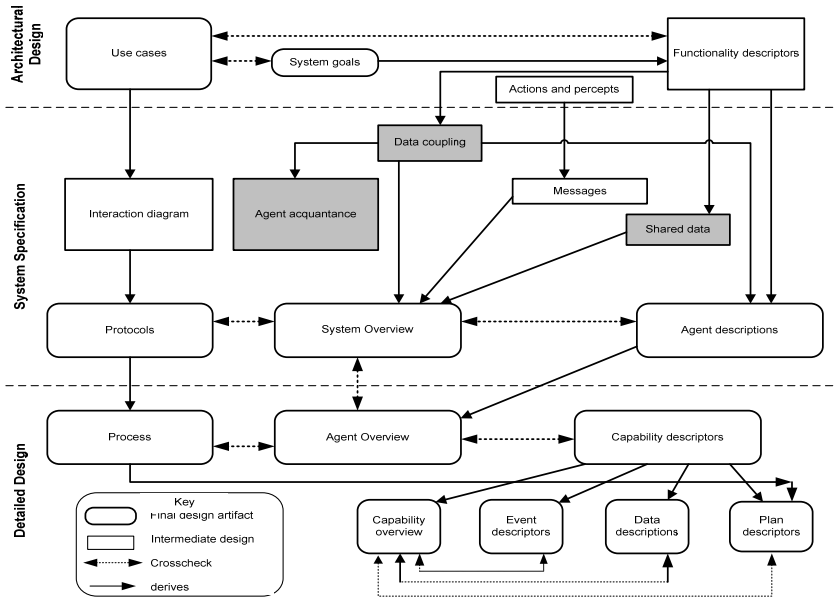

Figure 2.11: Prometheus architecture.

The research has deployed Prometheus in designing multi-agents systems as part of the research evaluation experiments and published the results "*Enhancing Prometheus to Incorporate Agent Cooperation Process*" (Alhashel et al., 2008). The paper explained Prometheus deployment including using PDT and its interface with JACK programming language then discussed its strength and weaknesses. Prometheus is a descriptive methodology which depends on a set of forms and information tables in its analysis and design supported by numbers of its own notations. The PDT support both the analysis and the design phase with two useful options, first it checks the design syntaxes, second it checks the labelling (description) consistency to avoid duplicate descriptions or misspell the entities description during the design process.

The main advantage of using PDT for applying Prometheus is to utilise its abilities to automatically generate the JACK java skeleton implementation codes that help the programmer to extend it further to a complete system. Prometheus's detail design is well mapped into JACK agent development language with some adoption. For example, JACK does not have concepts corresponding to percept and actions therefore percept is presented as events and action performed within the plan body using Java code.

More advantages are:

1. Prometheus design has the capacity to engineer agent internal reasoning, BDI or intelligent agent. This will satisfy part of an agent cooperation process if developed based on belief sharing.
2. Prometheus is goal oriented, dealing with goals starting from analysis up to implementation.
3. It covers the complete software development life cycle.
4. Prometheus is implementable oriented and tools based development. Its final products are well mapped with the implementation programming language, JACK (Vaughan et al., 2003) and smoothly use the JADE platform as a runtime environment (Bellifemine et al., 2007a)..

The experiment also reveals the weaknesses of Prometheus:

1. The main disadvantage in Prometheus is that it is not designing a group of agents working together to achieve a common goal. The methodology design independent group of agents, Prometheus designing individual BDI agents systems linked by messages specified in the interaction diagram which is directly borrowed from object-oriented design (Padgham and Michael, 2004) *pages 28, 68*.
2. Prometheus is a descriptive methodology therefore it could mislead the developer because natural language has multiple explanations. In addition, linking between graphical diagrams and text based are not practical.
3. There is a gap between the three phases, the specification artefacts are not seamlessly converted into design details constructs and the detail output does not seamlessly flow into the design details. For example converting the capability into events and plans has no clear guidelines on how it can be done.

4. The PDT is helpful to a certain extent but it is incomplete tool and not yet promoted to the industrial production. This research experience it shows that there are some errors and some menu options are not functioning.

2.12.3 PASSI

A process for agent societies, specification, and implementation (PASSI) is a comprehensive development methodology for agent-based software systems. PASSI view agent-based systems as societies of agents interacting with each others forming a team or society of agents. PASSI originated based on both object-oriented and intelligent approaches, using UML notation to model the agent-based system. For specific needs of agent design, the UML semantics and notations are extended to satisfy agent concepts that are not considered in UML. The PASSI architecture illustrated by Figure 2.12 is composed of five main models: System requirements, Agent society, agent implementation, Code, and Deployment. Each model is composed by a number of sequential processes performed at a particular phase of the system development and design process.

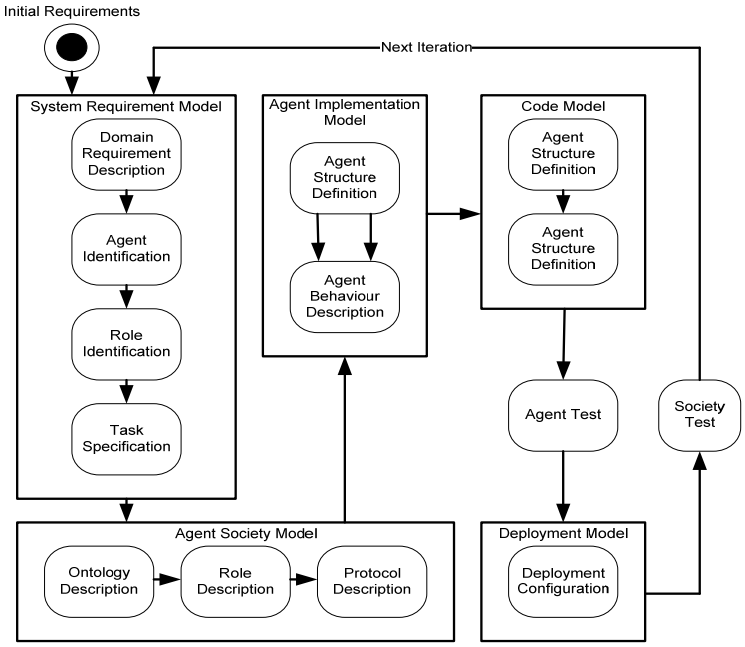

Figure 2.12: The models and phases of the PASSI methodology.

The PASSI development method is iterative in two cycles. First, it is led by new requirements and involves all the models. The second iteration occurs, involving only

modifications to agent implementation model. The implementation model characterised by a double level of iteration: the multi-agent and single agent level. For each level there are structure definition and behaviour description processes. On the multi-agent level the structure definition relates to the cooperation and tasks and the behaviour description relates to the flow of events depicting cooperation. On the single agent level the structure definition related to the attributes, methods and inner classes where is the behaviour description relates to that specified in an appropriate way.

However, PASSI development methodology is the only agent method that implement ontology concept to support the agent knowledge. It is also comprehensive (full software life cycle) starting from requirement to implementation and coding. Despite these advantages, there are some drawbacks to the PASSI development techniques as follows:

1. There is a transition gap between the design and implementation phase, the final artefacts do not map into the FIPA standard or map into JADE as a FIPA standard platform.
2. The analysis phase needs further detail to refine the design artefacts and make it easy to further process in the design phase.
3. The agent identification phase does not explain how agent can be identified. If the name of the role becomes an agent what about if the role is imbedded in another role (sub-role)? Are there further refinement techniques to identify the system's roles?
4. PASSI design open agent architecture and it does not specify and instance deployment.
5. Software agent definition relied on agent goal oriented but it is not clear how the system's goals presented.

Overall PASSI techniques direction is promising for designing close independent multi-agent systems. The design techniques need further enhancement to improve artefacts and to match agent-based system abstraction with the implementation.

2.12.4 Gaia

The Gaia methodology is intended to develop multi-agent systems in a form of organisation structure. Gaia takes the view that a MaS is a society or an organisation

of agents. Each organisation represents a goal or sub-goal of the system's objective. The Gaia methodology models both the macro (social) aspect and the micro (agent internals) aspect of the multi-agent systems. The Gaia methodology only processes three phases of software life cycle; analysis, architecture design, and detailed design, as described in Figure 2.13. Capturing and modelling the initial requirement is not part of Gaia processes, for this reason the Gaia methodology used ROADMAP as extension to carry out the early requirement analysis (Thomas et al., 2002).

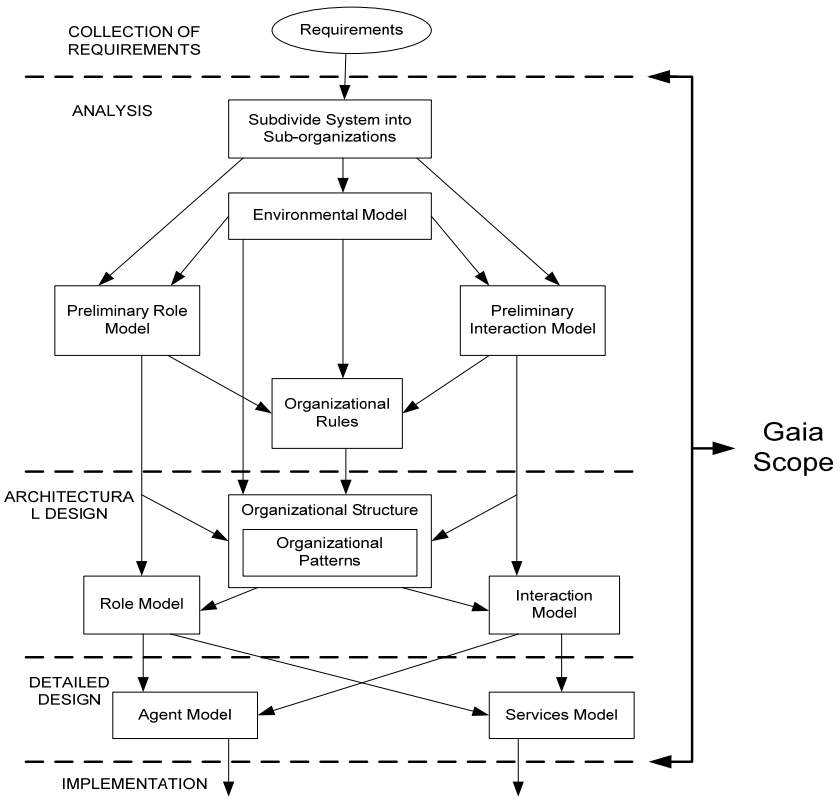

Figure 2.13: Model of the Gaia methodology (Hederson-Sellers and Giorgini, 2005).

The Gaia *analysis phase* constructs the application conceptual organization tree and all the preliminary related models that support the organisation structure. At this phase the system will be divided into sub-systems based on independent functionality, coherently, or modularity techniques then the preliminary role model, environmental model, preliminary interaction model, and organisational rule model are specified. The role is the centre attribute in the Gaia design strategy in connection with the interaction and the protocols applied. The role in Gaia defines the agent tasks in term of functionalities, activities, responsibilities, and the protocols used in concert with other agents and in respect to its organization. The output of the analysis phase

processed further by the architecture design phase to establish the system organisations basic blocks. The architecture design phase will develop and finalise the organisation structures, the role model, the interaction model, and the patterns model. To finalise and integrate the organizations, then Garlan and Shaw's (1994) techniques are applied and used as a guideline to identify and refine the organisations design.

Finally the details design phase is responsible for constructing the agent model and the services model using the analysis and the architecture phases' artefacts. The agent model represents the agents that construct the required system in forms of organizations structure. Each agent is represented in terms of role. The role consists of functionality, activities, responsibilities, interaction protocols and patterns.

The idea of a system as a society or organizational structure is efficient in multi-agent systems concept. The role attributes; responsibilities, permissions, activities, and protocols form a coherent role which finally is embodied by an agent which is a useful approach in building a coherent agent. Overall the Gaia design is a closed system and each organisation (agent society) is a rigged sub-system which does not allow any dynamic behaviour. This fundamental drawback is the core of the MaS design where the research DMMAS intended to recover it. Gaia methodology is incomprehensive as it does not cover a complete software development life cycle. Another limitation is that there are no particular techniques or standard notations that can follow through the design process. The UML, AUML and the Gaia notation in addition to the ROADMAP (extension) modelling and techniques make the development process complicated. The methodology's components are not well integrated "*Gaia does not commit to any special-purpose notation for its models.*" (Hederson-Sellers and Giorgini, 2005). Finally, the Gaia methodology is subject to further enhancement and improvement over its processing and organization structure concept.

2.12.5 MaSE

The Multi-agent Systems Engineering (MaSE) methodology described in Figure 2.16 consists of two phases, analysis and design (DeLoach and Kumar, 2005). MaSE is an

object-based methodology applying a strong top-down software engineering mind-set that is more suitable when designing heterogeneous database integration systems to a biologically based, computer virus-immune system to cooperative robotics systems (Hederson-Sellers and Giorgini, 2005). MaSE is architecture-independent, views the agent-based system as a goal oriented model characterised by an agent that may or may not possess any intelligence. In the analysis phase the focus is on identifying and defining, then classifying the system-level goals using KAOS approach (Axel van, 2000). Goals are embracing sub-goals (roles), and roles are defined explicitly via a set of tasks, which are described by finite state models. Initially, goals are extracted from the system requirements then restructured in main and sub hierarchal goal tree diagrams. MaSE constructs agent class in two steps first, defining the agent architecture, second defining the individual architecture's components while MaSE does not enforces or pursue any particular agent architecture. The other important step in the design phase is to develop the deployment diagram which is similar to the UML deployment diagram. The deployment diagram represents the system configuration which explains the types and the number of agents that make the system and their implementation platform.

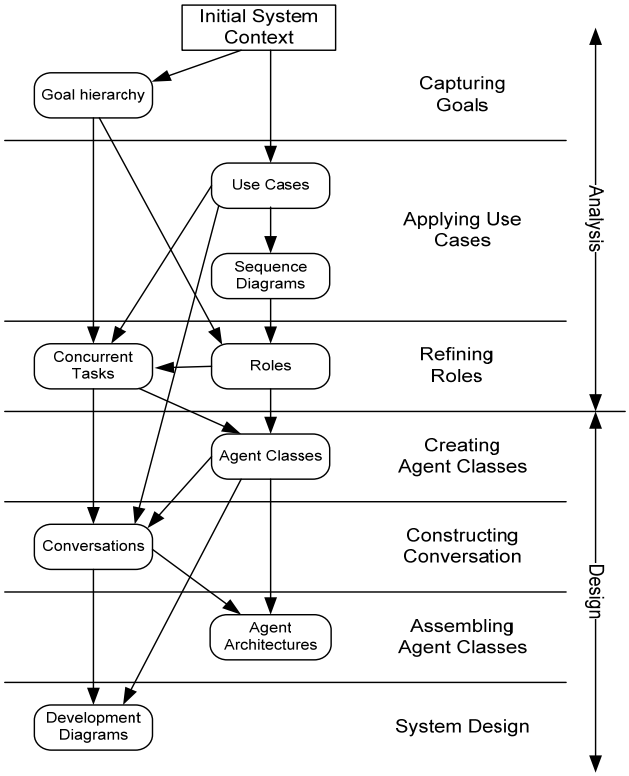

Figure 2.14: MaSE overview diagram.

For the productivity improvement MaSE is supported by a software development tool called agentTool that has the potential to automate the development process and cross check the design semantics to avoid design errors and labelling redundancy. The agentTool has a menu option to generate Java skeleton program codes and also agent conversation verification process based on sender and receiver existence.

MaSE is independent of any particular agent architecture or development environment, It has a traceable development approach through its processes. In addition MaSE lends its development components to integrate with any agent architecture. Robinson (2000) has deployed MaSE on verity of agent architectures including reactive, BDI (Belief Desires and Intentions), knowledge-based and planner based. MaSE is also supported by a software development tool AgentTool which adds significant advantage to the development productivity.

MaSE engineer close multi-agent systems, with limited agent autonomy. Their final software system does not exert any dynamic actions at runtime, it generates a rigged multi-agent system (Alhashel et al., 2007). Despite the advantages, there are four fundamental weaknesses within the MaSE approach, summarised as;

1. It designs a closed MaS system, therefore an agent cannot demonstrate its autonomous concept subsequently there is no cooperative actions. The research argues that software agent systems without cooperation and without autonomous behaviour lead to object-oriented system under agent label (Alhashel and Mohammadian, 2008).
2. MaSE does not support common Ontology schema instead embedding Ontology within agent task communication protocols "*In general, however, MaSE implicitly defines an ontology that is embedded in the task communication protocols and is implemented within each agent*." (Hederson-Sellers and Giorgini, 2005). Furthermore, MaSE does not follow a particular agent's architecture, agent may or may not possess intelligence (Padgham and Michael, 2004).
3. MaSE is not intended to build a system with dynamic team formation, instead it predefines fixed organization at this design stage. This techniques leads to a rigged system (Fisher et al., 2006).

4. Padgham and Michael states that "*Thus, MaSE (intentionally) does not support the construction of plan-based agents that able to provide a flexible mix of reactive and proactive behaviour*" (Padgham and Michael, 2004).

2.12.6 Inadequacies in the Existing Development Methodologies

In the previous section the main software agent development methodologies have been studied and discussed. This section generalises the limitations in those methodologies.

The main deficiency in the existing MaS development methodologies is that they do not address the techniques to build cooperative open multi-agent systems. The organization structure proposed by *Gaia* and *MaSE* is actually a predefined model where agents are set earlier (at the design stage) into predefined constant groups. As this design approach impedes agents playing their key concepts (autonomous) behaviour so there is no need for a dynamic cooperation process. Eventually, these development approaches result in building close multi-agent systems. Furthermore, *Prometheus* built multi-agent systems from individual intelligent agents based on BDI architecture without any design consideration for a group of agents working together to achieve a common goal instead of the Prometheus design independent rigged multi-agent system. Padgham and Michael (2004) Prometheus's authors stated that *"There are a number of types of system that it does not yet handle well. For Example, Prometheus as described in this book does not address agent teamwork or mobile agents."*

Tropos is a software tools development methodology pursuant to BDI agent architecture that is represented at the lowest level by component plan diagrams proposed by Kinny and Georgeff (1996) to specify the internal processing of actors on how to perform the system goals. The methodology does not address development techniques for agent's cooperation or dynamic behaviour. In fact, Tropos is object-oriented based suitable for generating an e.business type of system and has nothing to do with multi-agent systems in the general understanding. PASSI is driven from embedded robotics applications. It introduces multi-agent social behaviour in interaction communication based on share knowledge using an ontology domain. The

multi-agent concept in PASSI relies on agents residing in conceal boundary. This approach also does not satisfy the open MaS concept. The AUML is not evaluated because it is an attempt at standardisation of agent-oriented modelling and a notation not a methodology. Overall, this concludes that the existing MaS methodologies do not address the open MaS design classes and components (Alhashel et al., 2008).

To outline the existing limitations Table 2.1 summarises three research papers evaluating Gaia, MaSE, Tropos, and PASSI (Dam and Winikoff, 2003, Sellers and Gorton, 2003, Tveit, 2001). The result of the evaluation shows that the existing methodologies are either not deploying cooperation or are using interaction between agents to create cooperation where the intra agent social structure is not addressed (Iglesias et al., 1999, Wood and Deloach, 2001). In the agent collaboration process, communication is used as a tool for coordination between agents using standard protocols, such as FIPA-ACL, KQML, or KIF.

Method	Agency Type	Cooperation Model	Cooperation Base	Cooperation Type
Gaia	Any	UML Interaction Diagram	Fix Organisation Structure	Independent
MaSE	Any	UML Interaction Diagram	Close system agent protocols	Independent
Prometheus	BDI	Object-oriented interaction diagram	-------	-------
Tropos	BDI	Communication base on Ontology	Close system agent society	Independent
PASSI	Mainly Robotics	Object-oriented interaction diagram	--------	-------

Table 2.1: Illustration of cooperation in AOSE.

To create cooperation between agents there must be logical built in components within agents logic, for example, the process of team formation (Cohen et al., 1997a), agent goal adoption of other agent goal (Luck and d'Inverno, 1996) , the shared plan or the process of team formation. All these concepts need to be represented within a development methodology to transfer the MaS powerful concept to the real world. In fact, it is hard to find software agent systems on an industry level that have been developed using one of the existing methodologies;

"*Many of the existing agent-oriented methodologies are not yet ready to be used by industry developer: they are under research, and either focus on specific*

aspects of agent design , or are not describe in sufficient details." (Padgham and Michael, 2004).

The research focused on building cooperative open multi-agent systems that are characterised by autonomous agent autonomous behaviour that can function in open distributed heterogeneous systems and have the ability to dynamically, at system runtime, form a team of agents to achieve a common goal. To build this type of system there are a list of components and functionalities that must be available within the methodology to build such system. The existing methodologies and practices did not satisfy these requirements. However, the research found that the existing methodologies attempts to possess some useful concepts in designing an individual agent and present some agent classes such as goal, role, task, subtask, functionality, capability, plan, organisation and agent mantel attitude, like BDI, and knowledge representation based on ontology or databases. These classes form a foundation for the research aim.

2.13 Agent Communication Languages

From the perspective of the research aim it is important to investigate the agent development infrastructure where agent communication is a key issue. In this section the research concisely studies the existing agent's communication languages and technologies.

The performance of the communication enables the agent to interact with others to share knowledge and exchange information to enable them to coordinate their actions in any cooperation process or to delegate on behalf of the user to achieve the tasks. In the core design of an agent-oriented system or the multi-agent systems, the communications model must be designed with embedded knowledge about the object domain or about other agents (Timothy and Chris, 2001). Exchanging information and knowledge between autonomous software agents required a shared, reusable communication language that the potential to allow an agent to share a common syntax, semantic, and pragmatics models (Tim et al., 1994). The processing of information to share knowledge between agents, or forming a common syntax, is the

major problem in the agent communication field. There are three aspects of communications:

- syntax: how the systems of communication are structured (grammars);
- semantics: what the symbols denoted (meaning, logics);
- pragmatics: how the symbols are interpreted (compilations).

In general, the structure of the existing agent communication language was inspired by the Speech Act Theory (SAT) introduced by Austin (1962), then the theory was developed further by Searle (1969). Marco and Mario (2002), through their analysis of agent speech act as institutional action, found that the intelligent systems attempt to incorporate the concept of the SAT into agent communications. furthermore, Fasli (2007), state that "*Perhaps the most influential theory in agent communication is that of speech Act*.".

This study helps the research to select the most suitable communication practice and applicable language, then to utilise it into DMMAS. However, this requires further investigation of the main agent's languages; KQML, ACL, and KIF are prerequisites where each has its own approach and application domain (Marco and Mario, 2002).

2.13.1 Knowledge and Query Manipulation Language (KQML)

Knowledge Query Manipulation Language (KQML) is proposed by the Advance Research Projects Agency (ARPA) to support the National Information Infrastructure (NII) strategy then developed by the External Interface Group (EIG) as part of the Knowledge Sharing Effort (KSE) project. The purpose behind the KQML is to facilitate an open, large scale, knowledge-based system with the ability to communicate between two knowledge-based systems or conventional database management system (Tim et al., 1994). The KQML is generic knowledge query language; it is not dedicated to any particular system architectures or development environment (Kone et al., 2000). Finin et al., (1994), modified KQML language to support the agent's communication process. As a result, using the KQML, enabled agent to send inquiry messages or advertise themselves or on other hand agents able are receive an answer to their inquiries or information about other agents. The structure of the KQML language is divided into three layers, as depicted in Figure

2.15: the communication layer describes the sender, the receiver, and the message ID; the message layer defines the performatives and message format; the content layer holds the actual message, the ontology of the object and term used, and the content language. The processing sequence of the KQML language is to first deal with the pragmatics (identify the target node and establish the communication) then deal with the semantic (the meaning of the contents).

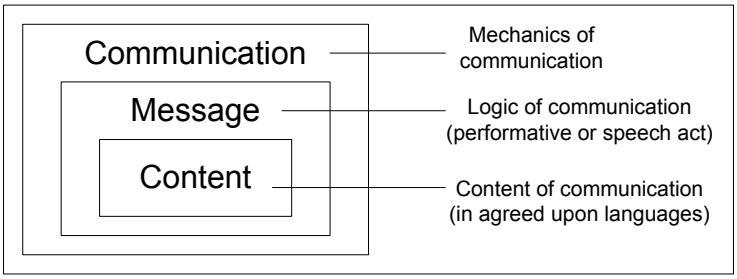

Figure 2.15: The three layers of the KQML communication language.

The KQML language schema is divided into performatives and performatives' argument. The performative is used to communicate with the facilitator and the argument will guide the facilitator to the preferences. The performatives are reserved words divided into seven basic categories they are: basic query, multi responses, response, generic information, generator, capability-definition, and networking performatives. To explain how KQML interpret an agent communication for example, assume that a conversation about advertising job vacancies in the university and "A" is the application assessment agent, "R" is the receiver agent, and Agent "A" broadcasts advertisement for recruitment;

```
(ask-all
:sender A
:receiver R
:reply-with ID1
:content "job (UNIVERSITY, [?accountant, ?vacant])"
:language Prolog
:ontology JOB-VACANT)
```

Agent "R" inform agent "A" that it submitted an application to fill this vacant job;

```
(tell
:sender R
:receiver A
:reply-with ID1
:content "submit (UNIVERSITY[?accountant, ?vacant])"
:language Prolog
:ontology JOB-VACANT)
```

After assessing the applications, agent A informs agent R that his application is successful;

```
(ask-one
:sender A
:receiver R
:reply-with ID2
:content “successful (UNIVERSITY[?accountant, ?vacant])”
:language Prolog
:ontology JOB-VACANT)
```

The KQML supports a wide range of agent architectures regardless of their implementation languages. It is also an extensible communication language to provide a wider range of performatives depending on the application domain requirements.

There are also limitations in the KQML, for example, the agent communication is a lack of standardisation in the actual transport of messages. In addition, the semantics of the language have not been rigorously defined to lead to interoperability issues (Labrou and Finin, 1998). The KQML performatives do not cover all categories of discourse. For this reason, subtle and fine-grained interactions between heterogeneous agents cannot take place (Kone et al., 2000). There are no techniques to respond or notify the sender about the message status after it has been sent it. When we send a message we do need a conformation it has been received. Finally, KQML performatives are limited to assertive and directive categories, inappropriate choice of performatives and different interpretations of KQML performatives (Bouzouba et al., 2001).

2.13.2 FIPA Agent-Communication Language (FIPA-ACL)

Foundation for intelligent physical agents (FIPA) specified Agent Communication Language (FIPA-ACL) as standard agent communication language. The structure of the FIPA-ACL contains a set of one or more message parameters. Each parameter has its own syntax and which parameters are needed for effective communication session remain optional and depend on the situation, the only mandatory parameters are the *sender, receiver, content, and performative.* Table 2.2 lists all the FIPA-ACL parameters and their categories. The FIPA-ACL is an open architecture type of communication language similar to XML, therefore the user can define additional

message parameters provided that a string "x-" is predefined prior to the non-FIPA standard parameters. Generally the FIPA-ACL is a message handling model, it does not contain the communication body of the message or the message semantic. Instead the FIPA-ACL work as standard vehicle schema that capture the predefined message including the message interpretation in terms of ontology then broadcast the contents from the sender to the receiver. Each FIPA-ACL message representation specification contains precise syntax descriptions for FIPA-ACL message encoding based on XML, text strings and several other schemes.

Parameter	Category of Parameters
performative	type of communicative acts
sender	participant in communication
receiver	participant in communication
reply-to	participant in communication
content	content of the message
language	description of content
encoding	description of content
ontology	description of content
protocol	control of conversation
conversation-id	control of conversation
reply-with	control of conversation
in-reply-to	control of conversation
reply-by	control of conversation

Table 2.2: FIPA ACL Message Parameters.

2.13.3 Knowledge Interchange Format Language

The Interlingua Group belonging to ARPA Knowledge Sharing Effort (KSE) project develop a common language for expressing the content of a knowledge-based system. This group has published a specification document describing the Knowledge Interchange Formalism based on first order logic (Keller et al., 2004). A complete KIF report can be found in Genesereth et al. (1992).

The KIF is a formal language including both a specification of syntax for the language as well as a specification for semantic. It is applied to support the translation from one content language to another or as common content language between two agents

which use different native representation languages. Since a language must represent the information and queries aspects KIF extended SQL, and LOOM toward building a common language. In fact, KIF originated in a Lisp application and inherits its syntax from Lisp. However, unless there is a shared framework of knowledge in order to interpret messages, what is happening is not sharing semantics, but a sharing an ontology (Tim et al., 1994).

There are some advantages in using the KIF that it has semantics correlation between the terms and sentences of the language and conceptualization of the world. The formalization of the logic used is interpretable, subsequently executable in the form of knowledge based logic. But since the KIF is a formal language it is also true that different users will implement the same domain of objects, functions, and relations in their own way of understanding to that domain. This will confer unambiguous meaning on the objects described, perhaps for the same domain intended. Subsequently, in a multi development environment, the same object will be defined by different logic and knowledge descriptions.

A survey of the existing agent-based application development methodology shows that there are no deployments for KIF language. Generally, defining the knowledge in a formal language is a complex process. This conclusion is stated directly by the KIF developer bodies themselves in their report; "*Unfortunately, the formal details of KIF are quite complex*" (Genesereth et al., 1992). However, the research is not on KIF implementation in its design process and this avoids the complexity and knowledge mismatching in an open multi-agent system environment.

2.14 Agent Software Development Platform

Closing the gaps between design outcomes and implantation technologies leads to a successful software development process which should help the developer to map the design's artefacts into implementation cods seamlessly. In practice, the differences between available implementations are too fundamental to be ignored (Sudeikat et al., 2005).

The concept of agent-oriented systems has its own unique characteristics, in both internal structure and external behaviour. The agent's autonomous behaviour entails a dedicated environment that provides the necessary components and tools to support its free performance. The conventional software development languages for example, Object-oriented run under MS C# under .Net or Sun Microsystem Java under JBuilder or Java under Eclipse currently support the building of agents despite it not allowing complex mental attitudes to be represented (Anal et al., 1999).

2.14.1 Java Agent Development Environment (JADE)

Java Agent DEvelopment framework (JADE) is a software development framework to develop multi-agent systems in compliance with the FIPA specifications (Bellifemine et al., 2007b), Figure 2.16 illustrates the architecture overview. The central idea in JADE is to provide runtime environment to software agents and associated open architecture types of integration. Therefore agents can communicate and interact with each other and use each other's services. The communication run is based on messages of request following FIPA-ACL syntaxes through built-in agent message systems (AMS). Each agent operating on JADE platform has to register itself by a unique name "agent identification description" (AID) and identify its services into a directory facilitator (DF), which makes agents identifiable to other agents across the platform. JADE uses yellow pages techniques to demonstrate agent services. In addition, JADE provides a number of built-in libraries, classes and predefined packages. To examine the availabilities refer to Bellifemine et al (2007b).

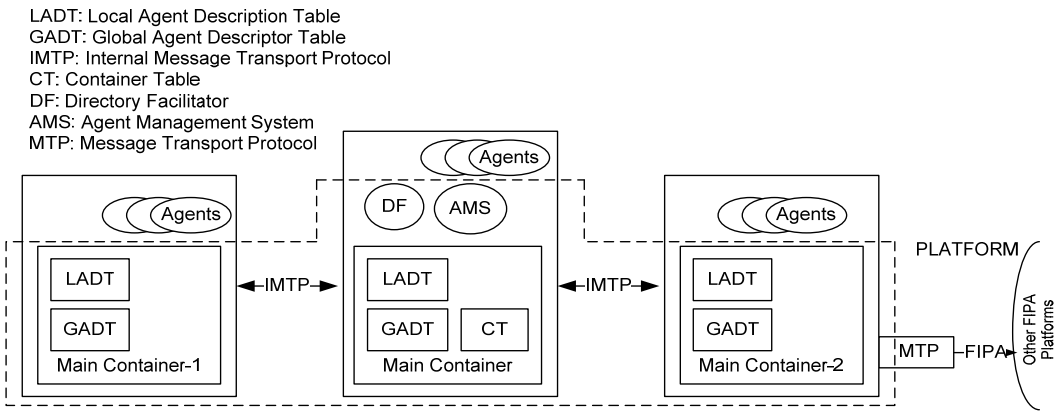

Figure 2.16: JADE architecture (Bellifemine et al., 2007b).

The essential parameters that a development methodology should provide in order to interface with JADE structure are:

Agent ID: Agent identifier unique, global name to each agent.
Agent classes: jade.core.agent: based class for agent
jade.core.behaviours.*: behaviour
jade.core.lang.ACLMessage: message class
jade.core.AID: Agent identifier

Behaviour: represent a task that an agent
Messages: set of interaction messages support by ontology schema.
Services discoveries: registry table for each agent located on each container.
Interaction protocols: the ACL used in the interaction (KQML, KIF, or FIPA-ACL).
Agent mobility: the physical and virtual machines of the network that agent travels cross.

Currently, there are no agent-based development methodologies that provide these inputs as part of their design artefacts, and the development process is ad-hoc. This AOSE imperfection occurred because there is no consent on what constitutes MaS structure and how it is supposed to build then operate. For example, Prometheus is BDI architecture, Gaia is organisation structure, MaSE is role based, and the other were published before JADE released. However, JADE is the most popular agent platform and is widely used for its abstraction, usability and is compliant with FIPA (Bellifemine et al., 2005).

Overall, JADE has been developed with robust multi-agent systems development components that assist the agent developers with a set of abstractions and built-in classes and packages. JADE also has the potential to run the MaS and control many system exceptions. Comparing JADE with the other agent platforms for example, JACK, Aglet, and Cougaar show that JADE is the most appropriate MaS platform, a comparison schema available in (Leszczyna, 2008, Raquel, 2007, Vrba, 2004).

2.14.2 JACK

JACK Intelligent Agents (JACK) is an agent-oriented development environment built on the Java programming language (http://www.agent-software.com/, 07/07/2008). JACK extends Java classes and constructs at the same time offering specific extensions to agent autonomous behaviour. Therefore, JACK is fully integrated with Java and its source code must be compiled on regular Java platform before being

executed. In this regard, first, JACK language will be compile and converted to pure Java code, using JACK compiler then the result code can be run on any Java platform. JACK is dedicated toward the implementation of BDI reasoning concept type of agents.

In agent-oriented software engineering practice, the Prometheus development methodology implementation phase is well connected with JACK, the interface between both tools depicted in Figure 2.17. Prometheus development tool (PDT) provides JACK code skeleton option within its menu tool bar options. If it selected then it will generate the outline main code. A complete research project of travel agent system (TAS) has been developed as part of a research paper (Alhashel, 2008). The result shows that JACK is able to provide substantial agency abstraction base on BDI reasoning concept but it does not contains the agent cooperation constructs, beside that JACK is coding specific architecture while not all the software agent-based systems are BDI architecture. It also recommended following the FIPA agent architecture platform standard where JACK not compatible. However, the research does not recommend the BDI agent architecture but follow the FIPA standard for this reason JACK is not considered an implementation language for the proposed methodology.

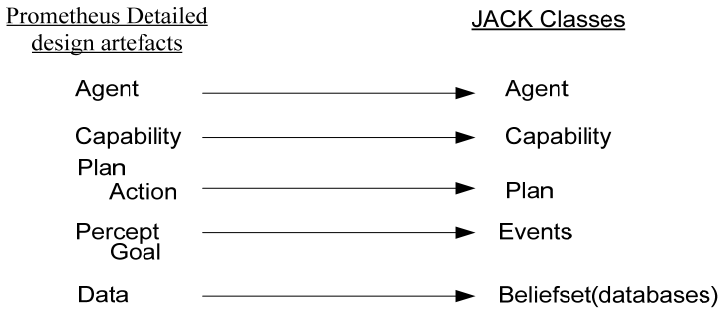

Figure 2.17: Relationship between Prometheus construct and JACK abstract.

2.14.3 Aglet

Aglet is a Java based agent-based development platform developed by IBM Tokyo, focusing on agent's mobility crossing different networks. Aglet views the agent-based system as an individual intelligent agent operating over a large scale distributed environment with vast information repositories, databases and mobile code similar to

the internet. These agents are able to roam the internet/large system to answer the user enquiries. However, using Aglet is straightforward, following an applet is like developing techniques where the programmer defines a few methods to implement the agent's main behaviour. The platform is open source and can be found on the Aglet web site: (http://aglets.sourceforge.net/, 23/07/2008).

The research excludes Aglet from the implementation approach for its specific modelling, i.e. mobile agency, which is outside the research development domain. In addition, Aglet is not FIPA standard as it has been developed when MASIF (Mobile Agent Systems and Intercommunication Facilities) was the agent standard. It is also difficult to learn Aglet because of a lack of documentation (Ferrari, 2004). Generally, Aglet is almost obsolete, as no maintenance or bug fixing has occurred for a long time (Leszczyna, 2008).

2.14.4 Cougaar

Cougaar (for Cognitive Agent Architecture) is an agent-based application development environment base on Blackboard architecture and Plug-in techniques (see Figure 2.18). Cougaar in its original copy developed to function as a total integration solution component for the United States Department Of Defence (DOD) logistic software systems to integrate a complex monolithic software pool consisting of an multi distributed platform with a different operating system and different databases for different business processes (BBN, 2004a).

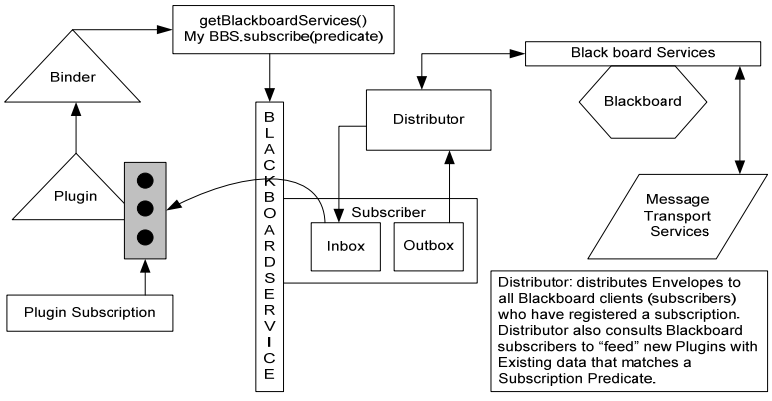

Figure 2.18: Blackboard Service API Internal Concepts (BBN, 2004b).

Cougaar provides powerful, expressive, scalable and maintainable application as it is a code baseline with the ability to construct dynamically complex distributed systems.

Cougaar uses dynamic plug-in techniques to load or unload agents while the system is running without any interruption to the system. This is a powerful tool introduced by Cougaar and verifies agent-based system characteristics as loose cabling and high cohesive software architecture. Furthermore, the plug-in facilities resolve the agent-based system problem as an incremental, scalable open distributed type of system that can be developed from anywhere at any time by anybody.

However, Cougaar design agent-based system as a set of communities each community forming a node. A node is a single Java Virtual Machine (JVM) instance that may contain and maintain multiple agents, where all the agents of one node share all the node resources; CPU, memory pool, disk and compete for incoming and outgoing bandwidth traffic. Thus agents of the same node exchange the message locally, messages to agents in different nodes must use the Cougaar Message Transport (MT). The core component of Cougaar is the blackboard architecture that controls and manages all the nodes and node's agents. In addition, the Blackboard is where the agent behaviour and business logic are plugged in, therefore it is the main agent service provider and infrastructure source, the Blackboard is what give the autonomous behaviour to an agent.

Taking a close look to Cougaar and JADE architectures reveals that there are differences in component naming and terminology and techniques used but there are similarities in the principles and overall design, for example:

Criteria	Cougaar	JADE
Techniques	Blackboard	Agent Management System
Organisation	Domain	Main container
Structure	Node	Container
Services	Plugins	Directory facilitator
Registry	Agent subscribe UID	Agent ID
Messages	Message transport	Message transport

Table 2.3: Comparison between Cougaar and JADE.

There are also considerable functional differences between Cougar and JADE platforms in the implementation, availabilities, maintainability, usage, agent internal

structure, and following agent standard FIPA architecture in addition, Cougaar has more services parameters. With regard to other agent-based platforms focus, Cougaar aims the Blackboard agent techniques for application integration rather than business agent-based systems. Further comparison criteria are presented in Braubach et al. (2008). For these reasons the research does not consider Cougaar implementation in the research case study.

2.15 Summary

This chapter reviews and compares agent versus object then summarises that there are fundamental mismatching between the two concepts. Subsequently, object-oriented development tools are not adequate to model the agent-oriented software. Furthermore, the chapter investigates the software agent system infrastructure focusing on the existing multi-agent systems development methodologies and conclude that these developments focus on building independent (close) MaS and do not address the cooperative open multi-agent systems development issues.

The chapter explores the available multi-agent systems development tools in the literature including MaS design methodologies, agent standards, communication languages, and agent runtime platforms. It discusses the weaknesses and limitations in the MaS software engineering development, and concludes that there are useful software tools and background in the fields. However further enhancement is required to transfer the multi-agent system design from closed systems to the open system environment. In addition, the literature provides evidence of the need for open structure software tools, and a new approach to design agents autonomous behaviour and agent cooperation processes. These discussions underpin the research background and motivation to close the existing gap by providing a new development methodology for building cooperative open multi-agent systems. Based on this conclusion the research focuses on the ontology approach as a new implementation solution. This ontology approach is efficient to facilitate the agent autonomous behaviour and systems that function in open environment.

However, while this literature creates the research background the next chapter reviews more deeply the relevant insights for multi-agent systems design and

development methodologies emphasising the strength and inadequacy of each method from different software engineering perspectives.

Chapter 3 A Review of Existing Agent-based Software Engineering Methodologies

3.1 Introduction

The objective of this chapter is to investigate the relevant existing multi-agent systems development methodologies, including Prometheus, MaSE, and Gaia. These methodologies are then evaluated to investigate their limitation and weaknesses. The evaluation process is based on tabular frameworks proposed by Stum and Shehory (2004), and Yogesh et al. (2008), and Braubach et al. (2008). To map the evaluation criteria with the open cooperative multi-agent systems, the framework criteria are extended to include; application domains, development approaches, processes involve, and techniques used. This chapter is based on a travel agency system (TAS) case study presented in two research papers that have been published by this researcher (Alhashel et al., 2007), and (Alhashel et al., 2008). The chapter aims to summarise the limitations in the existed development methodologies and establishing a background for building a new enhanced development methodology for open cooperative multi-agent systems through a motivating, distributed, cooperative application, the Travel Agency System (TAS).

3.2 Travel Agency System: Motivating Problem

In order to test the strengths and weaknesses of the existing multi-agent systems development methodologies, the research uses the domain of a travel agency system (TAS) as an application scenario. The motivation for selecting TAS is twofold. First, TAS is widely available in the real world and easy to understand and at the same time it is a distributed system where the *hotel* accommodation booking, the *car* hire, and the *flight* reservations are each represented by an independent unique agent. The TAS scenario includes coordination characteristics where the system must be able to book for a customer as per his preference of date, place, and services class from a variety of open distributed agencies emphasising the preferences and availability. Reserving a seat on a flight from a particular place to another place on a particular date and seat class then booking a room in a particular hotel class at the arrival place, date, and time and hiring a particular car type on the arrival date and time is a complex problem requiring advanced coordination skills and capabilities.

The characteristics of the TAS scenario can be viewed via two systems development approaches; either an open, distributed heterogeneous environment system similar to the research aim or as a closed system similar to the existing methodologies development approach. Generally TAS has been applied as test bed in many agent-oriented and multi-agent systems software projects and research (Dignum et al., 2008).

3.3 Prometheus, MaSE and Gaia

Numerous methodologies for agent-oriented software development have been proposed in the literature. However, their application to real-world problems is still limited due to their lack of maturity. Evaluating their strengths and weaknesses is an important step towards developing better methodologies. This section presents research results obtained by applying three agent-oriented methodologies, namely Gaia, MaSE and Prometheus in the context of a Travel Agency System (TAS).

3.3.1 Selection of Prometheus, MaSE, and Gaia

The research selected the three methodologies; Prometheus Gaia, and MaSE to deploy a TAS case study for the following reasons. Basically, those methodologies have been tested over academic case study projects for building independent multi-agent systems. Each is supported by graphical software development tools which ease the development processes to check their syntax and consistency. In comparison with the other agent-based development process, Gaia, MaSE and Prometheus are available in the literature and their documentation is accessible, for example Prometheus is explained in a published text book. A survey on development and research into agent development tools found that these three methodologies are supported and undergoing research for further improvements where the others are ignored. These methodologies are rivals to DMMAS' hence their analysis and design correlated to the agents' problems domains of DMMAS. Finally, the approach that each methodology follows is unique and can be understood.

3.3.2 Prometheus

This section explains Prometheus analysis and design approach through a practical development experiment that has been conducted by this research. Prometheus development life cycle was applied to model a TAS case study.

Prometheus does not include an early requirement phase, but instead recommends using PASSI techniques. In fact there is no difference between the two approaches in the requirement analysis stage, therefore the functionality diagram illustrated by Figure 3.1 reflects the Prometheus specification analysis. The rectangles reflect the system functionality attached to the processes and events. The functionality diagram is the base used to decide on the system agent types.

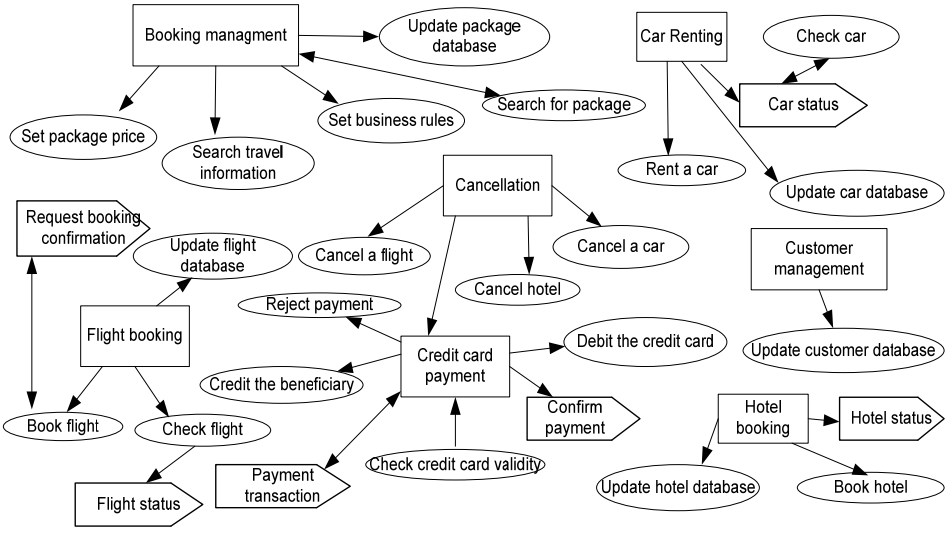

Figure 3.1: Prometheus functionality diagram for TAS.

Identifying the system agents' types is applied using grouping techniques where each group must consist of correlated collections of functionalities with high degree of dependency. Each functionality group together forms an independent system function or a unit of system capability. For example, in Figure 3.1, the hotel booking is one system capability representing one software agent. The next step is to transfer this manual diagram into the Prometheus software development tool (PDT). This starts to input the data coupling diagram illustrated in Figure 3.2. From this development stage until the detailed design process, Prometheus relies partially on PDT, which means

not all the development steps are automated and this immaturity is a drawback in Prometheus.

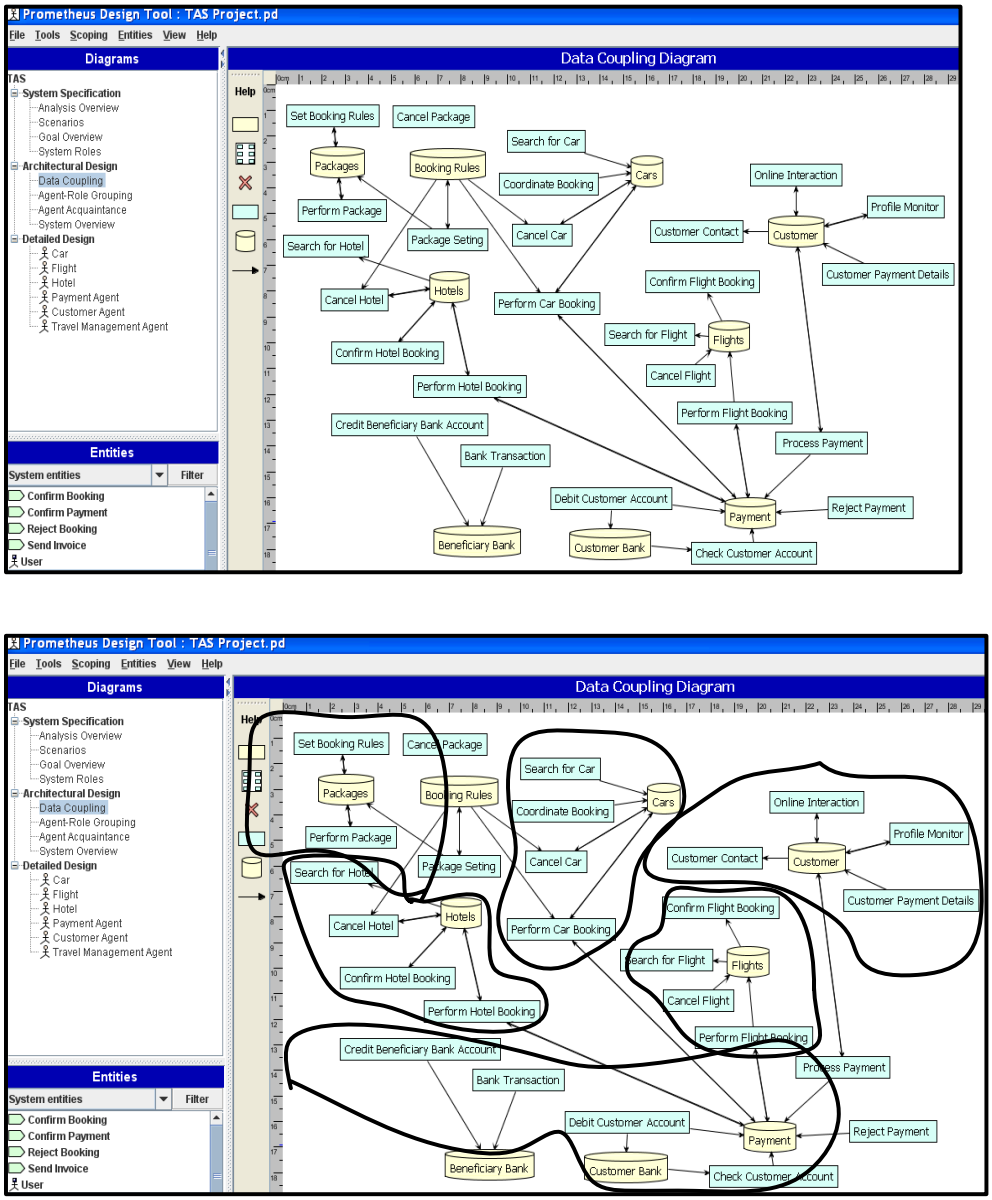

Figure 3.2: Prometheus data coupling.

According to the research experience with the Prometheus development approach, it focuses on the individual BDI agent structure. The agent team work (cooperation) process is totally dropped from the development process. The data coupling diagram (Figure 3.2) depicts that starting from the analysis process, the cooperation basic

elements are not grouped; indeed there is a separation for each agent. The research and part of its practical experiments had proposed to extend Prometheus to include an agent's cooperation model "*Enhancing Prometheus to Incorporate Agent cooperation Process*" (Alhashel et al., 2008). This experiment enhanced Prometheus to build better systems, but still there are fundamental changes required to transfer the focus from an independent to a cooperative multi-agent system.

To clarify the Prometheus independent agents design approach, we take a close look at Figure 3.3 which represents the *capability diagram* for the flight agent example. Although Prometheus focuses on the BDI agent structure, its agent's design does not include any knowledge domain or how intelligent agents will share their knowledge. Accessing the same database does not provide a semantic to the data in use, so there is no shared or common meaning to what this data refers to. The argument here is that if agents share the same database without an ontological domain model of that data, then its use is as value not as knowledge.

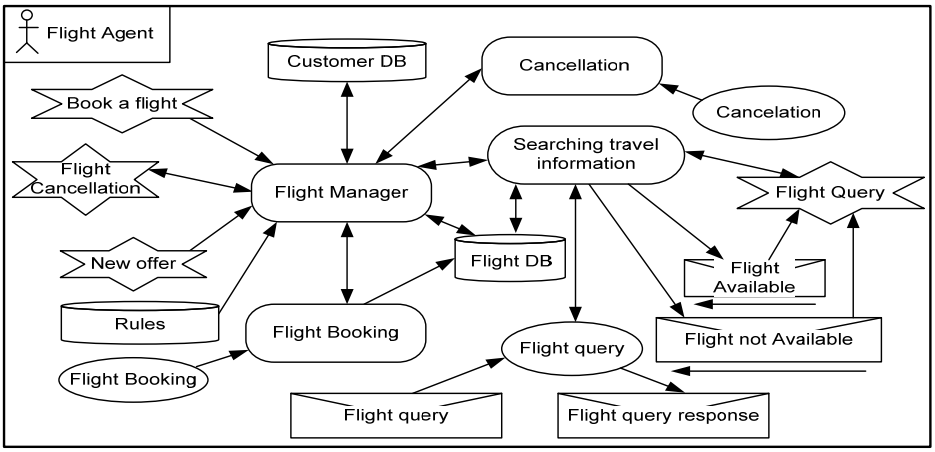

Figure 3.3: Prometheus capability diagram.

Figure 3.4 illustrates the TAS agent's interaction for "perform payment event" that is generated automatically from the PDT options menu. This automation facility adds advantage to the Prometheus development process. On the other hand it confuses the developer because, in the manual development process Prometheus applies AUML agent interaction diagram (Padgham and Michael, 2004). But PDT generates UML object sequence diagram. There is inconsistency between the automation and the manual practice. Another drawback, is that Prometheus does not include or provide any design model to represent agent autonomous behaviour either in the

implementation or in simple "accept/reject" model. This is a fundamental shortcoming in the Prometheus design approach and indicates that Prometheus design is a closed concealed multi-agent systems where agent behaviour is predefined and does not require autonomous behaviour.

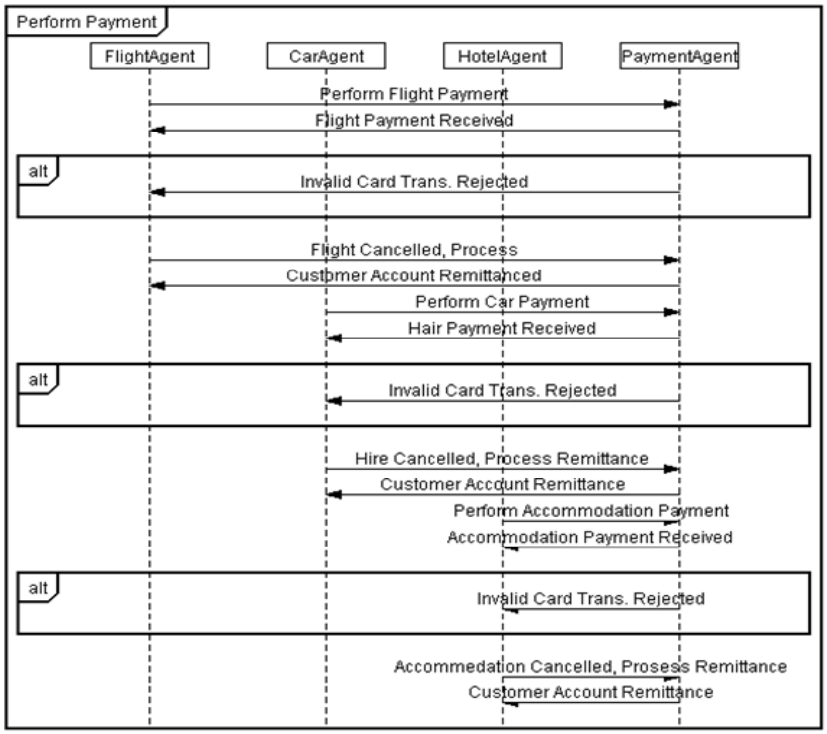

Figure 3.4: Prometheus interaction diagram.

The gap between the proposed methodology (DMMAS) aims and Prometheus can be interpreted through the Prometheus' system overview diagram which substantiates the following assessments:

1. There is no coordination or goal execution plan to coordinate between agents in order to achieve a common task cooperatively.
2. There is no option to add or delete an agent at run time.
3. There is no dynamic behaviour or agent autonomous behaviour.
4. Prometheus does not present any open architecture nor allow reorganizational structure to reflect with goal changes requirements.

Prometheus can successfully build a close independent multi-agent systems based on BDI agent's architecture. In contrast, Prometheus is not able to function in building open agent systems as it has fundamental limitations. Precisely Prometheus does not

address the concept and the abstractions of cooperative multi-agent systems that the research aims to achieve.

3.3.3 MaSE

The second experiment is to apply the Multi-agent System Engineering (MaSE) methodology to the TAS project then discuss and assess its capabilities. MaSE is a goal oriented methodology and presumes that every goal is achievable by a role and every role is played by at least one agent class. The relationship between the goal and role is one to one mapping, unless a many to one relationship is efficient for the nature of the system under development. However, in the initial requirement analysis the goals are extracted from the system scenario then directly set in hierarchal structures using the KAOS approach (Lamsweerde, 2001). MaSE uses case scenarios to represent goal and sub-goal along with UML sequence diagram in context of roles instead of objects. This process eventually results in capturing the events which define the communication between agents that will play these roles. After the roles have been defined, it will be decomposed into a set of tasks where each is designed to achieve the goals for which role is responsible. MaSE also provides a messaging format consisting of performative, activities, send-receive, and set. In general these are the basic principles of MaSE development steps.

Initially, a MaSE presumed agent does not necessarily possess intelligence. It may or may not exhibit an intelligent attitude. On the other hand the methodology did not describe how the intelligence could be captured or presented at the design stage so there is confusion at the implantation process, in case intelligence is required. In addition, Figure 3.5 illustrates the TAS role diagram but while the methodology documentation indicates to the agent autonomous behaviour, in contrast the analysis diagram does not show how hotel agent for example will provide its services to the other agents. MaSE relies on finite state automata messages-based exchanges rather than providing open linking architecture. For example, how do the three agents; hotel, car, and flight connect their services together using messaging techniques without a domain knowledge representation or language semantic model? The message format implied by MaSE is compliable, rigid and therefore an agent has no choice but either to perform or fail. This takes the design back to the object-oriented concept.

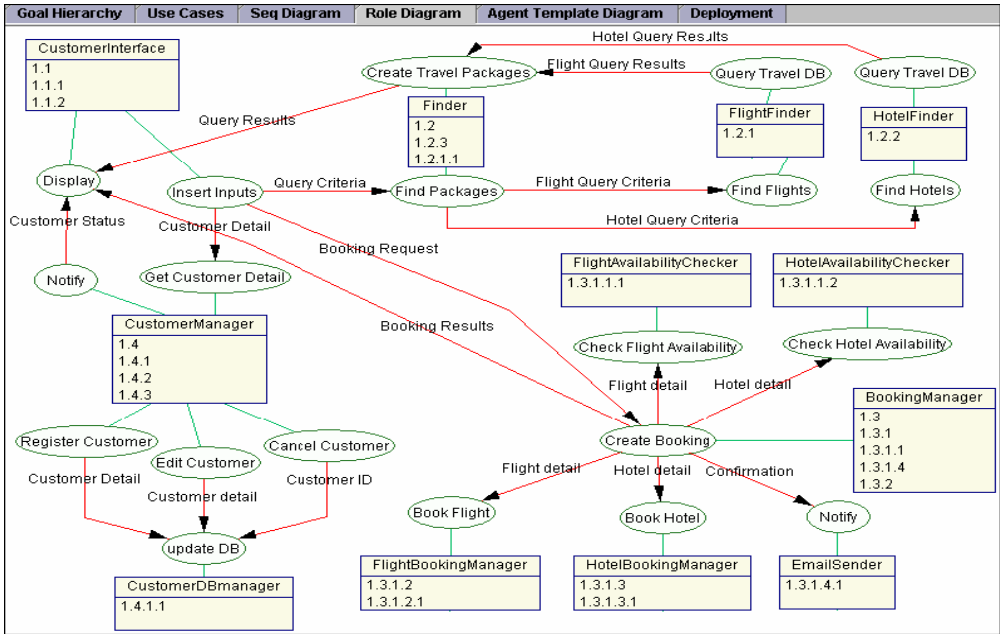

Figure 3.5: TAS role diagram produced during the analysis process.

The TAS role diagram (see Figure 3.5) shows that the role represented by a rectangle (class diagram) and the number reflect the goal to the sub-goal hierarchy while a role's tasks are represented by ovals attached to them by lines. The green lines represent the internal communication between initiator and respondent and the red line represents the external communication (role-to-role). But there is no indication for the services finder or search mechanism. The methodology does not include agent's services, for example in Figure 3.5, when the user books a complete package of hotel, flight and car, the reservation agent it is not clear how this agent will access or search for the required services, instead it calls the services by an agent name. This technique is efficient for a small limited closed system. There is confusion in mixing agent name and agent services. The services in this context mean the agent internal functionality. The particular or the unit of task that an agent can performed for the benefit of the system objective (Alhashel and Mohammadian, 2008).

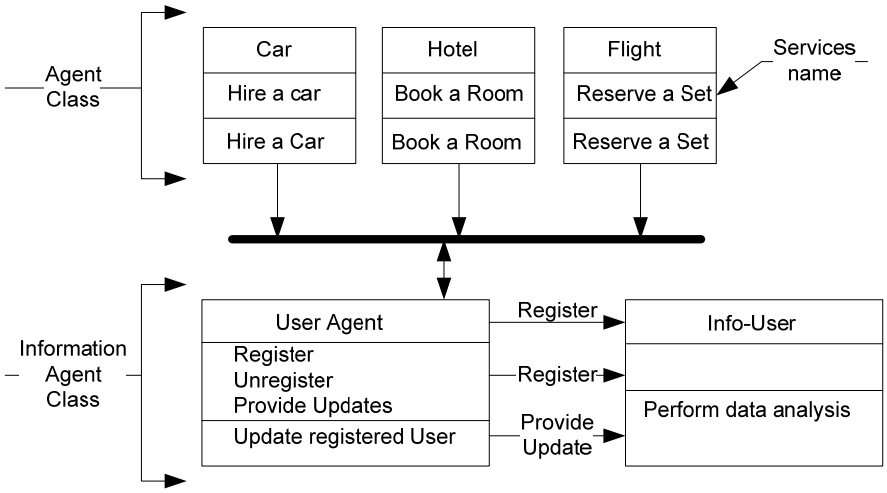

Figure 3.6: Agent class, agent information, and their conversation diagram.

Another observation from the TAS case study is when establishing the conversation session between agents presented in the communication class diagrams (see Figure 3.6) shows a control and handling of the messages beside that the messages are related to an action after it has been validated. However, is there any mutual understanding on the object discussed in those sessions? Is there any parsing to the messages language so that it makes meaningful terms (semantic) in the entire systems protocols, hereafter creating a common language vocabulary that can be used while the system grows up? Actually, MaS is able to successfully link TAS agent with their actions, roles and policies but in a hardwired predefined method.

The MaSE deployment diagrams used to define a system structure such as the agent numbers, types, and their location are based on the domain level, agent level, and component level design steps. This deployment diagram is hard to develop because the methodology does not explain or guide how to construct this diagram out of the other related levels. This limitation appears in the MaSE documentation “the step is not well describe or explain” (Deloach, 1999).

MaSE presents a good approach and contain real agent-oriented software abstractions. MaSE is also supported by a reasonable graphical software development tool (agentTool) that is able to simplify the development process. MaSE is powerful methodology in this stream of close independent agent-oriented system but when MaSE is evaluated by the criteria of open, distributed, dynamic software agent

environments MaSE does not demonstrate any dynamic actions at runtime. For example to add or remove an agent at system runtime, an agent adoption strategy, system restructuring, or a general agent services provider. These limitations conclude that MaSE is functional for closed multi-agent systems.

3.3.4 Gaia

The research deploys Gaia methods on the TAS case study to evaluate its development approach and techniques and furthermore, to assess its efficiency in designing open, cooperative multi-agent systems. The basic principle of Gaia is that the multi-agent systems is a society of organisations structure where every organisation has its role and policies and is represented by an agent or a group of agents. The operational result of each organisation role is governed to form the main system goal. Thomas et al. (2002) criticise the Gaia design approach and state eight weaknesses preventing Gaia systems from functioning in open system environments then propose to extend Gaia with Role Oriented Analysis and Design for Multi-Agent Programming (ROADMAP). The new merged model continues to be called Gaia or ROADMAP and some literature refers to as Gaia-ROADMAP (called Gaia). However, the extended Gaia provides four additional improvements;

- formal models of knowledge and the environment,
- role hierarchies,
- explicit representation of social structures relationships,
- incorporation of dynamic changes.

Gaia is supported by a software development tool called REBEL (Gaia Editor Built for Easy Development) which is designed to help the developer identify the goal models and the role models during the analysis stage (Pei Pei et al., 2005). However, the aim of the Gaia analysis phase is to collect then organize the system specifications and requirements for building the initial system organisations structures. In this way the analysis phase will develop the environment model, the preliminary roles, the interaction model, and the organisational rules for each of the sub-organisations that compose the entire system. The system organisational structure will be shaped on grouping techniques. Each highly cohesive interacted set of roles share the same

purpose or task and can be controlled by one unit of rules grouped together to form one organisation.

The analysis process of the TAS scenario delivers three organisational structures *car*, *hotel*, and *flight* located under the main organisation root TAS illustrated in Figure 3.7. In this approach Gaia isolates each organisation from the other and groups its agents and allows them to communicate through a predefined set of messages. Tveit, (2001) state that "*Due to the mentioned restrictions of Gaia, it is of less value in the open and unpredictable domain of internet applications, on the other hand it has been proven as a good approach for developing closed domain agent-system*".

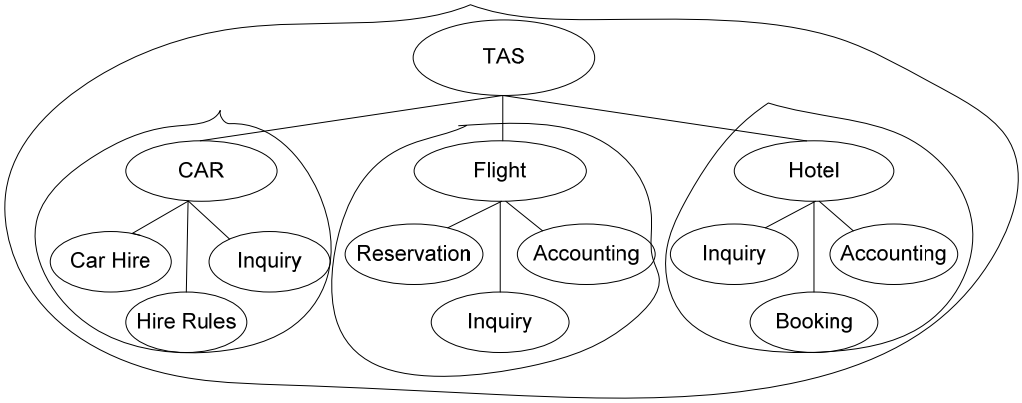

Figure 3.7: TAS organisations structure diagram.

The Gaia organisational structure approach was able to compose the system goal although it restricts cross organisational services (organisation to organisation services) relationships. For example, the TAS organisation structure diagram illustrated in Figure 3.7 implies that the car-agent is isolated from flight-agent and hotel-agent. In addition it is not possible for any agent to advertise its services (capabilities) to other agents. This facility is situated in the core of the open cooperative multi-agent systems concepts and without it agents lose one of the key advantages.

In Gaia techniques, the role capabilities are defined by permission and responsibilities. The permission attribute governs the runtime resources (environment) rules on the basis of the roles. It will manage the resources according to the roles eligibility for usage. The responsibilities attributes are aims of the role expected behaviour, by defining two properties; liveness and safety. The liveness property state

the positive changes carry out by the role and the safety property state the acceptable changes that are carry out by the role. Through the TAS design details phase for the role "book a room" (*BookRoom*) illustrated in Figure 3.8, there is no guideline to show how to state these attributes and how they will be used by the system agents. Theoretically, the definition of the role capabilities remains an open discussion as it is not clear how to develop the responsibilities and the permissions out of the complex large scale system. Furthermore, how will agent read these attributes, then based on what rules can the agent identify the role capabilities? The system artefacts generated by the Gaia methodology remains floated and difficult to convert into a computational software system.

Role Schema: BookRoom
Description: This preliminary role involves receiving request from customer through hotel agent, then after checking the availabilities and receiving the customer confirmation, the room is booked under customer details including from and to dates.
Protocols and Activities: CustomerRequest, CheckingAvailability, Confirmation
Permissions: Read Booking Request, Access Reservation Database // find all the available rooms Change Reservation Database // after booking conformation
Responsibilities Liveness: Booking = (ReceivConfirmation.UpdateReservationDatabase.PrintBookingData) Safety: • number-of-booking not > number-of-availabelRooms

Figure 3.8: TAS Role schema for BookRoom.

The protocol schema is the major process of the organizational rule development as it provides the execution plan. The protocol schema work as coordination processes between agents that operate within the same organizational boundary. Figure 3.9 depicts the preliminary protocol definition of Hotel organization for BookRoom role. Finding semiformal language used to define the organisational rule is an applicable solution to control the inter processing of roles and manage their execution sequences but there is also ambiguity in the development process about how to coordinate the

roles of organization *car* and the roles of organization *hotel* when a full travel package is the user option and there is contradictory availability at that particular booking date. Regardless of each organisation's rules, how is the most alternative choice to be generated as it happens in the real world travel business where they will suggest another nearest possible date. Furthermore, the experiment enlightens that it would be more efficient if there is a mechanism like the database journaling techniques. The transaction is first temporary reserved (hold) and when it satisfies the rule it commits or applies the transaction. If such a concept applied to Gaia organisation rules then it would enhance the operation. Actually this technique is considers for the research propose a methodology.

<table>
<tr><td colspan="2">Protocol Name
BookRoom</td><td></td></tr>
<tr><td>Initiator ??
(UserAgent or HotelAgent)</td><td>Partner
HotelAgent</td><td>Input
Check(InDate, outDate)</td></tr>
<tr><td colspan="2">Description When a the UserAgent or a HotelAgent requested to book a room, check the availabilities, send conformation then commit the booking, notify databaseAgent, else do not book.</td><td>Output
Book a Room
Or
Don't Book
Or
Cancel</td></tr>
</table>

Figure 3.9: TAS book a room protocol.

Applying Gaia on the TAS highlighted some limitations that also appeared in the Gaia documentation and survey. These are:

1. Does not follow particular modelling techniques for example, role, environment, and interactions are designed openly, without guidelines or standard modelling.
2. No implementation guidelines or approach.
3. No early requirement model to capture the system rules which play a main element in the Gaia analysis approach.

Generally, the case study shows noteworthy advantages in Gaia techniques. The approach of organisational structure introduces a new method into the software agent systems engineering. The organization rules model and the semiformal language for organization rules design, then identifying the role and defining its capabilities using permissions and responsibilities attributes is a unique approach that distinguishes Gaia

from other methods. Gaia adds new value to the closed multi-agent systems software engineering but when Gaia is evaluated for open cooperative MaS, it is inefficient "*Gaia methodology suitable for analysis and design of close MaS and adopting nonstandard techniques*" (Cernuzzi et al., 2004).

3.4 Comparison of Results

The purpose of this section is to present the TAS project findings and review the strengths and weaknesses of Prometheus, MaSE, and Gaia development methodologies. This evaluation provides a deeper understanding of these methodologies' approach and techniques and consequently establish a research background for DMMAS. To date, there is no standard evaluation schema that can be implemented directly to measure the methodologies capabilities (Bresciani et al., 2004). The research combined various agent-oriented software engineering evaluation proposals and extended them to serve the section's purpose. The comparison's criteria assess the agent concepts, development phases, artefacts and models, modelling notations, nature of applications, and development tools. Finally, these comparison results are summarised in the strengths and weaknesses schema.

Phases	**Gaia**	**MaSE**	**Prometheus**
System specification	Detailed	Medium	Detailed
Analysis	Detailed	Detailed	Detailed
Detailed design	Abstract, high-level	Detailed	Detailed
Architectural design	Not exists (Architecture)	Not exists (Architecture)	Detailed (BDI agents)

Table 3.1: Development phase details assessment.

Concept	**Gaia**	**MaSE**	**Prometheus**
Autonomy	Low	Low	Poor
Mental attitudes	Uses knowledge schema.	Agents do not have to be intelligent	BDI Mantel attitude.

Table 3.2: Presents the measure of an agent concept that each methodology supports.

Criteria	Gaia	MaSE	Prometheus
Clear notation	Strongly agree	Strongly agree	Strongly agree
Ease of learning	Strongly agree	Strongly agree	Agree
Ease of use	Disagree	Disagree	Agree
Adoptability	Agree	Agree	Agree
Traceability	Agree	Agree	Agree
Consistency	Agree	Agree	Disagree
Refinement	Disagree	Disagree	Agree
Scalability	Disagree	Disagree	Disagree
Concept overload	Agree	Agree	Disagree

Table 3.3: Shows the scale of the modelling criteria within each methodology.

Property	Gaia	MaSE	Prometheus
Openness	High	Low	Medium
Environment	High	Medium	Medium
Abstraction	High	High	High
Traceability	High	High	High
Modelling Transition	Medium	High	High
Complexity	Low	Low	Medium
Ease of use	Low	Low	Medium
Language adequacy	Low	Medium	High
Reusability	High	Medium	Medium

Table 3.4:Compares the properties of the methodologies.

Phases	Gaia	MaSE	Prometheus
System Specification	Object-oriented Use-Cases	Object-oriented Use-Cases	Stakeholders Scenarios diagram
Analysis	Environment, Knowledge, Goal, Role, Revised role model, Social model	Goal hierarchy, Sequence, Concurrent task diagram	Goal overview, Role, Data coupling diagrams
Architectural Design	Organisational, Agent Service, Acquaintance model	Agent classes, Conversations	Agent acquaintance, System Overview Agent Descriptors Protocols
Detailed Design	Organisational Rules, Role permissions and responsibilities	Agent's internal architectures, Deployment diagram	Process , Agent Overview Diagrams, and Capacity, Capability overview, Event, Data, and Plan

Table 3.5: Illustrates the available activities in each development phase.

Methodology	Application
Gaia	Coarse-grained computational, complex, open systems
MaSE	Heterogeneous multi agent systems
Prometheus	Intelligent (BDI) agents' systems

Table 3.6: Types of system domain for each methodology.

Methodology	Development Tool
Gaia	REBEL is a tool for building Goal Models and Role Models during the analysis stage. The research was not able to practice because not available.
MaSE	AgentTool, able to do printing, verification on developing system, and generating a skeleton code in java.
Prometheus	Prometheus Design Tool (PDT), which is able to do cross checking, saving diagrams as pictures, JDE (JACK Development Environment) generates JACK code.

Table 3.7: Software development tools support.

The comparisons demonstrate the capabilities of Prometheus, MaSE and Gaia giving an overview of how these methodologies handle the analysis and design of multi-agent systems, and summarise their strengths and weaknesses in comparison with each other for closed MaS modelling. For further investigation of the existing methodologies refer to Luck et al. (2004), and Hederson-Sellers and Giorgini (2005). In addition, Tveit (2001) and Stum and Shehory (2004) presented an additional survey of these methodologies based on their proposed evaluation frameworks. Iglesias et al. (1999), and Sudeikat et al. (2005) are also evaluated the existing development technology focusing on the runtime platforms and development tools.

3.5 Inadequacies in Existing Development Methodologies

Table 3.8 demonstrates the weaknesses of three main methodologies in developing open cooperative multi-agent systems that function in a distributed heterogeneous software environment. The key issues in developing any open multi-agent systems are;

- to design a flexible interchangeable system structure that has the ability to reconfigure itself at runtime,

- provides agents selection mechanism from a vast range of agents each possessing different capability (services),
- provides coordination or an execution plan model to allow agents working together to achieve a common goal.

The above three issues are the key components of any open multi-agent systems, however these key components are not addressed by any existing methodology. In the TAS research experience Gaia develops independent organisational structure isolated from each others form rigid system architecture without any cross relationship between agents. The three agents (*flight, hotel,* and *car*) are designed as independent three organisations of the same level leaf node under the main root TAS without any agent cooperation or agent team formation process. Prometheus delivers an independent TAS where each system agent (*flight-agent, hotel-agent,* and *car-agent*) function independently without any cooperation mechanism or autonomous behaviour. Instead Prometheus relies on an individual BDI agent structure, and agent working in a group (cooperation) is not addressed entirely. MaSE is not able to address the open environment for agent autonomous behaviour and does not address any open linking architecture to manage the TAS agents cooperation process. MaSE analyses and designs TAS (*flight, hotel,* and *car*) as three independent agent classes.

The three methodologies, Prometheus, Gaia, and MaSE, each follow its own unique development approach and aim to design closed multi-agent systems focusing on particular agent architecture and yet have not addressed the open multi-agent systems development method or modelling techniques. The software autonomous agent behaviour functions in a cooperative mode, bringing new challenges to software engineering. An open multi-agent systems is characterised by a coherent software agent, configurable at runtime, scalable, functioning in an open distributed environment and its agent can be developed independently and supported by an open search mechanism. The existing development methodologies fail to addresses these specifications in contrast with the aim and objective of this research.

	Existing Methodologies			Weaknesses
Criteria	**Gaia**	**Prometheus**	**MaSE**	
Goal oriented	In form of role	Goal diagram	Goal hierarchy diagram	Gaia and Prometheus has no explicitly deal with system requirement activities
task/subtask	Not specified	Included in the Functionality model	Within extended role diagram	Not necessary task defined directly
Role model	Role model	Within agent capability	Role diagram	Interchangeable but all include the basic abstractions
Agent architecture	Any (independent)	BDI	Independent (intelligent and non intelligent)	Independent architecture is efficient approach but it must be govern by service model
Team formation process	Not specified	Not specified	Not specified	Main weaknesses
Dynamic behaviour	Not specified	Not specified	Not specified	Main weaknesses
Cooperation mechanism	Interaction protocol within agent organisation only	Not specified	Goal base on finite state	Basic within independent close system
System Architecture	Independent organisational structure	Focus on independent agent	Independent organisational structure	Fix rigid no dynamic aspects
Restructuring facility	Not specified	Not specify	Not specify	Thus design a rigid system
Distribution agents model	Not specified	Not specified	Not specified	All follow a local agent
Agent services search mechanism	Not specified	Not specified	Agent class descriptor	MaSE claim it is open methodology and search can be attached as part of its modelling. However, all have not state search model
Any open techniques	Focus on independent organisation structure	One system environment similar to Object-based	Goal and finite state automata state transition for agent intra communication	The openness is base on local one unit system
Autonomous behaviour	Not specified	Not specified	Not specified	Local agents operate inside one system boundary

Table 3.8: Inadequacies in methodologies with respect to the TAS case study.

3.6 Summary

This chapter sets up the research background of examining related work. It overview a TAS case study as a practical experiment in multi-agent systems development project. The chapter is organised into three steps: first step studied and investigated the existing rival three multi-agent systems development methodologies, Prometheus, MaSE, and Gaia then evaluated and compared their analysis and design techniques. The second step revealed these methodologies' strengths and weaknesses based on open MaS requirements, components and criteria consisting of; agent autonomous and properties, cooperation constraints, development processing, application domain and open search mechanism. The third step summarised the TAS experiment finding and described the research main argument which the existing development methodologies failed to address, i.e. the analysis and design components to develop open cooperative

multi-agent systems. These methodologies failed under closed independent multi-agent systems.

The next chapter propose a new original development methodology that addresses the gap identified in this chapter. The proposed methodology has the potential to guide the developer to design open cooperative multi-agent systems that operate in a distributed, heterogeneous environment.

Chapter 4 DMMAS: A Novel Development Methodology for Multi-agent Systems

4.1 Introduction

This chapter presents a novel software engineering development methodology for multi-agent systems (DMMAS). DMMAS consists of four main development processes i.e. initial system requirements, system analysis, architecture design and detailed design. The primary aim of DMMAS methodology is to present the development techniques in sequential steps that lead to develop MaS which has the potentiality to operate in open distributed multi-agent systems. By building DMMAS the research intended to close the gap in the existing multi-agent systems software engineering development methodologies that focus on the closed MaS.

The chapter elaborates the potential of analysing and architecting open cooperative multi-agent systems in terms of dynamic organisational bases on agent adoption other agents to achieve the assigned goal. The chapter explains DMMAS development phases in detail using new diagram and graphical notations. The methodology concepts are illustrated through examples.

4.2 DMMAS Criteria

This section lists six criteria considered in DMMAS design processes. These criteria are to enhance the deployment process and to minimise errors and to maximise consistency throughout the development process. These six criteria are:

1. The methodology should be easy to use and easy to learn. Using clear and uncomplicated notation and avoiding the formal specification as possible will make it more straightforward and available for the domain expert and developer. Although AbS is hard to design, nevertheless, DMMAS has been designed to minimise the effort required to produce the required application.
2. The methodology must be complemented by clear documentation to guide the developer from the initial requirements step throughout the detailed design step. However, for successful implementation the methodology must have the potential to define the paradigm implementation elements. For example, object-oriented

implementation language elements class, object, method, attribute, and relationship. Therefore any object-oriented development methodology must be able to design the system in the programming concepts and deliver these implementation elements. Multi-agent systems do not have implementation concepts or programming language. For this reason DMMAS propose new MaS concepts based on *goal-schema*, *professional-agent*, *skill-agent*, *execution-plan*, and *functionality-schema*.

3. Utilising the best existing practice of agent-oriented software engineering, knowledge engineering, and object-oriented software engineering (OOSE) or any other software development paradigm wherever applicable. This strategy is to benefit DMMAS by: (a) being unnecessary to reinvent the wheel where there is a readymade device applicable for users. (b) The UML, Object Modelling Techniques (OMT), AUML, and URP each contain various will tested models techniques, abstractions and notations for analysis and design software applications developments process that can benefit the propos method. (c) Odell (2003) suggested that for successful industrial deployment of agent technology, present the new technology as an incremental extension of known and trusted methods (Odell et al., 2003).
4. Applying the agent-based FIPA standards wherever applicable, to adjust the final product with the existing technology and make it integrated with future developments. In addition, the research forward DMMAS to gain agent's community acceptance then contribute to the agent-based software standardisation.
5. The characteristics of multi-agent systems are unique and belong to next generation software (Alessandro et al., 2002). Therefore leaving some provision for the future modification will provide popularity and maintainability for DMMAS.

Meeting these criteria would enhance DMMAS as a methodology with high software engineering quality.

Terminology Used: The following terminology is used for DMMAS design methodology.

Goal-schema: Is the user goals defined in XML elements hierarchal structure.
Skill-agent: is an agent capability or particular functionality that form part of the goal achievement requirement.
Professional-agent: Is an agent expert in particular application domain responsible to achieve the user goal in this domain by adopting the skill agents required according to the goal execution plan.
Functionality: define the *skill-agent* functionalities what it does or what is its capabilities in one descriptive term or label for example hotel-booking, or flight-reservation.
Execution plan: the execution prescription for particular user goal in the system. The execution plan provides the professional agent with information on how to execute the goal.

4.3 DMMAS Applications Domains

The definition of an agent is intended to be general. In addition system behaviours encompass flexible features and, for this reason are viewed as universal software concept solutions. There is a misunderstanding in mapping the agent-oriented system concept with the appropriate application domain (Alhashel and Mohammadian, 2008, d'Inverno and Luck, 2004). There is a plethora applying agent-oriented system into a wide range of taxonomy applications, for example, robotics, computer games, human simulation, biometrics, web services, business applications, e-commerce, aerospace control system and the like. Scott et al (2001) listed five guidelines to decide whether an agent-oriented system is more suited:

- in a situation where complex/diverse types of communication are required,
- when the system must perform well in situations where it is not practical/possible to specify its behaviour on a case-by-case basis,
- if the application domain involves negotiation, cooperation and competition among different entities,
- when the required system needs to act autonomously,
- when anticipated that the system is expandable and changeable (Malley and DeLoach, 2001).

In such a wide range of different software approaches and application domains, it is important to specify DMMAS application domains, otherwise designing universal agent-based development methodology will result in complicated and hard to use methodologies (Jim et al., 2005). DMMAS methodology is suitable to designed multi-agent systems in the domain of multi applications distribution environment including military logistic, healthcare, transportation, and travel agencies systems. DMMAS can be used to integrate the services of open distributed heterogeneous systems or subsystems to benefit the system user interface to deal with different collections as one system.

DMMAS Problem Characteristics: In large scale services or business organisations software applications are developed on different locations at different times and using different skills. DMMAS problem characteristics can be described when the required system specified is a collection of heterogeneous subsystems operated in an open distributed environment the system main goals embrace a set of constraints and the solution is scattered over these individual subsystems. Such problem characteristics required a course of coordination between these subsystems and another coordination course to resolve and deliver the goal constraints. For example if the problem is a travel agency system consisting of multiple subsystems (hotel booking, bus reservation, and flight reservation) then coordination between these system within the user preferences for framework of date, time, place, class, and price is a complicated coordination problem. DMMAS can be deployed efficiently to integrate these systems to function as one homogeneous application. DMMAS resolve such problems using agent's cooperation concept base on *professional-agents* and *skill-agents* adoption techniques, execution plan for system coordination and ontology framework for subsystem functionality.

Figure 4.1 illustrates the role of DMMAS to integrate various standalone systems, then to generate the required multi-agent systems then reflecting the result on the user interface as one functional system. Systems A, B, C and D are each represents a logical application with its own goal and constraints where the user interacts with system as a single system to solve his objective. However, integrating such independent systems in one software environment is integral to the core concept of DMMAS analysis and design.

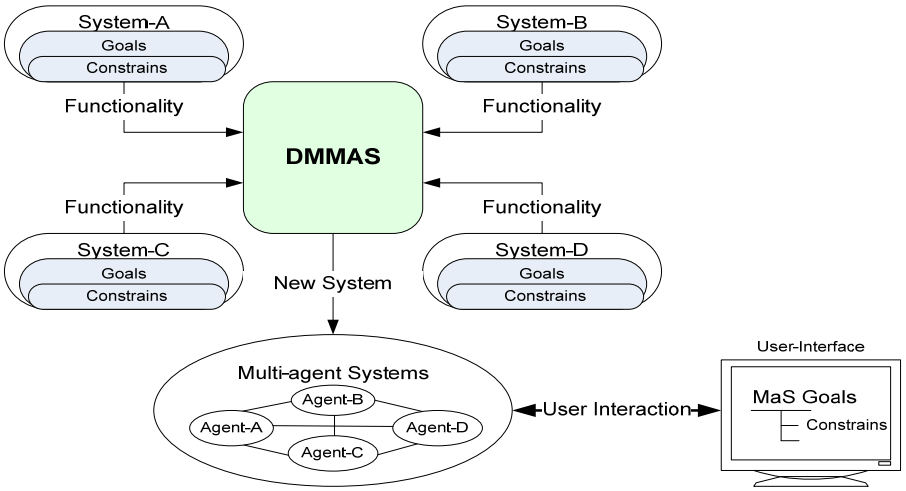

Figure 4.1: DMMAS problem characteristics.

4.4 DMMAS Architecture

The DMMAS methodology consists of four main development phases; requirements analysis, system specification analysis, architecture design, and detailed design. Figure 4.2 illustrates DMMAS architecture based on the input and output artefacts at each development phase. DMMAS is goal oriented methodology where the requirement analysis refines the system scenario to identify the system goals and sub-goals. Based on the system goals and the requirements captured, the development process starts as shown in Figure 4.2. In the detailed design there are three implementation models: system goals XML schema, SQL search statements, and execution plan database model. The implementation phase is divided into two steps and the three schemas complement the development process. The other system components are outside the scope of the current work.

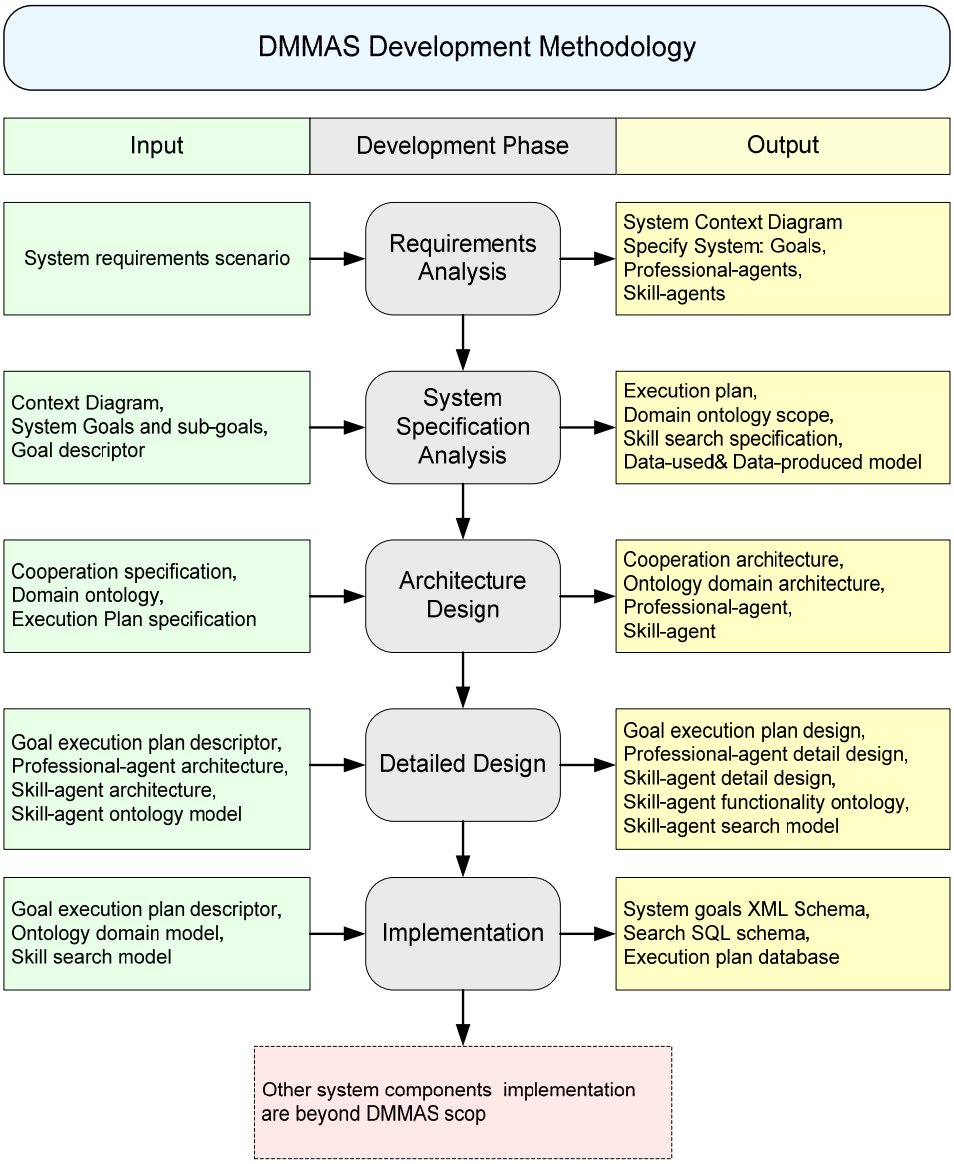

Figure 4.2: DMMAS Architecture.

The DMMAS design of multi-agent systems relies on ontology based search model to allocate the skills required for the goal achievement. Incorporating ontology into software engineering development methodology is innovative and entails two software approaches to merge together, ontology engineering and software engineering. Figure 4.3 shows DMMAS development methodology encompassing both traditional software engineering and the ontology engineering. It lists the components involved.

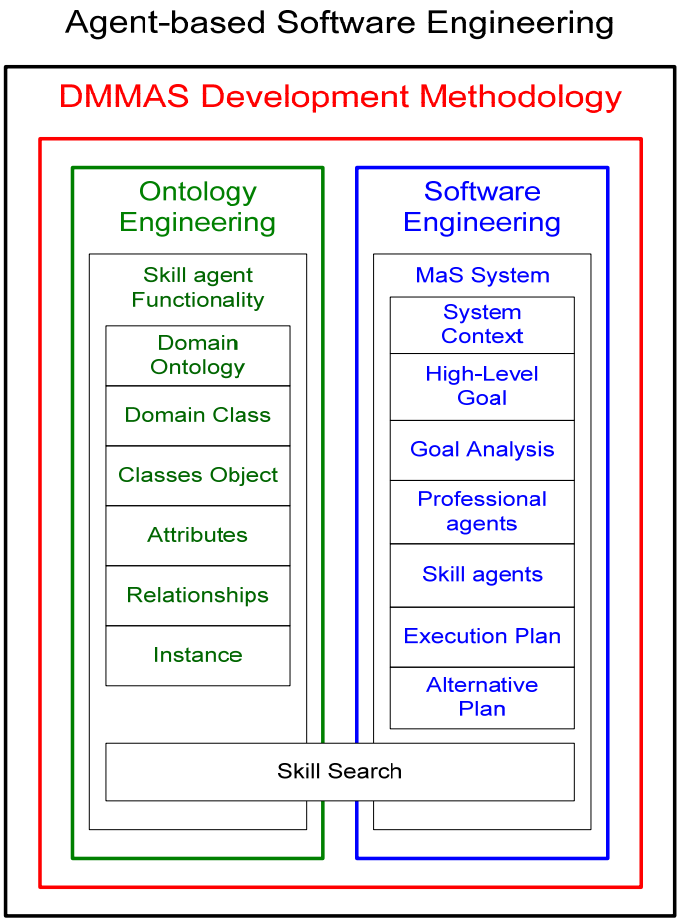

Figure 4.3: DMMAS architecture in context of components.

There are three fundamental techniques that distinguish DMMAS from all the other existing multi-agent systems development methodologies. First, DMMAS incorporates an ontology approach establishing a knowledge-based search model. This is to replace the existing text keyword based search. This technique allows the system to find the agents providing particular services (skill) to achieve a particular goal automatically out of a large set of software agents situated in an open distributed environment. The purpose of introducing the ontology schema for the system services domain is to devise a solution to the existing static multi-agent system that exists in current agent-based development methodologies. The services in this context mean the functionalities that are provided by the system and the tasks that agent is equipped to perform.

Second, a system structure is built based on an organisational structure comprising *professional-agents* in the root node and *skill-agents* in the leaf node. This structuring

avoids any direct prior physical link between the agents team members (*skill-agent*) and the team manager (*professional-agent*) allowing the model to provide a dynamic selection of the *skill-agent*s that match the goal requirements at runtime.

Third, DMMAS builds MaS with a capability for dynamic organisational structure. The organisational structure relies on two concepts, the *professional-agent* and the *skill-agent*. The *professional-agent* is the agent responsible for goal achievement by adopting the suitable *skill-agent* and supplying the plan recipient to the selected *skill-agent*. The *skill-agent* is the logical unit having the potential to perform the task allocated, following a plan managed by the *professional-agent*. When the job in hand is done the result is reflected through the *professional-agent* to the user. The *skill-agent* will not engage with more than one *professional-agent* at a time.

Figure 4.4 illustrates DMMAS development phases in relation to the system components including the design verification component. In the system requirements process both the system external entities and the system goals will be identified. For this task DMMAS propose a goal diagram to represent both the system goals and system external entities, for example stakeholder, software program, or any other external environment with interacts with the system. The outputs of this stage are context diagram, stakeholder, and system goals diagrams.

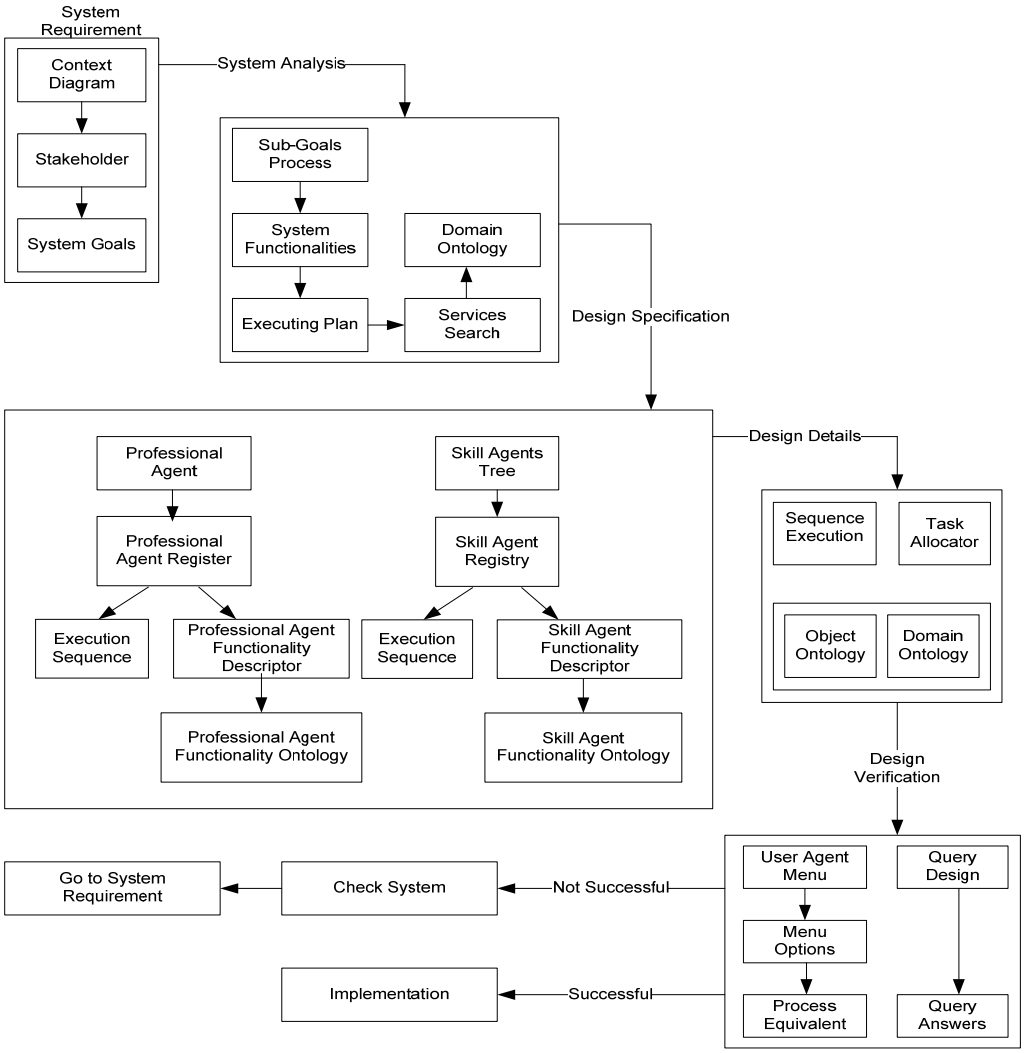

Figure 4.4: DMMAS development phases and system components.

The system analysis phase extends the system requirement to the system specifications. In this step the system is broken down into main components that play main roles in the system internal functionalities. The system goals are analysed further with respect to the system functionalities and the goal execution plan is established. Both the ontology domain model identifies in parallel with the services search strategy. This equips the system with a suitable ontology domain and decides on the ontology search model.

The architecture design develops the system further to specify the professional and *skill-agent* structures. In this step the system architecture is defined with the relevant descriptors, for example, the registry that will hold the agent identification, the plan of the individual *professional-agents* and their relative *skill-agent* functionalities and

identifications. The design specification puts the ontology map for both the *professional-agents* and the *skill-agent* in terms of the interaction protocols used and the objects that share the organisation structure.
The detailed design connects all the system components to finalise the system details and make it ready form for the implementation phase. The overall system execution plans the task allocated registry, the object ontology, and the domain ontology. These components are finalised and developed in a format ready for implementation. The system implementation artefacts that structure the *professional-agent* and the *skill-agent* are set in the last stage. These include the system goals, services ontology, search schema, functionalities, system sub-goals interaction domain protocols and the roles that allocated to each *skill-agent* are created.

To improve the design process DMMAS introduces a design verification model. This model is to verify the design with the system development phases before the system transfer to the implementation stage. The verification process uses a verification form to check the design artefacts and abstractions against the system requirements. If it does not verify the system goals then it loops back to the development process starting from the system requirement stage (based on iteration development techniques). The next section explains DMMAS development techniques in details.

4.5 System Requirements

The first phase in DMMAS methodology is the system requirements. This phase is divided into two parts: part one - concern with the high level system requirements (basic blocks) to identify and capture the system *stakeholder*, system high level *goals* and then draw the system *context diagram*. In this stage, the system appears as a high level black box with no need for system functionalities or details on how the system will work. However, the first part will study the system scenario (system document or user story) carefully and identify and develop system high level requirements. The second part – analyses the system goals that are captured in part one and refines these goals to specify the implicit goals, sub-goals, and alternatives goals along with the initial rules.

4.5.1 System Requirements: High-level

The system stakeholder: Any external entity that interacts or interfaces with the system, for example system user, computer system, internet application, or any other beneficiary. This step helps to specify the system external interactions requirements and system interface requirements. The actors mostly exist as nouns in the system main scenario, for example the user, bank, and customer.

The system goals: These are the objectives that the user is expecting the system to achieve on his behalf. Considering that the agent-based system is goal oriented identifying the system goals at an early stage is prioritized. In software requirements, engineering goals are divided into two types: functional, which are concerned with services to be provided by the system, and non-functional which are related to system qualities for example, security, response time, and training. At this stage of the requirements specification, the focus is on the system functional goals only.

There are several approaches for capturing the functional goals, but the most systematically effective method is proposed by Lamsweerde (2001) and Bolchini and Paolini (2004). According to Lamsweerde (2001), goals are identified from the first reading of the available source by searching for intentional key-words such as “objective”, “purpose”, “intent”, “concern”, “in order to”, and so on. McBreen also states that in the system scenario the goal itself is phrased with an active verb first: for example, “Customer: place order”, Clerk: reorder stock” (McBreen, 1998). This analysis stream identifies the system objectives of a stakeholder or the user requirement in terms of the system automation activities, and the expectation outcomes. All these are additional information in capturing the system goals. The most important output of this process is to develop a goal scenario for every individual high level goal. The goal scenario is a key technique in DMMAS for initial requirements.

As the system goals are captured and from the system requirement engineering point of view, the system goals are validated with the stakeholders to confirm that there are no ambiguities in understanding the system objectives. Reaching this stage, the actors

and the goals diagram is established using DMMAS goal diagram illustrated in Figure 4.5.

The system context diagram: In traditional software engineering the system context comes in the initial system requirement process to clarify the system's scope. It determines what is within the system development process and what is excluded. In addition, the context diagram reveals the external interactions objects with the system. The system context diagram is an important analysis activity for agent-based software engineering (to determine the agent operational environment, which is one of the fundamental interaction senses that the software agent acts with). The context diagram has another role for agent-based system analysis. It identifies the external sensor type that could affect the agent behaviours, since agents are autonomous, proactive, and effectors to the changes in their operational environment. In Figure 4.5, the dotted line shows the range or the scope of the system under construction with elaboration on what is inside and what is outside the system.

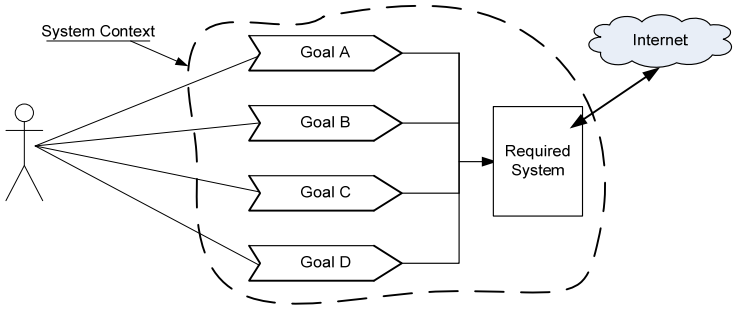

Figure 4.5: System context diagram including stakeholder and system goals.

4.5.2 System Requirement: Goals Analysis

This section complements the high-level requirement by developing the goal and sub-goals hierarchy tree, goals initial rules and goal descriptors. To simplify the steps explanation "Achieving PhD" example will be deployed throughout this Chapter.

Example: Achieving PhD scenario

Assume that a person is interested in achieving a PhD academic degree in a particular research field then there is a list of steps that must be achieved until all the requirements are accomplished and the degree obtained. For this example we assume

the process starts with searching for the preference course and educational institute. Then the person submits an application with evidence of the qualifications already obtained to the admission office. The assessment has process started and result informed. If the applicant is accepted then the enrolment process runs until the commencement date and student number is released. The student then contacts the assigned supervisor and a first meeting is arranged. The student understands the requirement and begins his/her research until the outcomes are achieved and approved. The student starts writing a thesis to document the research course work and results are obtained. Finally this thesis passes through the assessment process and recommendations provided.

Sub-goals: The goals captured in the earlier section are at a high level and difficult to achieve. Therefore it is necessary to break down each goal into more specific tasks or sub-goals. Sub-goals are goals within a goal (nested goals). Every high level goal encompass one or more sub-goals. Van Lemsweer's (2001), method provides an efficient approach to analyse the goal into sub-goals. The method suggests going through each goal scenario and applying "Why" and "How" refinement techniques. In this way, at every refinement cycle each system goal will link with a set of sub-goals. The Why-refinement is concerned with capturing the implicit goals. For example, the WHY question about the goal "Achieve PhD degree" results in extending the goals tree, as shown in Figure 4.6. On the other hand, How-refinement is concerned with capturing the goals assignment (task) that needs to be achieved. Using the same example, a HOW question about the same goal in Figure 4.7 provides the procedure to achieve that goal.

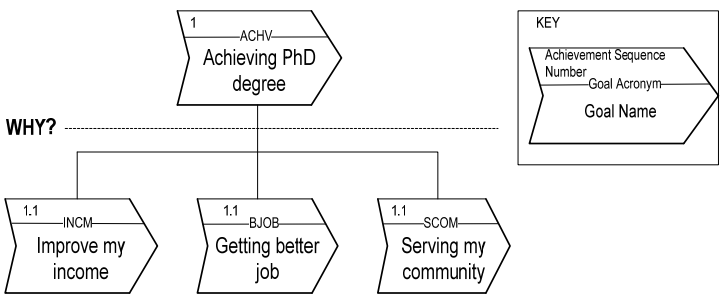

Figure 4.6: Why- Refinement extend the main goal assignment.

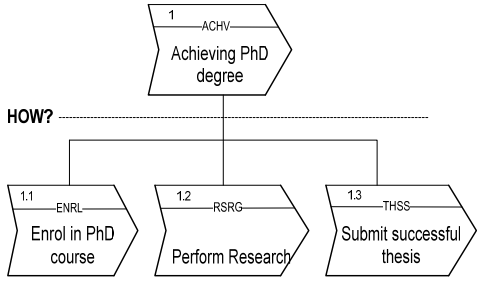

Figure 4.7: How-refinement extends the main goal object.

Whenever a goal or sub-goal is identified it must be labelled by a unique label of not more than four letters that can be readable. This forms the first part of the goal identification (GId) as it appears in the diagrams depicted in Figures 4.6 and 4.7. However, the refinement process continues and at each refinement cycle a new link is created. This process continues until assignable sub-goals are reached or there is no further expected level. In general and for less complication the advice is try to keep the number of sub-goals under super ordinate goal to a small number between three to ten (Stanton et al., 2005).

Goal achievement: After identifying the goals, and sub goals the next step is to allocate goal achievement cycle. This process prioritises the goal achievement sequence or order (at which stage the goal can be achieved in relation with other goals in the system). The achievement sequence is finalised by finding the goals processing relationships for example (as in Figure 4.6), if assignment sub-goal "Perform research" cannot be achieved without sub-goal "Enrol in PhD course" then in this case, perform research goal labelled by priority one in line with the main goal "Achieve PhD degree" achievement priority number. In case there is no goal achievement relationship, all the goals will have the same priority.

Notice that the achievement sequence number has a hierarchal structure and is illustrated by the point between the numbers represented in the hierarchal structure where the root goal starts with "1" and the next node in the tree will take the sequence "1.2" where "2" is the achievement sequence in the same node. If there is a third node level then the number will be "1.2.3". Figure 4.6 above shows the achievement sequence number directly above the line dividing the goal notation where the goal

acronym name appears in the middle of the line. There will be additional explanation on the use of these attributes in the analysis and design phase. The achievement relationship can be straightforward and the technique to follow is simply draw a goal dependences hierarchy.

Goal option and action: At this stage of the system requirements the system goals and sub-goals are identified and labelled by the achievement sequence numbers. The next step is to activate and deactivate each goal in the goals' tree by asking "what is the alternative procedure in case of goal failures". This is to establish a goal alternatives plan. The aim is to build a reliable system that can operate in open environment where the alternative plan is useful for any recovery process and to prevent the system from total failures. The alternative plan consists of options and actions. The option describes the failure condition and the option describes the alternative process. The goal descriptor depicted in Figure 4.8 provides additional explanation to the goal recovery process under the attributes option and action.

Goal descriptor: The last step in this phase is to develop the goals descriptors shown in Figure 4.8. This process is obvious and can be done straight forward in the following order:

1. Goal identification number (Goal Id): consists of two parts; the goal-label and the goal-achievement sequence number described in goal achievement, for example the Goal Id for "Enrol in PhD course" is "ENRL-1.1".
2. Goal name: as it appears in the goal structure tree, for example "Enrol in PhD Course"
3. Descriptions: a brief description on the goal, for example "The user approaches the system to enrol himself in a PhD course equivalent to his preference".
4. Verification: what are the tasks required to achieve this goal. In fact it is the how-refinement's leaf goals level to satisfy the node goal. For example verification for the goal "Achieve PhD degree" is Enrol in PhD Course, Perform Research, and Submit Successful thesis.
5. Data-Used and Data-Produced: will be explained in the analysis phase.

6. Option and Action: are the goal failure recovery plan describe earlier. For example, in case of goal “Enrol in PhD Course” option: No admission, action: send message “course not available”.
7. Priority: is the goal achievement sequence.

In this sequence all the attributes in the goal descriptor template (Figure 4.8) are supplied and in case there are more options and actions the descriptor can be extended.

Goal-Descriptor			
Goal Id		Priority	
Goal Name			
Description			
Verification			
Data-Used			
Data-Produced			
Option-1			
Action-1			
Option-2			
Action-2			
Option-3			
Action-3			

Figure 4.8: Goal descriptor template.

The requirement specification phase focused on the system initial document to capture the system high-level goals then specified the system context diagram. In this phase the goals hierarchal tree and the goal attributes are identified then finally the goal descriptor is established. The system requirements artefacts will be used as input in the next section to establish the system specification.

4.6 System Specification

The purpose of the system specification phase is to analyse system requirements and then construct the system functionalities specification. In this phase the system is studied and the system functionality and system concepts are built (John et al., 2007). The system specification phase starts by analysing and understanding the goals tree that was constructed earlier in the requirements phase. Then the system functionalities and operational strategy are set up.

According to DMMAS system specification, the system organisational structure defined by *professional-agent* and *skill-agent* is set up as well as the system functionalities in terms of components relationship, interaction protocols, execution plan, and the services search model is analysed. The system specification phase also provides and explains a new technique to identify the system domain ontology then set up the basic blocks for the ontology domain model. The outcome of this phase understands the system requirements and the system specification. The specification is designed in terms of functionalities and system main components: goals and sub-goals in term of *professional-agents* and *skill-agent*s, and decides on the ontology domain and ontology concepts.

4.6.1 DMMAS Cooperation Architecture

DMMAS methodology develops a system based on cooperative multi-agent systems architecture illustrated in Figure 4.9. The architecture consists of two main layers the *professional-agents* and the *skill-agent*. The DMMAS cooperation architecture is proposed and presented in (AlHashel et al., 2009a). To explain DMMAS cooperative architecture we first introduce the concepts of the *professional-agents* and the *skill-agent*.

Professional-agent: The *professional-agent* is responsible to achieve the user goal by adopting the services required and following a predefined goal execution plan. The *professional-agent* is categorised by its profession in a particular goal domain (from the services domain) for example, travel agency, properties agency, employment agent. Each *professional-agent* is supported by a set of individual agents which are *skill-agent*s possess the expertise to accomplish particular goal. The *professional-agent* does not have a direct link or connection with the *skill-agent*s. These two layers are connected dynamically at system runtime through a search model based on services (defining functionality) ontology domain. The *professional-agents* acts according to its profession (goal) requirement supplied by the goal execution plan then accesses an ontology based search to select the appropriate skills and when those skills become available it will adopt them starting to perform the goal. In this process it forms a team of agents to achieve the user goal.

Skill-agent: The *skill-agent* represents a unit of expertise in a particular goal domain and has the ability to achieve part or all of a goal. The *skill-agent* can be defined by a task in the goal set. This task could be an application program, a website, or a sub-system that cannot achieve the system goal by itself. The *skill-agent* is a free entity and can operate under any *professional-agent*. The *professional-agent* adopts the *skill-agent* to achieve the assigned goal and when the task is accomplished the links between them are dropped. In another words, the *skill-agent* can join and leave the *professional-agent* organisation. This means when the *skill-agent* joins it is part of the team and when it leaves it exits the team and is ready to serve another *professional-agent*. This is the core concept strength of DMMAS design as characterised by dynamic team formation process and dynamic system structure. DMMAS *skill-agent* is described in the system by its skill which is defined by its functionality i.e. what it does or the expertise or services it offers. This functionality is captured in the ontology model. The *skill-agent* functionality model works as a flexible link between the two layers, the *professional-agent* and the *skill-agent* level.

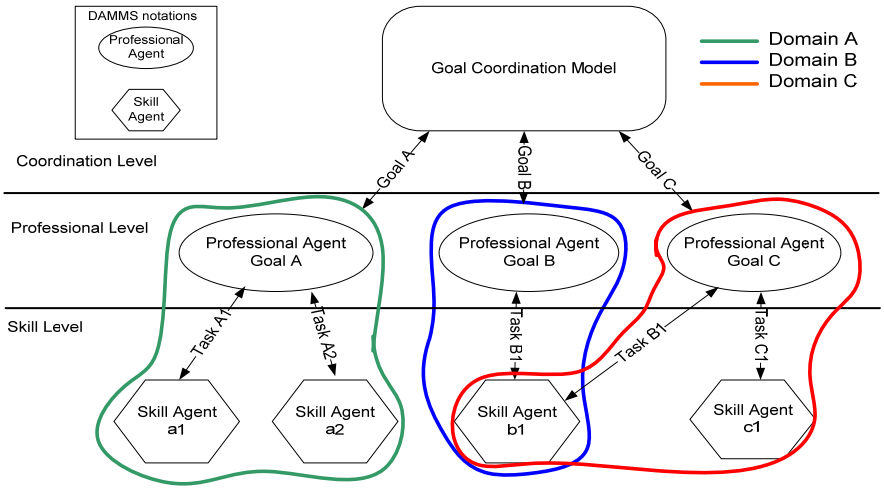

Figure 4.9: DMMAS cooperation architecture.

DMMAS cooperation strategy: DMMAS cooperation strategy is built around agent adoption strategy occurring between the *professional-agent* and *skill-agents* to achieve the goal. This cooperation is achieved through invitation messages sent by the *professional-agent* to the *skill-agent* based on the functionalities required in the goal descriptor. Figure 4.9 illustrates DMMAS cooperative multi-agent systems architecture consisting of *professional-agent* and *skill-agent*. For example assume that

there is a main goal represented by the *professional-agent* "A" that has two sub-goals denoted by two *skill-agent*s "a1" and "a2" consecutively to satisfy goal "A". The second goal scenario is denoted by *professional-agent* "B" and satisfied by binding a *skill-agent* "b1". The third goal denoted by *professional-agent* "C" is requires two *skill-agent*s "b1" and "c1". Therefore there is team intersection between Pa (B) and Pa (C) as depicted by the red line crossing the blue line. This intersection explains that the *skill-agent* offer itself (role) to partially or totally to serve any professional goal in the system. This operation provides a flexible system that can form its own achievement team in any combination that satisfies its goal. In other words the system form a team of agents to achieve a common goal.

The process continues until all the *skill-agent*s accomplish their tasks and acknowledge the *professional-agent* at task finish stage. After finishing the entire process both the *professional-agent* and the *skill-agent* refresh their log and get ready to engage in another goal. Thus the *skill-agent*s are not allowed to engage with more than one team at a time. The professional- agent and the *skill-agent* do not change their role but the main goals keep changing, so the appropriate *professional-agents* are changes and accordingly the *skill-agent*s are changed. Operationally, the system dynamically reconfigured its structure according to the goal requirements.

In large scale systems the *skill-agent*s can be links with sub-*skill-agents*. The adopting rules are applied in the same scenario with the consideration that an agent is not allowed to engage with more than one team at a time. This architecture has three potential design advantages:

1. It allows the *skill-agent* to participate with any *professional-agent*; the *skill-agent* is a free member when it is not engaged with a specific *professional-agent* and can be used by (adopted by) any *professional-agent* to accomplish its goal.
2. It provides flexibility, scalable system environment; adding or removing skill-agents from the system will not cause a runtime failure but could block particular goals.
3. The skill-agents are non-repeatable; the skill-agent that performing a particular role can be adopted by any high level agent professional-agent. This means the system has no functional redundancy. For example if there is a skill-agent that

4.6.2 Identify the System Organisation Structure

In this section both the *professional-agent* and *skill-agent* will be identified based on the goal descriptor and the system context diagram developed in the initial requirements specification. There is one sub-organisation structure for every *professional-agent* attached to a set of *skill-agent*s, then all of these sub-organisations are connected together under one system hierarchical organisation structure. The system organisation structure consists of *professional-agent*, and *skill-agent*s complemented by basic services components of goal execution plan, goal alternative plan, and domain ontology model. All these components structured and connected together in this section are based on "perform PhD" example.

Specify the system professional-agents and skill-agents: The *professional-agent* concept has been explained previously. Specifying the *professional-agent* is a straightforward process, converting the main goals captured in the goal refinement process and shown in Figure 4.6 into the oval notation shown in Figure 4.10 then labelling it by a goal description name. Use the goal descriptor and goal scenario to verify the system main goals. The next step is to attach the *skill-agent*s that relate to that *professional-agent* then form the system basic organisational structure. Specifying the *skill-agent*s is a straightforward process; convert each sub-goal shown in Figure 4.6 to a *skill-agent* represented by hexagon notation in Figure 4.10 and label each with a meaningful description relevant to its role. This step concludes the system basic organisation structure. The next step is to extend this structure by goal execution plan that explains the course of coordination between the *skill-agent*s in order to achieve the *professional-agent* assigned goal.

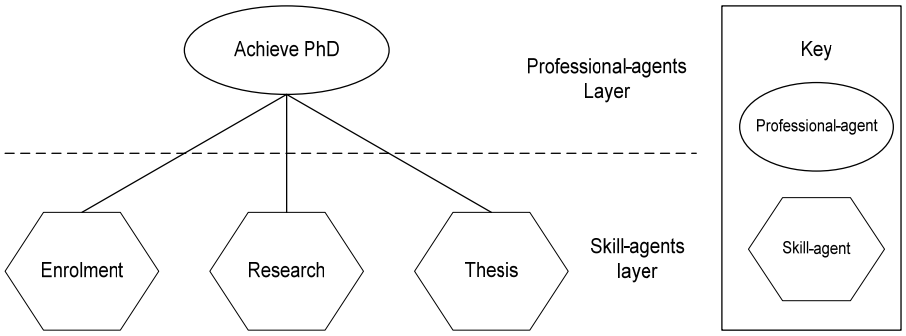

Figure 4.10: System basic organisation structure example.

Goal Execution Plan (GEP): The goal execution plan provides the course of execution coordination between a set of *skill-agent*s belonging to a particular goal. The goal execution plan contains the *skill-agent*s execution sequence for goal achievement in addition to the information about the goal (see Figure 4.11). In some cases the goal needs to be achieved by a team of *skill-agent*s therefore the team must execute or provide its services in specific sequence otherwise the goal will fail. For example "you cannot *submit a thesis* before you *conduct the research*". The goal execution plan includes the skills required to achieve each goal in the system. Building the execution plan can be done in multiple stages and depend on the goal tree. To set up the goal execution plan considers (a) Database sequence: for every sub-organization node in the goal tree and in line with the goal scenario classified each main goal data requirement dependency. For example is there a goal to update the database before the second goal start using this database? (b) Logical sequence: for example first start the car engine then drive. (c) Protocol sequence: depending on given rules or procedures for example traffic control systems. However, the general rule in setting the goal execution sequence is to investigate using And / Or conjunction. Whenever there is "and" there must be sequential order, and whenever there is "or" means in random order.

Professional Agent Goal Execution Plan						
Goal Id		**Goal Name**				
Goal Description						
Plan Id		**Plan Name**		**Execution Type**	Sequential	Random
Execution Sequence	**Skill Agent Id**	**Skill Agent Name**				
1						
2						
3						
4						
5						
6						

Figure 4.11: *Professional-agent* goal execution plan.

Figure 4.11 illustrates the goal execution plan descriptor that will be used by a particular *professional-agent* to achieve the goal assigned to it. The goal identification, name, and description are supplied by the goal descriptor shown in Figure 4.8. Each plan is defined by identification and name then assigned by a list of *skill-agent*s execution sequence. The execution plan can fail at the system runtime. To handle this exception an alternative plan is proposed. The next step specifies an alternative plan that can take over the system goal in case of a goal execution plan failure.

Alternative plan (AP): Alternative plan acts as a recovery plan in case of goal failure. Alternative plan can be set up by analysing the goal scenario, i.e. at each performative goal statement ask "what to do if this *skill-agent* fails?". The alternative plan is defined at every *professional-agent* for every *skill-agent* in the system. However, the alternative plan is not a compulsory component but it is advisable the system operates in open distributed heterogeneous environment and failure is expected. Figure 4.12 illustrates the alternative plan descriptor defined by plan identification, name then the *professional-agent* identification and name. The condition field describes the *skill-agent* identification and name along with the action field to describe the alternative system behaviour.

Alternative Plan Descriptors				
AP-Id		AP- Sequence		
AP Name				
Agent ID:		Condition	1	
Agent Name			2	
Option				
Action	1			
	2			
	3			
	4			

Figure 4.12: Alternative Plan Descriptor.

Initial system diagram: The purpose of initial system diagram is to establish a high level system organisations structure and its main components: the *professional-agent*, *skill-agent*s, database, the execution plan, and the alternative plan. Figure 4.13 illustrates an initial system diagram for the "achieving PhD" example. By this step we realise the required system structure and verifies its main components and organisation structure.

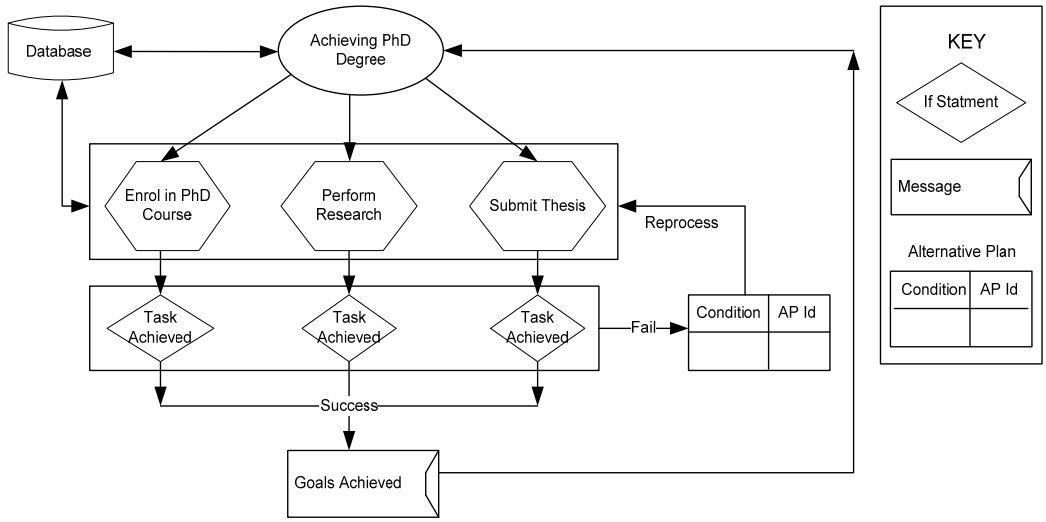

Figure 4.13: Initial system organisational structure example.

Data-used and data-produced: The data-used is the input data required for a particular process to establish a transaction and the data-produced is the change or the new data generated through that process. Capturing the data-used and data-produced provides a realisation to the type of process or identifies the process functionality. The

data that is outside the system context diagram (if any) must be also identified and included in the data model. DMMAS introduce the data-used and data-produced approach for each *skill-agent* in the system for two reasons first, to specify the *skill-agent* functionality subsequently using this functionality as domain ontology for this *skill-agent*. Second is to develop the entire system database schema.

Figure 4.14 illustrates the data model emphasising the table name, data name, and data groups. The goal descriptor is the source for the data used and produced where the table name defines the object of the data. The idea behind the data-used and data-produced is similar to the pre-condition and post-condition techniques used to specify the process in use-cases scenarios for object-oriented requirement analysis as proposed by John (2005). DMMAS instead use techniques to define the *skill-agent* functionality based on the changes over the data. The example in Figure 4.14 the data-used name is "Applicant" and the data-produced is "Student". In this case the *skill-agent* functionality is "Enrolment". After the functionality is labelled this label is Ontologies in respect of the data in the model to represent the *skill-agent* functionality.

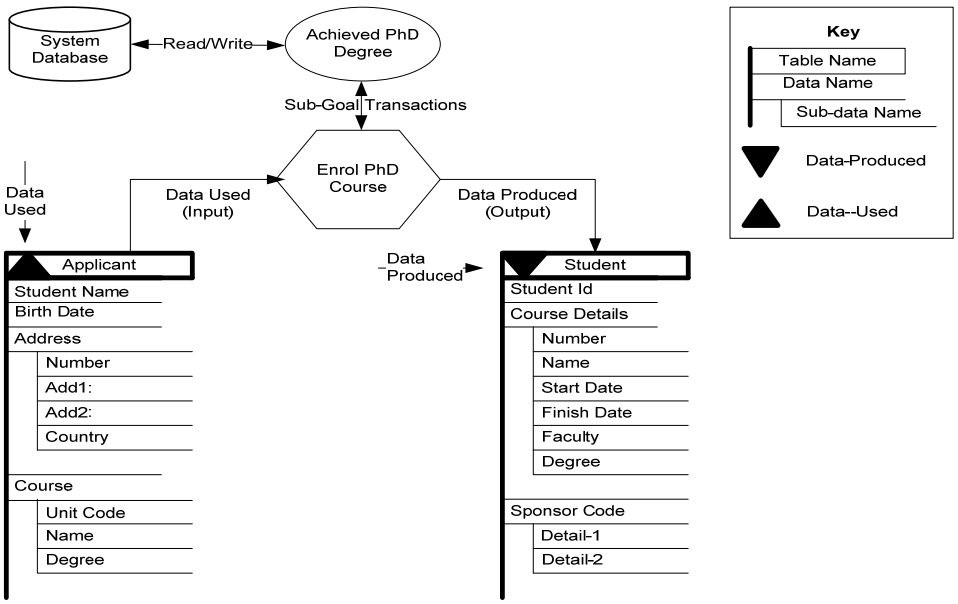

Figure 4.14: data-used and data-produced model example.

4.6.3 Ontology Design Consideration

Ontology can be structure in a long hierarchal concepts relationship but the aim at this design stage solely identifies the ontology domain or the scope of the ontology within the system problem domain. The system problem domain is represented by the union of ontology schema for each *skill-agent*'s functionality including domain classes, and concepts. Improving the ontology schema design process requires considering the following:

1. Currently, ontology paradigm has no standard schema, one domain model can be Ontologies using different ontology schema depending on the implementation technology. For example class, subclass, relations, functions, concept, slots, object, instance, role, individuals, elements all have different meanings in different ontology programming languages. Therefore the properties of ontology schema for particular domains are relevant to the language used. The DMMAS ontology model is designed toward ontology markup language (XML, XML(S), URI, RDF, RDF(S), and OWL). These implementation approaches focus on classes, relations, functions, and instance. On the other hand DMMAS ontology development processes deliver these properties.

2. Avoid complication and minimise the number of concepts per class and select those related to the agent main functionality. For example the "enrol in PhD course" is enough to define the concepts "member", "PhD", and "qualification" where concepts like Faculty, University could be generalized over the entire system.
3. Consider to using the existing legacy ontology thus there are ready made ontology definitions on the web and in the literature (Helena Sofia et al., 2001). For example DMAL ontology library, provide commercial Ontologies (UNSPSC, RosettaNet, and DMOZ). There are also predefined reusable domain Ontologies for specific application including medical, engineering, law, accommodation.

4. The ontology domain is not necessarily generalized to all the classes in the *professional-agent* node, thus some concepts can be generated by composite classes.
5. According to Gomez-Perez et al. (2004), UML class diagram might be used to model domain ontology but UML is inadequate in formal semantics, and for example transferring the model from UML to DMMAL+OIL ontology implementation language was not straightforward and required transformation effort. Sirichroen (2007), proposed extending UML notations to improve ontology modelling. Other ontology modelling options are defined in Chapter 2 along with their limitations. For these reasons DMMAS proposed new techniques supported by diagrams and notations and as the DMMAS is developed these techniques and the notations are explained in the following sections.

4.6.4 Ontological Analysis

Currently, in the agent-based system development practice the agent name or agent identification is used as the key for agent services search. The services search technique is text-based, listed by agent name to represent the agent services description. For example JADE platform services discovery model asks the user to insert a name for an agent in JADE database table (registry) and when particular services are needed a query statement loads that name then searches to retrieve the services required. This search technique is known as the yellow pages directory method.

The text-based services search technique limits the services description. It involved predefined agent names within each search process. Furthermore in the real open domain applications it is not possible to distinguish the services name throughout a large scale open heterogeneous system specifically if there are many agents providing similar services. This weakness is resolved in DMMAS through Ontologies. Each *skill-agent's* functionalities then develop a service discovery model that can parse the ontology to infer the *skill-agent*'s functionalities (what it does).

The aim of using the ontology approach is to create loose coupling links between the *skill-agent*s and the *professional-agent*. Subsequently, the system becomes open and the *skill-agent*s are autonomous and not committed to particular *professional-agent* or application domains. The ontology based search also enables the system to be flexible, scalable, dynamic, and provide the potential to automatically create a team of *skill-agent*s at system runtime. The team process is essential behaviour in the open multi-agent systems, to allow agent to exercise its fundamental key concepts "agent cooperation" (Alhashel, 2008), (Bulka et al., 2007a), (Matthew and Marie, 2008). According to Pal'chunov (2005), ontology is a useful tool for knowledge engineering, as it describes an object domain knowledge represented by ontology as inter-subjective (it means that different experts in the given object domain should agree with the statements presented in the object domain ontology) (Pal'chunov, 2005).

Identify an ontology scope: To start developing the ontology schema for the problem domain it requires identifying ontology scope. This will help to establish a guideline for the ontology range and avoid exceeding the purpose of the ontology.

Note: to improve the readability the next steps are explained in a common template of:

Objective: The purpose of the step.

Resources: The artefacts required to achieve the objective.

Potential techniques: How to achieve the objective

Output: Artefacts delivered.

Objective: Identify the purpose and scope of the ontology.

Resources: Required system main document, Why-refinement diagram, How-refinement diagram, goal scenario document.

Potential techniques: define why the ontology is being built and for what it is going to be used, for example knowledge sharing, knowledge reusing, or as part of an existing knowledge base. The answers to these questions provide the specification for the ontology (Breitman et al., 2007).

Output: ontology scope descriptor shown in Figure 4.15. The first row describes the ontology purpose, i.e. why this ontology is needed (the objective). The usage

is to define the application purpose and the beneficiary identity or who will use it.

Return: Define the project plan and limit the scope of the ontology model.

Ontology Scope		
Aime		
Usage	Sharing	
	Reusing	
	Part of	
	Other	
Beneficiary	Description	

Figure 4.15: Domain ontology scope descriptor.

Identify an ontology domain: According to Breitman et al. (2007), the second step in the ontology development method is to determine the domain relevant concepts that are to be included in the ontology schema then establish the initial domain hierarchal concepts. At this step it is not necessary to identify all the concepts as other concepts may emerge at a later development stage.

*Objective**:*** Identify the ontology domain and the high-level concepts related to the *skill-agent*s (the domain conceptual model without the relationships).

Resources: How-refinement diagram, goal scenario, the initial system diagram, and data-used and data-produced diagram.

Potential techniques: DMMAS proposed new ontology notations as appears in Figure 4.20 and for identifying the ontology concepts apply the competency questions techniques proposed by Gruninger and Fox (1995) with the consideration to Noy and McGuinness (2001) guidelines to determine the ontology domain: (a)There is no one correct way to model a domain, there are always viable alternatives. The best solution always depends on the application that you have in mind and the extent anticipated. (b) Ontology development is necessarily an iterative process. (c) Concepts in the ontology should be close to objects (physical or logical) and relationships in the domain of interest. These are

most likely to be nouns (objects) or verbs (relationships) in sentences that describe the domain

To specify the domain ontology, focus on How-refinement organisation structure along with the goal scenario related to particular *professional-agent* goal. These two inputs provide an overview of the goal purpose and scope of that goal. The process is bottom-up approach, first identifying the concepts belonging to each *skill-agent* then finding which logical domain model can represent these concepts. Apply Gurninger and Fox (1995) competence question approach, for example; what are the steps this *skill-agent* aims to perform? Name the main entities in the process? Name the data resources required? Investigate the answers to these questions and list the concepts under each *skill-agent* as per the example depicted by Figure 4.16. At this stage, the ontology domain concepts are allocated under each *skill-agent.* The next step is to classify these concepts. Group these concepts under one common class or type. It can also use iterative grouping until the end with a final single classification. For example, the concepts belong to the *skill-agent*s "Enrol in PhD", "Perform Research", and "Submit Thesis" under the *professional-agent* "Achieve PhD Degree" organisation need to be grouped under particular classification term, i.e. "*Education*". However, if it becomes difficult to classify the concepts, then use the sub-grouping approach. In this situation every similar concept is classified by a particular sub-group and every sub-group classified under another sub-group. This process continues until all the subgroups are set under one main group.

Output: The ontology domain diagram shown by Figure 4.16 is the *skill-agent*s concepts for "PhD goal achievement" example. The ontology domain "*Education*" is denoted between open square brackets. The ontology concepts for each *skill-agent* are depicted by the circles attaching their own *skill-agents*. These concepts given in Figure 4.16 are not finalized, there could be additional concepts required depending on the strength of the schema expressions to answer the expected ontology query.

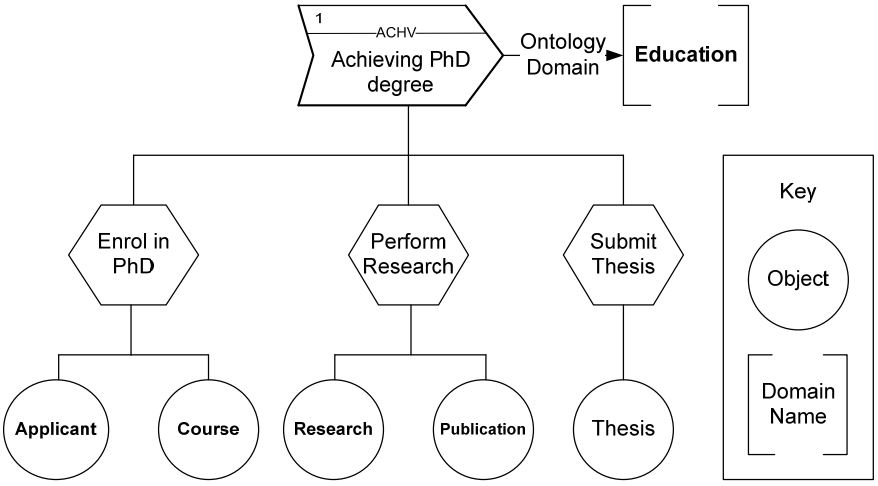

Figure 4.16: Domain ontology high-level model.

Specify the domain ontology scope: The next step in the ontology analysis is to reveal the range of concepts belonging to each *skill-agent* in the system domain. When the concepts are allocated then a provision on the range of the required domain ontology can be decided and eventually the ontology scope is determined. The underlying purpose behind this process is to adjust the ontology schema with the ontology purpose (usage) and set up a boundary around the required ontology domain and discard what is outside the boundary.

Objective: Extend the domain ontology to identify the concept's objects and identify any additional concepts then determine the scope of the required domain ontology.

Resources: Domain ontology scope descriptor, domain ontology diagram, goal scenario, and application problem scenario (required system story).

Potential techniques: Gruninger and Fox (1995) ontology development techniques based on three input devices: Scenarios that arise in the applications (problem story), Requirement (Query: set of questions that an ontology must be able to answer), and competency questions. Noy and McGuinness (2001), have suggested a useful set of questions to help in the development of an ontology by defining its domain and scope (Noy and McGuinness, 2001): (a) What is the domain that the ontology will cover? (b) For what we are going to use the

ontology? (c) For what type of questions should the information in the ontology provide answers? (d) Who will use and maintain the ontology?

Considering the two suggestions and the initial domain ontology scope descriptor, hereafter the previous scope descriptor is developed further to contain these suggested question techniques as described in Figure 4.15. To explain the usage of the descriptor an example of the *skill-agent* "Enrol in PhD" has been applied for illustration.

Domain and Scope Ontology Descriptor						
Aime	To Ontologies the skill agent functionality, and make it knowledge-based searchable			**Owners**	Enrol in PhD	
					ENRL	
Usage	**Sharing**	Non				
	Reusing	Available for the future developments				
	Part of	Non				
	Other	Non				
Beneficiary	**Description**	Achieving PhD application				
	Others					
Domain	Education		**Exist**		**Full**	
			NO	Yes	**Partial**	ProtegOntologyLibrary
Queries	1	Domain	6	Skill title		
	2	Degree level	7			
	3	Course name	8			
	4	University name				
	5	Duration				
Client	Owner					
	Other skill agents					

Figure 4.17: Domain and scope ontology descriptor.

After establishing the scope descriptor as in Figure 4.17, use this descriptor to facilitate the analysis process and then extend the domain ontology model by the domain objects that belong to each *skill-agent*. Since there is a shared understanding between the two paradigms ontology and object-orientation, a class diagram can be used to represent objects (Breitman et al., 2007). On the other hand the database table actually is similar to objects that have both attributes but in terms of representation in their own paradigm they are the same. As stated earlier in Chapter 2, the entity relationship diagram has been used to design ontology with some limitations but DMMAS proposes its own new notation to represent ontology domain objects as shown in Figure 4.17.

This analysis phase means a conceptual modelling not the physical final model.

To identify each *skill-agent*'s objects involves focus on: (a) Concepts in the ontology should be close to objects (physical or logical) and relationships in the domain of interest. These are most likely to be nouns (objects) or verbs (relationships) in sentences that describe the domain. (b) A set of variable names that describes its states. (c) A set of operations which can act upon the object to alter its state. (d) Determine where an entity in the application should be regarded as an object in the system. The object condition is occurred when an entity contains a set of variables that can be defied under one action. For example the *skill-agent* "Enrol in PhD Course", has entities like; Course, Faculty, and Units' which are objects because they contain attributes that can change states.

Output: Ontology domain objects diagram (Figure 4.18) depicts the ontology domain elements objects and objects attributes existing within each *skill-agent*. The single line across the circle means it is a subclass and the double line across the circle means an object in a class but converted to sub-class because it has many attributes and it is available as a class in the existing ontology library. The attributes under each object illustrate the object properties. This provides the background to evaluate the object to whether include this object in the ontology schema or to ignore it.

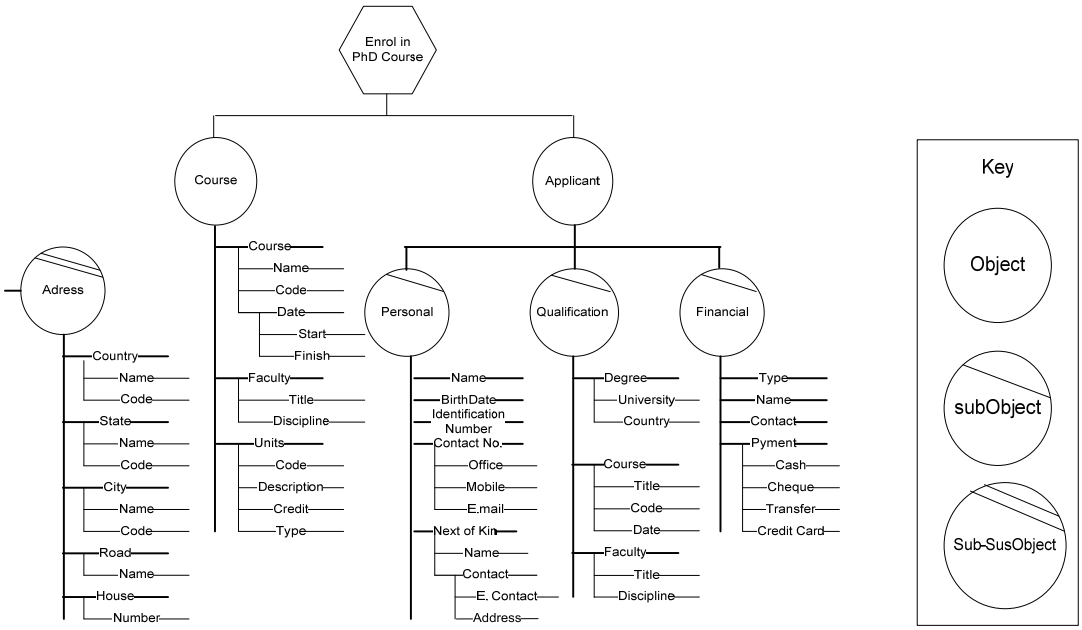

Figure 4.18: Ontology domain objects diagram.

Skill-agent transition status: The purpose of the *skill-agent* transition status is to define the *skill-agent* functionality based on pre and post status. Each *skill-agent* in the system must have input-status and output-status then the functionality or the process that transfer the input to output is called transition-status. The DMMAS view is that the *skill-agent* functionality is a single abstract process aiming to change the status of one class to another class. By defining the transition-status we defined the *skill-agent* by input-status, transition-status, and output-status. The focus in this step is to name or specify the transition-status which embedded the *skill-agent* functionality. The transition-status helps to define a class for the *skill-agent* functionality then categorises the functionalities improve the skill search process.

Objective: Identify the *skill-agent* transition input and output status in order to focus the ontology design model and provide first level functionality search.

Resources: Ontology domain objects diagram, *skill-agent* scenario, and domain and scope ontology descriptor.

Potential techniques: Study the domain and scope ontology descriptor carefully and identify the aim of the ontology main objective by asking what this descriptor informs about the *skill-agent* specific function. Identify the relation between the descriptor's queries field and the ontology domain objective diagram then produce one (verb, or act) word for describing its functionalities, remembering

to view from a functionality (task) perspective, for example the *skill-agent* "*Enrol in PhD*" can be described by "*Enrolment*". Now, in the same manner the input and the output transition status can be specified. The main question for the transition status description is "what object or class does this *skill-agent* aim to change in term of *from*, and *to*? The answer to this question will provide status name as object or class. For example the *skill-agent* "*Enrol in PhD*" can be defined by input-status "*Applicant*" and output-status "*Student*". Note that if the answer falls in attributes only then identified the class or object to which those attributes belong to.

Output: Text based *skill-agent* transition status as shown in Figure 4.19. The elements of the descriptor are transformed from the resources where are the transition name, the input, and the output status.

Skill-agent Transition Status				
Skill-agent	**Code**	ENRL	**Note**	
	Name	PhD Enrolment		
Status	No	**Input**	**Output**	
	1	Applicant	Student	
	2			
Transition	**Name**	Enrolment		
	Attribute	PhD Course		

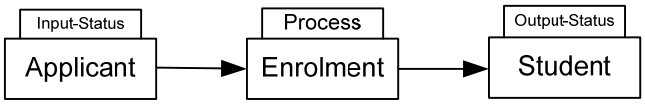

Figure 4.19: *Skill-agent* transition status.

The DMMAS requirement and specification phases are able to complement each other to form an entire system analysis phase. The DMMAS analysis phase is able to function in two approaches; the software system requirement and ontology requirement. Generally, the analysis phase is the initial step in the software development life cycle where the requirements are defined independently of any implementation details. However, at this stage DMMAS covered all the required system requirements and specifications and were able to deliver the system conceptual components listed in Table 4.1.

Step	System Software Analysis	Ontology Domain Analysis
1.	System context diagram	Ontology domain Scope descriptor
2.	System goals and sub-goals diagram	Conceptual ontology domain model
3.	*Professional-agents* and *Skill-agent*s system hierarchal Model	Ontology domain scope descriptor
4.	*Skill-agent*s.	Ontology domain objects diagram
5.	Execution Plan	*Skill-agent* transition status descriptor
6.	Alternative Plan	
7.	Data-used and Data-Produced model	

Table 4.1: DMMAS analysis phase outputs.

4.7 System Architecture Design

DMMAS architecture design uses the system specification artefacts. It develops the required multi-agent systems components architecture. The architecture design defines the interconnection and resource interfaces between the system components, and modules in ways suitable for their detailed design and overall system configuration. "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*" (Bass et al., 2003).

Background: The DMMAS design approach specifies building dynamic organizational multi-agent systems architecture consisting of *professional-agents* at the root node and *skill-agent* at the leaf node. This organisation structure is dynamically reconfigured at system runtime to meet the goal requirements in terms of skills. The *professional-agent* is the goal expert agent which will select the appropriate *skill-agent*s the agent possess the skills required to achieve that goal. The selection process performed through search model matches the goal with *skill-agent* functionalities. When the matching process is successful, the search will provide the *skill-agent*s name, and locations in the system. The *professional-agent* then **adopts** these *skill-agent*s forming two levels of organizational structure. The *professional-agent* set is on the head and the **team** of the skills agents that had been selected to achieve the goal is set on the branch node. In such a dynamic organizational structure achieving the goal in team work style involves coordination rules to govern the team

performance. The rules are pre-design presented by goal execution plan and can be accessed by the *professional-agent* only. This is because the *skill-agent*s are totally independent and available to engage with any team depending on the goal plan.

Skill-agents: This step concerns identifying the *skill-agent*s intra processing then developing the appropriate architecture to connect these processes together to form the required *skill-agent* design architecture. Begin by decompose each *skill-agent* into tasks where each task is characterised by input, output, and processes. All the tasks are connected with each other through their inputs and outputs forming integrated chains of tasks. For each *skill-agent* there must be an initial input name and output name; these input and output names are the input-status and output-status. The input-status and the output-status reflect the changes or the *skill-agent* functionality or what it intended to do. The objective in this step is to identify the *skill-agent* functionality and label it as an object name for the consumption of ontology.

Objective: To build the *skill-agent* architecture.

Resources: Skill-agent descriptor, data-used and data-produced diagram for each *skill-agent* diagram, *skill-agent* scenario, goal scenario.

Potential techniques: Study the data-used and data-produced diagram and identify the input status and the output status for each *skill-agent*. The input-status is the name of the ownership of the data-used and the output-status is the ownership of the data-produced. For example the *skill-agent* "*Enrol in PhD Course*" has input-status "*applicant"* and the output-status is "*student"*. On the other hand the *skill-agent* "*Research*" has input-status "*student"* and the output-status is "*researcher"*. However, after the input-status and the output-status are specified the *skill-agent* internal tasks must be identified. Breaking the *skill-agent* functionality into a set of tasks where each task represents an independent unit of work and has inputs, outputs and process. To capture the tasks investigate the data-used and data-produced then map both by a set of tasks with consideration to the *skill-agent* scenario. Propose activities to perform the scenario then group each coherent set of activities to form one task. For example, the data-used and data-produced for the *skill-agent* "*Enrol in PhD Course*". The activities that change the data (*studentId, courseDetails*, and *sponsorCod*e) are can all be mapped with *studentDetails* in the tasks

Apply: fill application and send application to the admission office, and *Assessment*: application received, and application number associated, course requirement and applicant qualifications evaluated, then decision made, and *Pay Fees*: associate sponsor code, choose the payment method, verify the account balance, make payment. The functionality of these tasks defined between the input-status "*Applicant"* and the output-status "*Student"* is the enrolment.

Output: Figure 4.20 presents high-level *skill-agent* architecture in context of "*Enrol in PhD Course*" example. The diagram divides the *skill-agent* "*Enrol in PhD Course*" into three main tasks *Apply, Assessment, Pay Fees and On-Hold*. The diagram also shows the input-status *Applicant* and the output-status *Student*. The notations used in this diagram are explained by the key diagram.

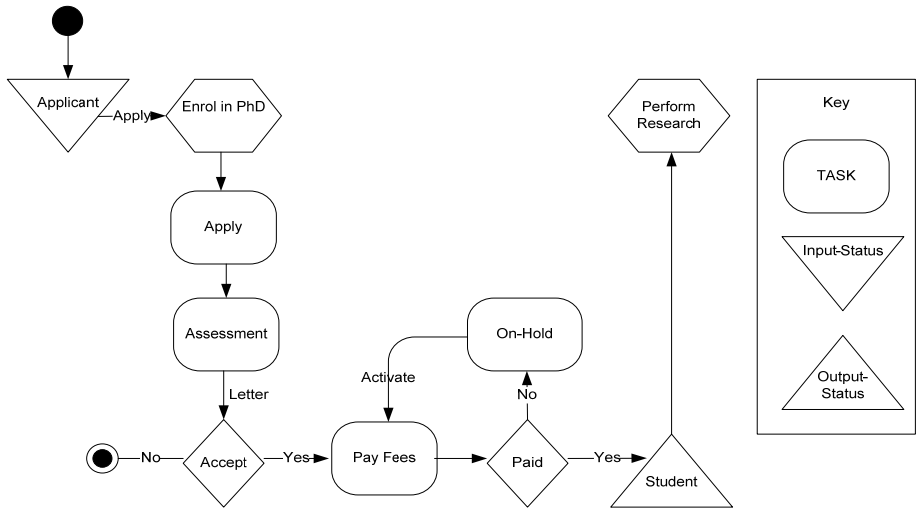

Figure 4.20: *Skill-agent* architecture example.

Professional-agent: The *professional-agent* is the goal expert equipped by the goal execution plan and a set of *skill-agent*s required to achieving the goal. DMMAS cooperation structure is located the *professional-agents* on the head node where the *skill-agent*s dynamically set on the branch node. The *professional-agent* has the capability to adopt the *skill-agent*s according to the goal requirements and skill selection. The selection is process performed through search based on *skill-agent*s functionalities ontology schema. Therefore the *professional-agent* is the goal

execution manager with privileges to access the goal execution plan, and ontology schema.

Objective: Set up the *professional-agents* architecture.

Resources: System scenario, goal scenario, *professional-agents* diagram, *skill-agent*s diagram, goal execution plan.

Potential techniques: Extend the *professional-agents* node diagram

Output: Professional-agents architecture with the access mechanism to the *skill-agent* functionalities ontology model, see Figure 4.21. The *professional-agents* architecture consists of the *professional-agents* abstractions, *skill-agent* search model, goal execution plan, and alternative plan component. The key diagram explains the notations used and provides their meaning in the context of goal executions. The random and the sequential notation are used for the execution order, and the rest are used for the execution options.

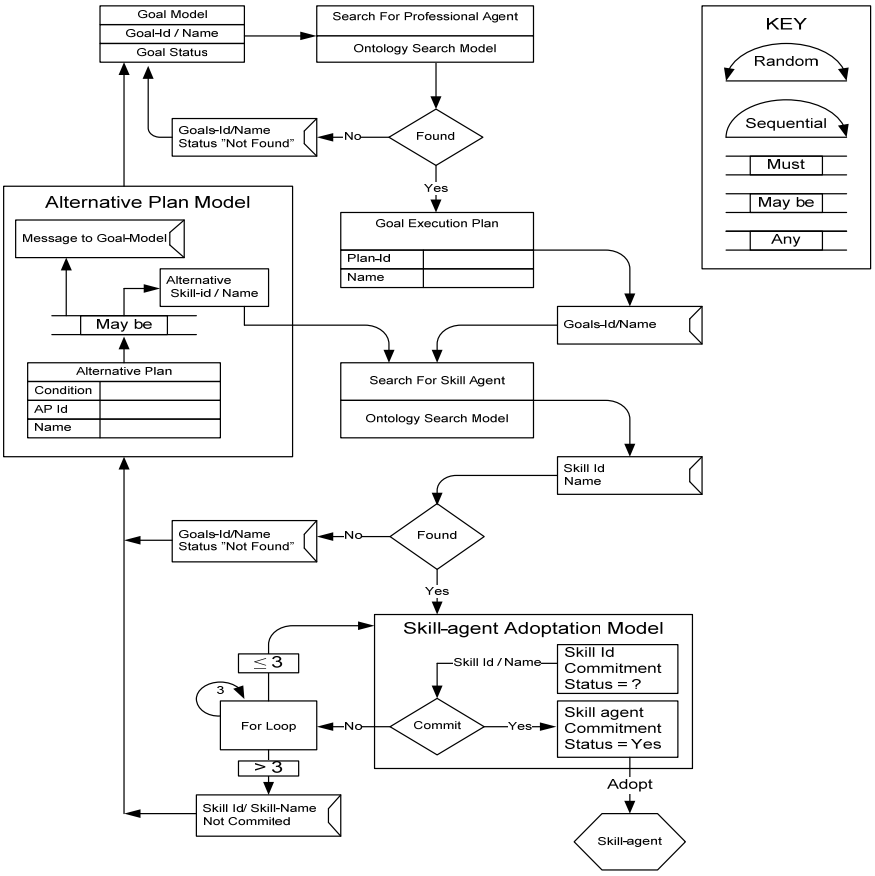

Figure 4.21: *Professional-agents* architecture including search component.

Skill-agent search model: The *skill-agent* search model is the logical mechanism that provides the system with the ability to enquire for the *skill-agent* functionality (what it does). This search model is the core advantage of the DMMAS methodology over the entire existing multi-agent systems development methodologies. Ontologies *skill-agent* functionalities enable the system to operate in an open distributed mode thus the *skill-agent*s are represented logically as a record in the ontology schema without direct link to the *professional-agents*. The implementation platform for the *skill-agent* functionalities is based on XML technologies Ontologies. The research uses semantic web technologies, XML-based markup languages, RDF resource description framework, and Web ontology Language (OWL) ontology classes to define the *skill-agent* functionality in the previous section. Utilising the web ontology technology on agent-based system has been proposed by (Alhashel et al., 2009b). Furthermore, "*Although OWL was initially designed for use in (the development of) the semantic web, it has rapidly become a de facto standard for ontology development in general*" (Grau et al., 2008).

DMMAS proposes its own techniques for ontology analysis and design. It develops the *skill-agent* functionalities ontology domain into the type of services similar to the semantic Web approach then uses software engineering to engineer the domain ontology similar to that presented in Dragan et al. (2007), and Gaševic et al. (2009). While maintaining the development objective, the Web ontology development technologies are applied over the skill- agent domain and reformatted into a fully ontological model. The ontology modelling process followed in DMMAS design phase considers the implementation requirements. This means through the design process, emphasis on the outcome artefacts to interface with the ontology Markup Languages (XML-based) implementation constructs listed in Gomez-Perez et al (2004).

However, ontology engineering involves additional design approach to the traditional software design. It also required generally accepted notations for representing ontology classes (Vladan, 2002). DMMAS proposes a set of ontology graphical notations to represent the ontology constructs and relationship, as shown in Figure 4.20. Each graphical notation present in Figure 4.20, describes itself and has a

meaning exactly like its text label, where the meaning of each term is available in Gomez-Perez et al. (Alessandro et al., 2002, , 2004).

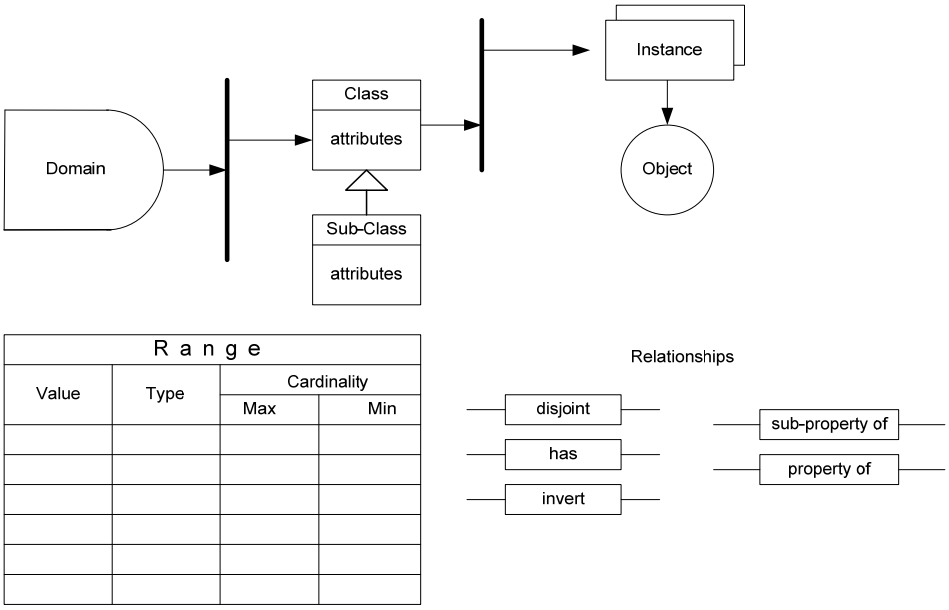

Figure 4.22: DMMAS ontology notations for design phase.

Domain ontology class: While the *skill-agent* domain ontology has been defined in the analysis phase, in this phase it must specify the classes that form this domain. The class diagram describes the ontology structure by aiming to convert the requirement development process from "what" to "how".

Objective: Developing the initial domain ontology class to determine the classes involved and outline the ontology basic relationship. The outcome of this process is the domain ontology class diagram which will gradually break down in the next steps until the objective ontology map is reached.

Resources: The data-used and produced diagram, transition-status descriptor, *skill-agent* architecture, domain ontology diagram, *skill-agent* scenario, goal scenario.

Potential Techniques: The data-used and data-produced diagram is the static data that reflect the system functionality from a data perspective. The *skill-agent* diagram and the domain ontology diagram are also strong inputs that help to specify the system classes. The principles of capturing the system classes are similar to the object-oriented practices with one main difference, that the ontology classes does not deal with methods or behaviours and they have

different relationships, that is property based relationships, of which the multiplicity is not relevant. For capturing the classes for ontology purpose perform the following steps:

1. The most important is to start with a domain ontology model which specifies the ontology boundary then uses the *skill-agent* scenario, in line with the data-used and produced diagram and try to build a kind of glossary for each *skill-agent,* i.e. try to popup nouns and verbs.
2. Extract the entities that seem important and related to the *skill-agent* functionalities description, for example there is no need to represent payment class in the domain ontology "Education"
3. Make a domain diagram containing this entity this would be the domain class diagram, where all the classes contain only those attributes which have impact on the transition event for this purpose using the data used and produced diagram. The methods (behaviours) are not relevant to the Ontologies modelling so ignore them
4. Add the relationships between the classes in a form of association properties
5. Finally you have to translate this to an ontology Class Diagram (some of the domain classes will probably disappear and some others related to ontology will appear) while you iterate the processes, a common practice
6. Now you have to repeat the same steps on the other the *skill-agents* (one at a time) then gradually extend the diagram to include all the *skill-agent*s in the applications, to be sure there is no class redundancy.

Output: The domain ontology class diagram illustrated in Figure 4.21 is an example for the "Enrol in PhD" *skill-agent*. A class is represented by a rectangle divided into two parts, the upper part is the class name and the lower part is for the correlated attributes that have impact at the transition-status. For example there is no "*payment"* class because the assumption is that payment has no ontological relation to the domain. The dotted lines are for clarification only and are not part of the design process. The class relationships are represented by a set of properties as they are defined in Figure 4.23 and each

relationship has a particular semantic definition by the implementation platform.

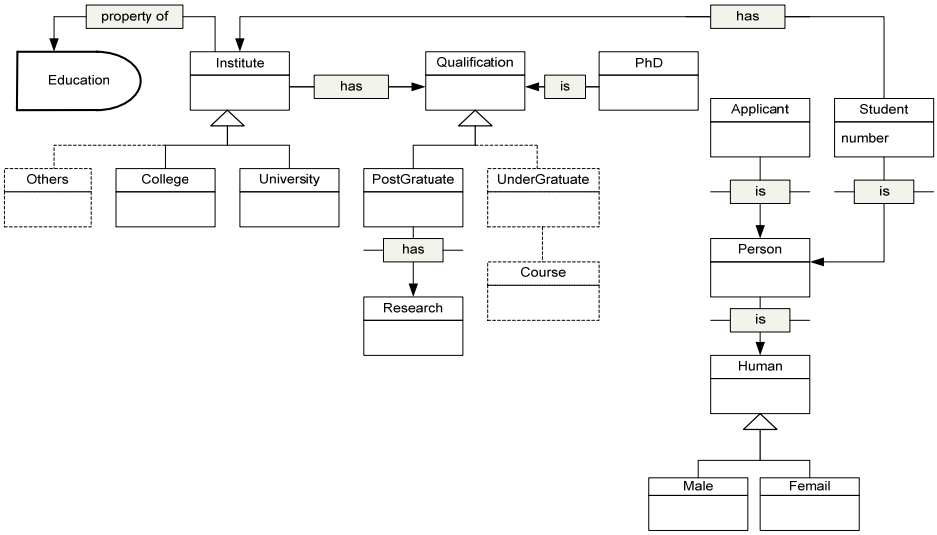

Figure 4.23: Domain ontology class diagram.

Extended execution plan: The goal execution plan descriptor introduced in the analysis phase does not contain guidance or information for the implementation phase because at the analysis stage there was not sufficient knowledge about the system architecture. Therefore it needs further detail to support the *skill-agent* team formation process. The extended execution plan is aimed to complement the goal execution plan by providing the plan details. The execution plan contains the skill required, the goal execution sequences and the execution rules.

The goal execution plan is sensitive to the system implementation and must also be precise and clear for the developer to understand it. For this reason, the goal execution plan is described in a semiformal language. The formal language provides an insight into analysis, specification, design and development of computer systems (Andrews, 1996). To manuscript the goal executions plan DMMAS use the semiformal notations introduced in Gaia methodology because it is easy to implement and expressive (see Table 4.3) and to comprehend the execution plan constraint DMMAS propose the additional semiformal notations set presented in Table 4.2 to fill the residual requirement. The interpretations of these operators are summarised in Tables 4.2 and Table 4.3 respectively.

Operator	Interpretation
$S \rightarrow Q$	skill Q can be executed by an agent only if it somewhere earlier skill S has been executed
$S^n \rightarrow Q$	execute skill R *n* times then you can execute Q
¬(S / Q)	cannot execute S and Q at the same time but one and only one
¬((S / O) / (Q / O))	cannot execute S and Q at the same time and cannot execute one only instead
$S^{1..n}$	The skill S must executed at least once and no more than *n* times (loop)
S (Property), S = Skill.1, Skill.2, .. Skill.*n*	The specified property has to be spread over all the skills listed, for example; S (*activate*), S = (reserveFlight, bookHotel, hireCar) mean activate each skill in a sequence.

Table 4.2: DMMAS semiformal set.

Operator	Interpretation
x, y	x followed by y
$x \mid y$	x or y occurs
x^*	x occurs o or more times
x^+	x occurs 1 or more times
x^ω	x occurs infinitely often
$[x]$	x is optional
$x \otimes y$	x and y interleaved

Table 4.3: Gaia semiformal set (Wooldridge et al., 2000).

Objective: Using semiformal method implementation to illustrate the goal execution plan.

Resources: Goal execution plan descriptor, *Professional-agents* and *skill-agent* relationships, *skill-agent* transition status, and domain and scope ontology descriptor.

Potential techniques: To make the implementation decoration more understandable, the *skill-agent* "*Enrol in PhD Course*" example will be implemented using both Table 6 and Table 7 semiformal operators to describe the plan.

Output: The semiformal schema presented in Figure 4.24 demonstrating the *professional-agents* "*Achieve PhD Qualification*" goal as an example. The schema started by defining the goal and the main execution plan then presentation of the *skill-agents* involved in the plan. Because the example focuses on the *skill-agent* "*Enrol in PhD Course*" the schema represent the execution rules for this *skill-agent* including, the skill requirement for this goal, execution sequence, the successful indicator, the execution cycle, the safety action, and finally implies the successful factors.

$G \Rightarrow Goal(AchievePhDQualification)$
$P_e \Rightarrow Main(ExecutionPlan)$

$s_e \Rightarrow SkillAgent(EnrolInPhD)$
$s_p \Rightarrow SkillAgen(PerformResarch)$
$s_h \Rightarrow SkillAgent(SubmitThesis)$

$\therefore skillRequired(s_r) ::= achieve(G), required(\forall(s_e, s_{p,} s_h))$

$success(P_e) ::= iff\ (execute(P_e(s_r)) \bullet execute(P_e) \wedge achieve(G)$

$executionSequence ::= sequantial(s_e \rightarrow s_p \rightarrow s_h)$

$executionOptions ::= must(s_r) \wedge sequantial(s_e \rightarrow s_p \rightarrow s_h)$

$e \Rightarrow executionCycle$
let $i = 1$
$executionCycle ::= e_i(s_e, s_p s_h) \bullet \forall e_i(execute(s_e, s_p, s_h))$
$safetyAction ::= terminate(P_e) \rightarrow restore(execute(s_e, s_p, s_h)) \bullet \forall database : x \mid x' \Leftrightarrow x \mid$
$\therefore$ for successful goal achievement the goal plan must verify the plan constrains to specify successful goal achievement:
$G_s \Rightarrow$ goalAchievedSucessfuly(G)
$G_s ::= \forall execute(plan(P_e)) \bullet sucess\forall((execute(p_e) \wedge (execute(s_r)) \Rightarrow Achieve(G)$

Figure 4.24: Execution Plan example for Achieve PhD goal focus on "Enrol in PhD course" *skill-agent*.

In case writing a formal language schema for goal execution plan is difficult, the diagram shown in Figure 4.25 can be used as option to explain the goal plan for every individual goal in the system. The arcs indicate to the execution directions started

from square number one on top of the *skill-agent*. The "Must" label on the arc indicates that this execution is compulsory otherwise the goal fails.

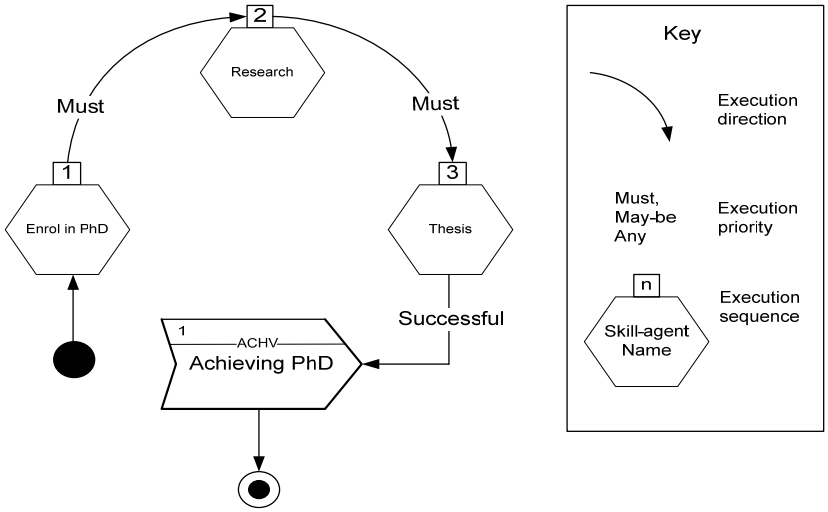

Figure 4.25: Goal execution plan diagram.

Ontologies skill-agent functionality: This step is to develop the ontology schema that defines the *skill-agent* functionality. The ontology schema consists of three main components; Information, Identification, and Functionalities. Each component has its purpose and structure designed to support the general idea, to make the *skill-agent* functionalities identified and understood from outside the model. Figure 4.26 depicts the *skill-agent* functionality structure highlighting the main components and their related objects. The information and the identification components are not for the Ontologies but to provide information about the *skill-agent*.

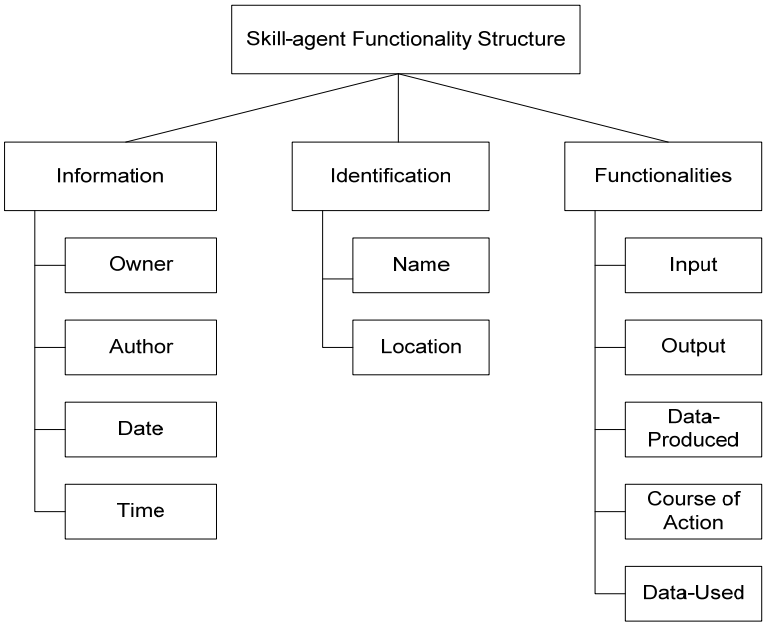

Figure 4.26: *Skill-agent* functionalities structure.

The information component: The information component captured the *skill-agent* ownerships.

Objective: To provide development and implementation information for documentation and security reasons.
Resources: Fill the information component attributes given in Figure 4.27.
Potential techniques: Straightforward fill the boxes with the information about the *skill-agent*.
Output: information form appears in Figure 4.27 which illustrates the *skill-agent* "PhD Enrolment" example.

The identification component: is the signature used to call the *skill-agent*. It consists of two parts, the *skill-agent* unique name already assigned at the analysis phase and the second part indicates to the *skill-agent,* the saved location on the network directory.

Objective: Tolerate the notion of distribution open environment that agent can act and reside in and to specify the call signature at the runtime.
Resources: skill-agent transition status and define the *skill-agent* save directory.
Potential techniques: Straightforward, fill the identification form, see Figure 4.27.

Output: Identification form appears in Figure 4.27 which illustrates the *skill-agent* "PhD Enrolment" example.

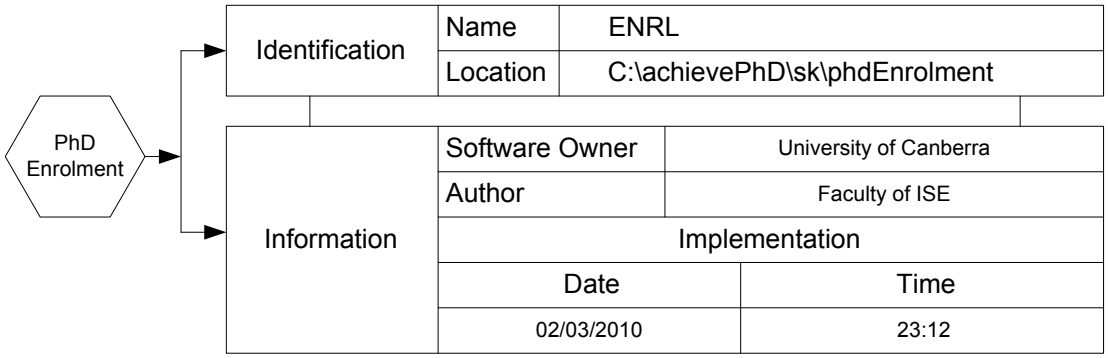

Figure 4.27: Information and Identification components example for *skill-agent* "PhDEnrolment".

The functionalities component: is an XML-based Ontologies component design to acknowledge the *skill-agent* functionalities (what it does) from outside access. The component deals with the input-Status, output-status, data-used, data-produced, and the course of actions. These objects are selected to be able to fill the criteria of the *skill-agent* functionalities proposed by the research. This functionality criteria are flexible and in certain circumstances can be adjusted or extended to suit specific requirements. Figure 4.28 modelled the criteria to two perspectives the expressive and the behavioural. However, both form one ontology body described as following:

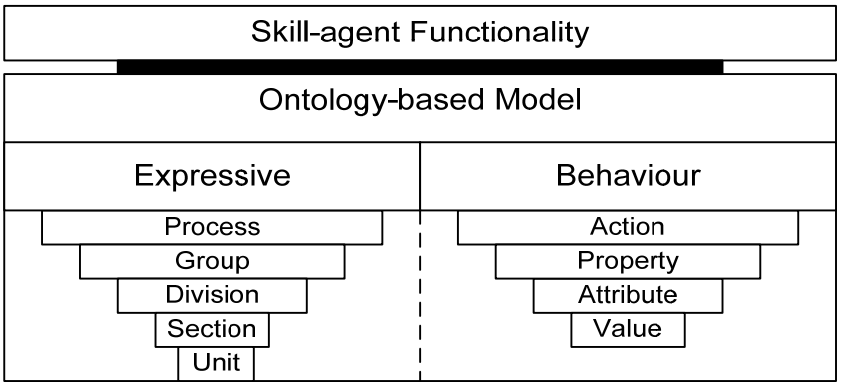

Figure 4.28: *Skill-agent* functionality architecture.

Process: functionality name or label of one word that expresses the functionality;

Group: Is the classification of the process. Group is the name of the type that categorises the process. If there is an existing ontology grouping to the process then it is preferable otherwise use the experience to decide a grouping.

Division: Is the breakdown of the group to subgroups in order to emphasise and point deeper into the process meaning. If the process is not highlighted then the designer can specify, **Section** then if not enough, proceed to **Unit** and so on until the functionality is fully expressed. The idea is to build hierarchical expression to the *skill-agent* functionality and when this hierarchy is established it can be considered as standard.

Action: Is the transition instances from input to output status described in a set of ontology classes.

Attribute: describes the internal structure of the of the action's classes. Attribute specifies the action in term of data-name required or produced by the class at the action course. Attribute is associated with classes.

Value: Is properties of attribute that give useful or desirable degree of clarification to the attribute. Value represents the magnitude of attribute in terms of data.

Property: Is a relationship between the classes of the process model. Property describes the logical links between the classes.

Instance: Is a result of the ontology query process for a particular concept. Instance is the ontology inference outcome in form of meaningful information.

Objective: To Ontologies the *skill-agent* functionality and make it a knowledge based model, with the potential to answer the enquiry from the search process about the *skill-agent* functionality or capabilities.

Note: Applying ontology is similar to software programming; it is creative work and has multiple implementations to achieve the same purpose. It also uses multiple iterations to get the design right. Therefore every developer Ontologies the same domains differently, depending on their design experience. For this reason it is unnecessary to explain a complete ontology schema at this stage however, Chapter 5 provides further development details.

Resources: date-used and data-produced model, Input-status, output-status (*skill-agent* transition status diagram), domain scope ontology descriptor, ontology domain object diagram, and domain ontology class diagram.

Potential technique: There are three main ontology development approaches: top-down; bottom-up; and combinations (Noy and McGuinness, 2001). There is no guidance for matching the approach with the problem type. Therefore whatever is convenient and simplifies the development process can be applied. However, study all the listed resources and enumerate all the important terms in the ontology then develop the query statement that the ontology schema must infer the answer to. For example, the *skill-agent* "PhD Enrolment" ontology schema must be able to infer this scenario:

Functionality is: **Enrolment** in course: **PhD** at institute of: **University of Canberra** Faculty of: **Information Science and Engineering** course code: **352AA** course name: **Information system**.

The infer scenario can be retrieved from the query statement:

```
select    skillAgentId, skillAgentLocation from skillAgentOnto
          where   process like "enrolment" and subject like "course" and
          attribute like "PhD" and value like "Information System and
          Engineering";
```

The Answer is: *skill-agent* Id = ENRL, Location = C:\achievePhD\sk\PhDEnrolment

The purpose for setting up the ontology scenario, the query statement, and the expected answers is to create guidelines to the ontology development process. With the ontology decoration in mind, apply the following steps:-

1. Out of the domain ontology class diagram, group each *skill-agent* classes then check the result with the domain scope ontology descriptor.
2. Apply appropriate one word name that is able to describe the domain functionality for example "*Enrolment*". Consider this domain as entire independent domain that the ontology schema should be able to express and answer the functionality query.
3. List the input-status and the output-status and define each as a class within the domain and associate their properties. At this stage both the domain and the concepts have been specified.

4. Following the same steps described earlier for designing the domain class diagram, start developing the class hierarchy. Maintain the classes naming consistency i.e. singular or plural names.
5. Ensuring that the class hierarchy is correct. The class hierarchy represents an "is-a" or "kind-of" relation: a class A is a subclass of B if every instance in A is also an instance in B. For example, a student is subclass of a person. Or student is a kind of a person.
6. Associate the date-used as instance of input-status and associate the date produced as instance of output-status.
7. Decide on the subclasses disjoint, for example "*applicant*" is disjoint from "*student*" class, thus each of these classes cannot have any instances in common while both classes are subclasses from a class "person".

Output: Figure 4.29 depict for functionality ontology model example of *skill-agent* "PhD Enrolment". The data used and produced become attributes to the classes attached or can be implemented as slots, depending on the ontology language. The dotted line classes are for clarification only, and are not within the scope of the ontology domain.

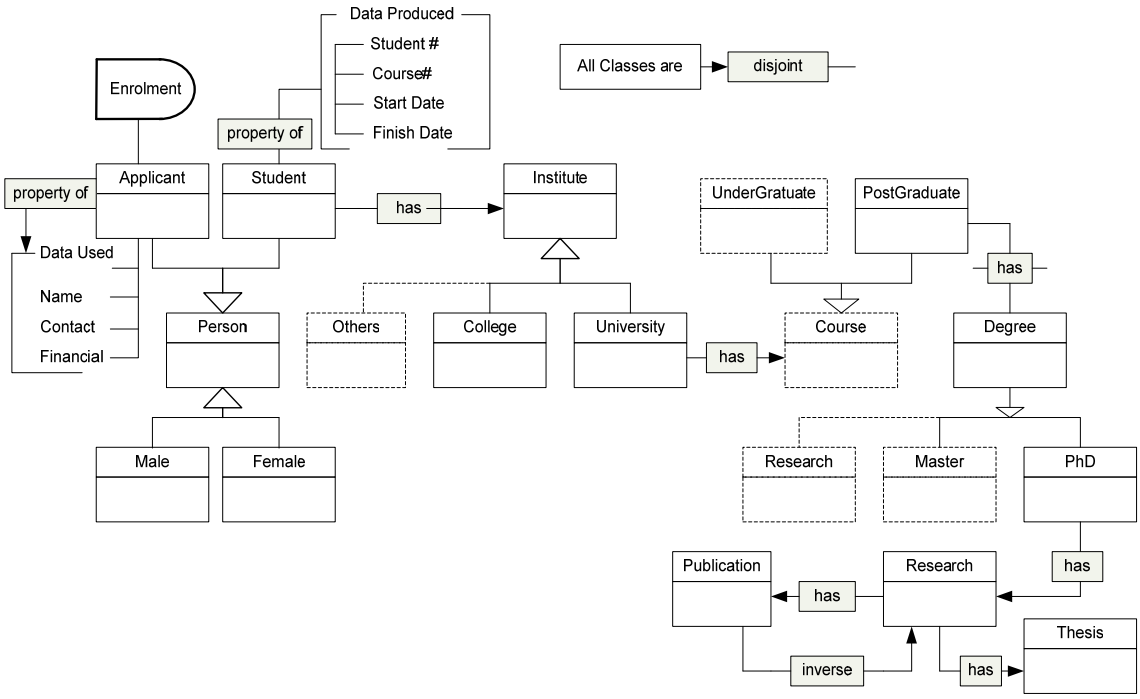

Figure 4.29. Example for *skill-agent* functionality ontology.

To illustrate the implementation of the functionality criteria below is the mapping between criteria and the *skill-agent* "PhD Enrolment" functionality ontology example shown by Figure 4.29:

Domain: Education

Process: Enrolment

Group: Course

Division: PhD

Section: Not applicable

Unit: Not applicable

Action: transition from *applicant* class to *student* class reflected by the new (produced) attributes associated with the *student* class.

Attribute: Used: applicant.Name, applicant.Contact, and applicant.Financial

Produced: student.Number, student.CourseNumber, student.CorseStartDate, and student.CourseFinishDate.

Value: the values are associated with the attributes in case the query needs to be specific. The values are defined by XML sheet or database schema. For Figure 62 ontology design the values are what answer the attribute, for the *applicant*; "Ebrahim Alhashel" "address", and "BDFsponsor". For the *student* class"u3002064", "352AA", "17-02-2010", and "18-11-2014"

Property: disjoint, inverse, has, and property of. Jules

Instance: Domain is: **Education** Functionality is: **Enrolment** in course: **PhD** at institute of: **University of Canberra** Faculty of: **Information Science and Engineering** course code: **352AA** course name: **Information system**.

Search model: The search model is the process that can discover the appropriate *skill-agent*s that can satisfy the goal achievement. The search model can read the user goal, the goal plan, then develop a query to access the *skill-agent* ontology schema then match the required skills with the existing skills. Figure 4.30 illustrates the search model structure. The develop queries component has access to the goal plan and based on that, it develops the quires of each *skill-agent* in the plan then accessed the main ontology schema to retrieve the required *skill-agent*. If the *skill-agent* is found then it will pass its Name, Id, and location on the network to the rule component.

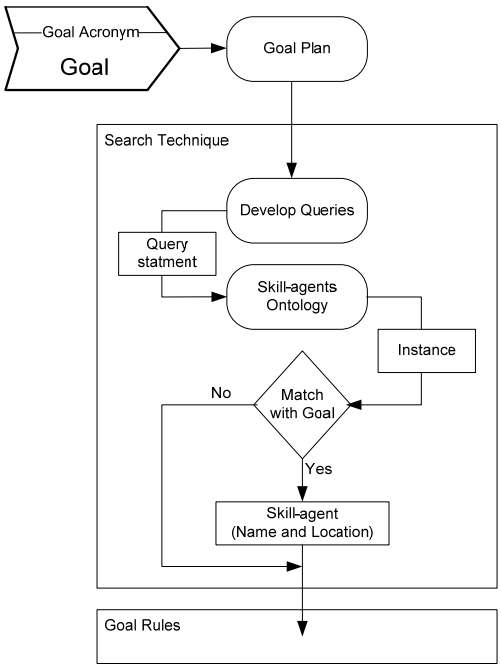

Figure 4.30: *Skill-agent* Search model structure.

4.8 Detailed Design

Through the architecture design phase, the system components, the agents and main descriptors have been defined. The detailed design phase takes the development process further to build the internal structure of the individual agent in terms of capabilities, activities, plan, events and messages, action and percept. The communication protocols will be developed based on AUML diagram to represent the agents' interactions. However, the details design finalise the system structure and make it ready for the implementation phase. The DMMAS detail design is not for any particular implementation technology but for an ontology model. The ontology model is driven toward XML-based and OWL ontology implementation technologies. Thus the ontology technologies are yet premature and different implementation technology has a different application approach, for example OWL has data aggregation features that match the *skill-agent* design and this is why DMMAS drive the ontology modelling toward OWL.

The DMMAS detail design notations: The DMMAS detailed design phase proposes its own modelling techniques and notation that has the potential to bring the entire

system artefacts in more readable and understandable diagrammatical notations. The DMMAS detail design does not join the system classes directly but it defines the conditions that link classes together. For this purpose DMMAS use a combination of UML diagrams and software flowchart notations making the detail design diagram between the UML class diagram and software flowchart. The UML activity diagram and class diagram are used along with the decision and merge notation.

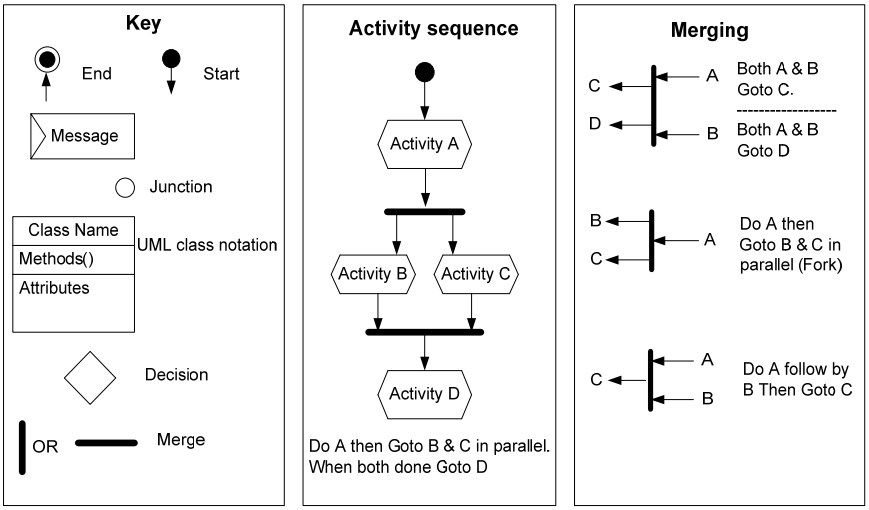

Figure 4.31: Diagram illustrating notation used in the DMMAS detailed design phase.

Figure 4.31 illustrates DMMAS detailed design notations and describes their semantics and uses. The message notation, start, and end notations have been explained and used earlier. The class diagram is AUML notation and the other notations are commonly used in software engineering. The activity sequence diagram illustrates a breakdown of a class into methods then to activities or processes and shows the flow (sequence) of these activities. The developer can use the activity notations and expand the *skill-agent* design into further details to expose the lowest possible process. The merging notations represent the process direction in case of multiple destinations (splitting the output) or merging different processes in one destination.

Skill-agent internal structure: Usually the multi-agent systems consists of more than one *skill-agent* and each *skill-agent* has its internal structure of input, output, and set of tasks (processes), and perhaps internal and external interaction. The overall delivery of these tasks forms the *skill-agent* main functionality. The functionality

determines the role of the agent in respect of its application domain toward achieving the system objective.

The *skill-agent* architecture depicted in Figure 4.31 is abstracted and needs to be extended further to expose its intra element details and make it more understandable and map able to the implementation process. It is unnecessary at this stage to break down each *skill-agent* class. Instead this section demonstrates internal structure of the *skill-agent* main component and their artefacts. For further design details see Chapter 5.

Objective: To break down the *skill-agent* intra structure and expose its task and processes. *Resources*: *skill-agent* architecture diagram, data-used and data-produced model, UML class diagram and activity diagram, AUML interaction diagram, and DMMAS detailed design notation.

Potential techniques: For every task in the *skill-agent* determine the set of the possible processes that can achieve that task. View the *skill-agent* as independent program or logical unit that forms a standalone sub system to define functionality or purpose to achieve. Analyse the *skill-agent* architecture diagram and identify the possible classes that can shape the *skill-agent* main objective. Link those classes together and try to establish a processing scenario with consideration to simplify the classes to not contain more than three methods at a time. Check the *skill-agent* classes if they contain subclasses, in which case split to super and subclass relationship.

Output: Figure 4.32 demonstrates the *skill-agent* detailed design diagram consisting of activities, databases, classes, and their intra messages for the *Enrolmen*t *skill-agent* example defined earlier. The main common component within each *skill-agent* body is the accept or reject class "*saAccept*". This class must be located at the beginning with three possible statuses; accept, reject, reserve. Reserve status indicates that the *skill-agent* is reserved for a particular *professional-agent* and not yet executed. The accept status means that it agrees to participate, usually takes place when the *skill-agent* is not committed to any job. The reject status indicates that the *skill-agent* is committed to another *professional-agent* or removed from the system or other reason that the

developer may assign. However, the process is started by requesting the *skill-agent* to offer its services. If "agree" it commits itself to the requester and changes its commitment indicator to "reserve". If the reply is "reject" then the goal plan is tested to verify the *skill-agent* commitment indicator, if "Any" then the goal execution will continue without this skill, but if "Must" then the entire goal is rejected then goes to exit.

The other components of the *skill-agent* are similar to any standalone application design for a particular purpose. It is important to highlight that DMMAS does not imposes any particular structure for its *skill-agent.* But it emphasises on the accept-or-reject component, the goal plan, and the *skill-agent* functionality to be defined in the ontology schema. Figure 4.32 illustrate the accept-or-reject components in the *acceptReject* class diagram. The goal plan class depicted by the *checkGoalPlan* class and the functionality is defined in the ontology schema.

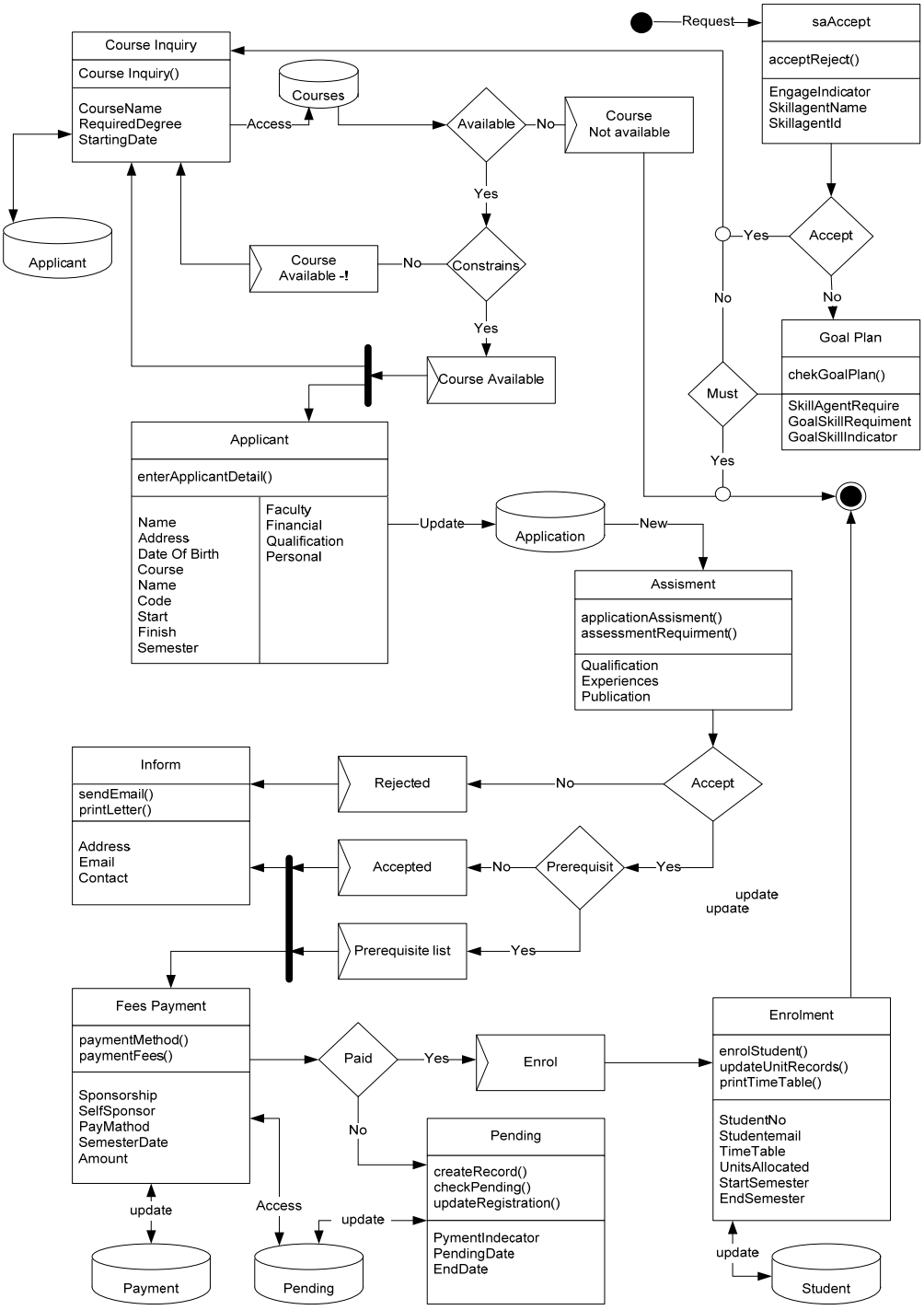

Figure 4.32: *Skill-agent* detailed design diagram.

The Professional-agent detailed design: The *professional-agent* is the goal expert and is responsible for the goal achievement within the resources allocated for this purpose. Each *professional-agent* encompasses the goal resources consisting of goal execution plan, goal XML schema, goal controller, *skill-agent*s team, skill SQL, alternative plan (optional), and request *skill-agent* component.

Objective: To detail designed the *professional-agents* intra structure and their interactions.

Resources: professional-agents descriptor, *professional-agents* architecture diagram, alternative plan descriptor and diagram, execution plan descriptor and diagram, and goal XML schema.

Potential techniques: Study the *professional-agents* descriptor and architecture then develop the goal component in respect of the goal domain and goal XML schema. Using the goal domain set up the develop skill SQL class. Set the goal controller class messages and methods. If there is an alternative plan then use the alternative descriptor to set the alternative plan class. Create and establish the access to the execution plan based on the goal Id and goal name. Create a *skill-agent* list from the execution plan, *skill-agent* search model, and *skill-agent* request. All the messages and other activities are available in the diagram including the linking structure.

Output: The *professional-agent* diagram depicted in Figure 4.33 started from the user goal selection to a predefined system goal. The goal defined by XML schema in hierarchal organisation structure. Thus to make the goal parse able subsequently the system convert this goal to known elements in order to use it as keyword in the SQL statement then infer the *skill-agent* functionality ontology. The main output of the *professional-agents* is the *skill-agent* list which guides the system to achieve the goal. The *professional-agents* diagram shows the interaction with the execution plan in order to check each *skill-agent* existence if "Must" or not. When the *skill-agent* list established this means all agents are committed and reserve under this goal execution. In other words all these *skill-agent* are under adoption and ready for execution.

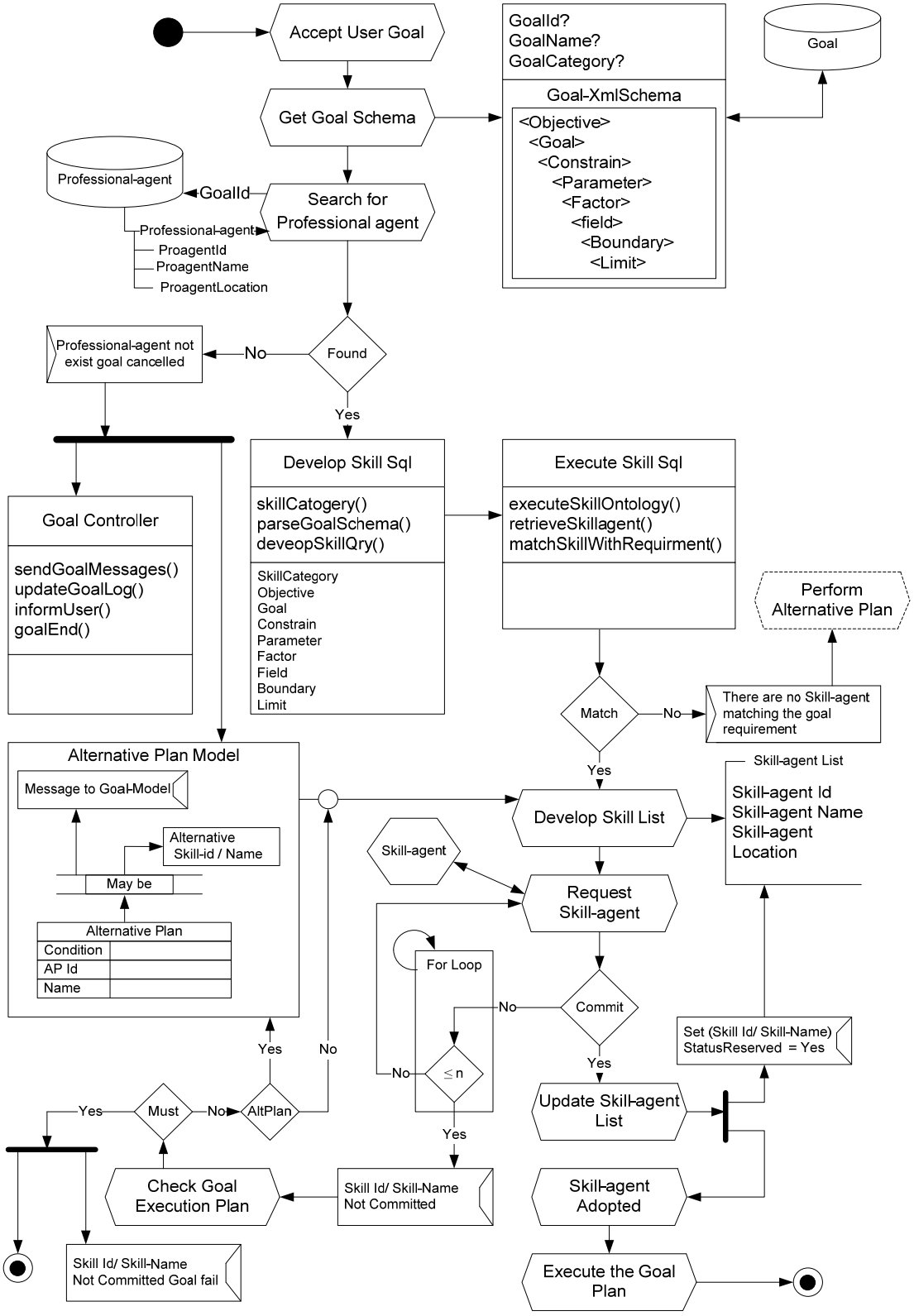

Figure 4.33: *Professional-agents* detailed design diagram.

System goals XML based structure: The DMMAS analysis and design consider the system goals as the key component that triggers the selection of the *skill-agent*s. The

goals must be understood by the system in the form of parse able, subsequently used it in the *skill-agent* ontology search process. For this reason DMMAS propose to define each goal in the system into XML based organisational structure that can accommodate the goal in XML elements (refer to Appendix A). The XML and XML Schema are the most suitable implementation technology for such purposes. Thus XML has compatibility with the ontology languages OWL and is characterised by open expressiveness structure tags.

Objective: To convert the system goals to XML based goal schema in order to make it parse able, knowledgeable subsequently used in the search process.

Resources: Goal descriptor, data-used and data-produced model, domain ontology high level model.

Potential techniques: The system goals schema develops in a straight forward way. The schema has predefined elements to be described by the developer. The proposed schema elements are not mandatory. If the schema elements are inadequate to define the goal constraints it can be amended to convey the goal constraints sufficiently.

Output: Figure 4.34 shows the structure of the system goal schema defined in XML document and XML Schema (for describing the XML document). The schema defined in a set of rectangular boxes where each box defines one XML element divided into three areas. The top rectangle represents the element tag and the two bottom boxes are to specify element value and element data type in XML Schema format. The goal schema is divided into three main parts. The schema header holding the goal identification and name are followed by the second part the domain, goal, and constraint which represent the goal characteristics and the third part, representing the goal constraints structure. Since data is the extensively used constraint in many applications, the schema extends the limit element into date-from and date-to in case the user goal embrace dates attribute.

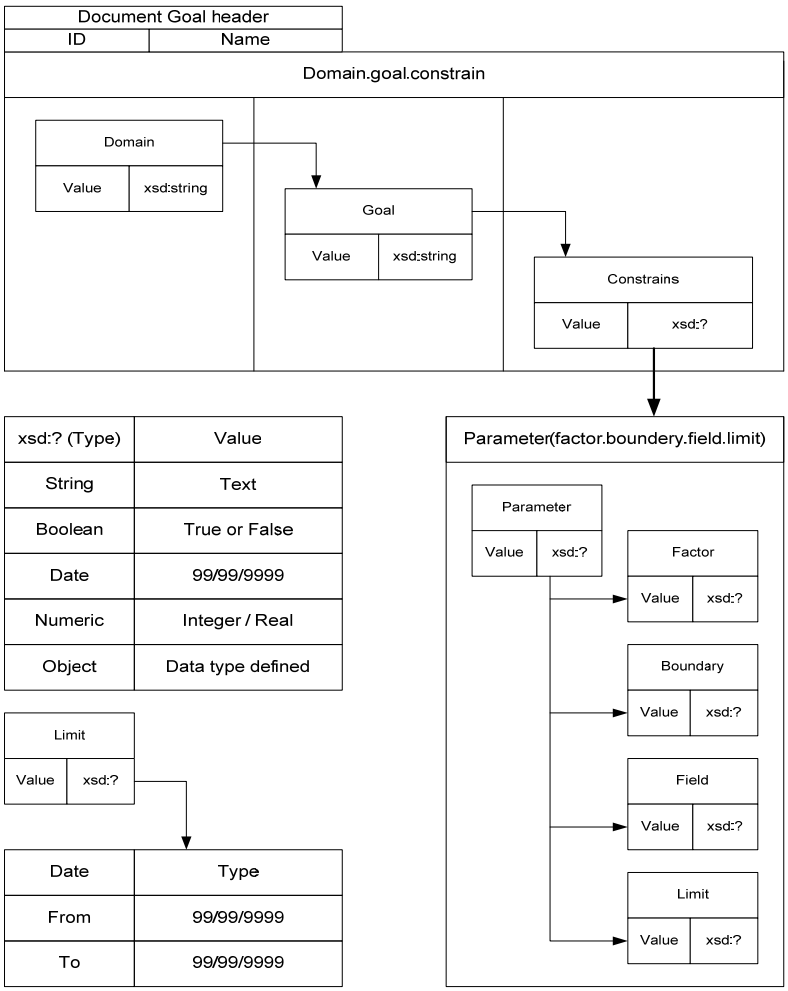

Figure 4.34: XML Schema for system goals.

The goal execution plan: The goal execution plan is the component that will access the goal plan database through built in SQL statement then retrieve the goal execution record (refer to Figure 4.31) along with the SQL statement defined in Figure 4.35 (refer to Appendix C for the results and database definition). Each goal must have a plan associated with it and explain the steps that the system must follow to perform that goal.

Objective: Goal execution plan detailed design diagram.

Resources: Goal execution descriptor.

Potential techniques: Reading the goal execution descriptor build the execution plan database. To get the goal, use the "*goal-ID*" and access the goal database. Use the goal "*ID*" to access the goal execution plan database and fetch the associated plan. When both the execution plan and the goal plan are matched send a request for the *skill-agent* follow by checking the obligation. When the

obligation completed, then set up the execution list in the same execution order defined in the plan.

Output: The execution plan detail design depicted in Figure 4.33. The tasks can be broken down further if the tasks are not directly implementable. However the output of the execution plan component is the execution list indicated by the square notation in the bottom with three main elements. The agent name, the execution order, and the obligation all represent one execution transaction in the overall plan. The number of the transactions must be equal to the number of the *skill-agent*s participating in the plan or required to perform the goal.

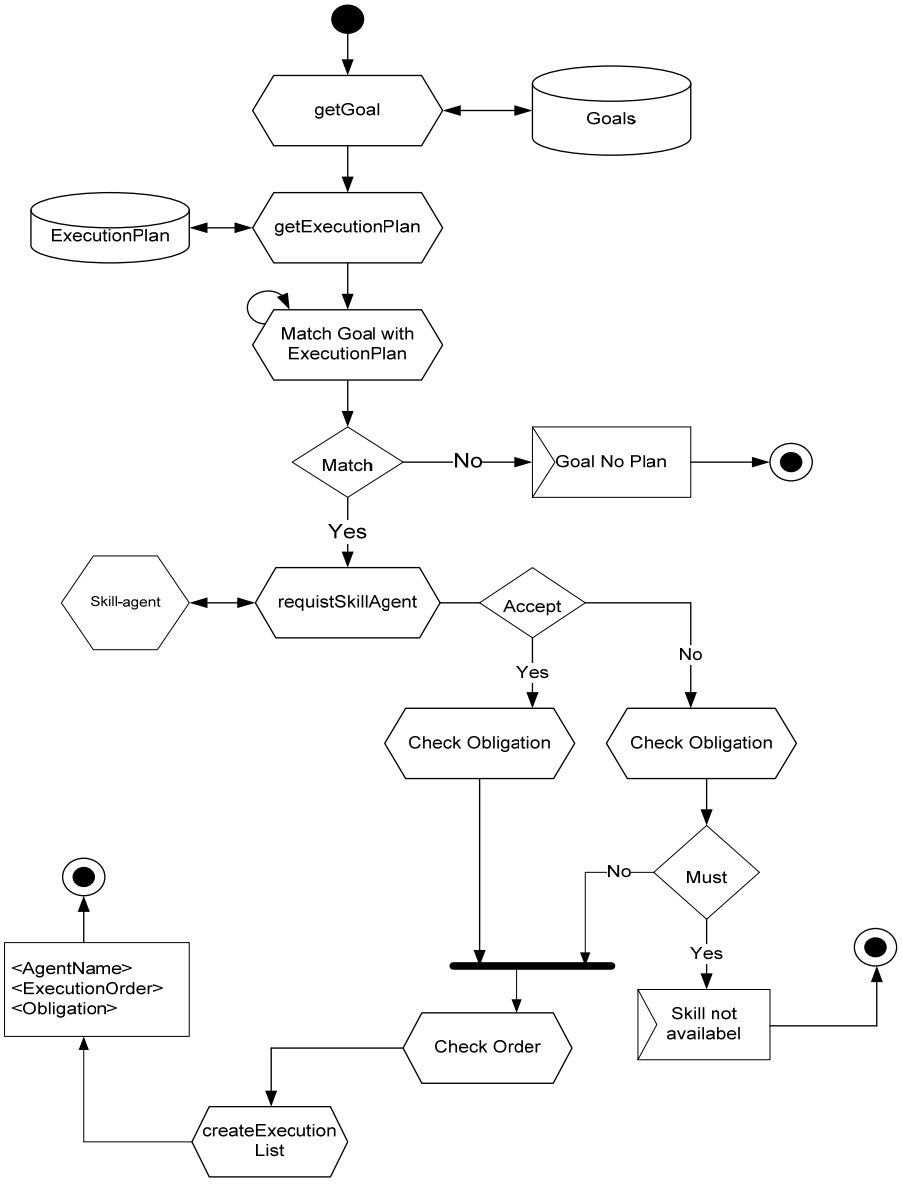

Figure 4.35: Goal execution plan detailed design.

Goal execution component database: In an open distributed heterogeneous operation environment where the *professional-agents* pursue to select the appropriate set of *skill-agent*s to achieve the system goal, it must be equipped with particular guidance and information about the components that will manage the *skill-agent*s team. For example, the alternative plan in case there is a skill failure, the goal XML document, the execution plan, and the *professional-agents* are linked together to form one informational source associated with the goal. Figure 4.36 illustrates the goal execution components database relationship and the SQL segment retrieves the data record. This database schema contains the rules and location of each component (for further details refer to Appendix C).

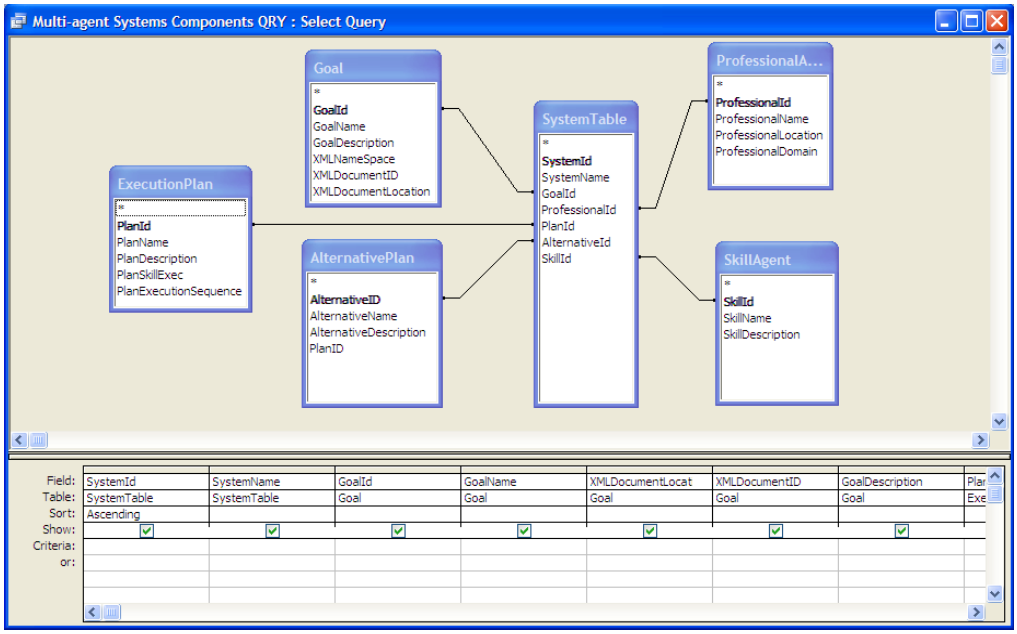

Figure 4.36: Screenshot for goal execution database of *Achieving PhD* example.

```
SELECT SystemTable.SystemId, SystemTable.SystemName,
        Goal.GoalId, Goal.GoalName, Goal.XMLDocumentLocation, Goal.XMLDocumentID,
        Goal.GoalDescription, ExecutionPlan.PlanId, ExecutionPlan.PlanName,
        ExecutionPlan.PlanDescription,
        ProfessionalAgent.*,
        AlternativePlan.AlternativeID, AlternativePlan.AlternativeName, AlternativePlan.AlternativeDescription

        FROM    ExecutionPlan

                INNER JOIN      (SkillAgent
                INNER JOIN      (AlternativePlan
                INNER JOIN      (ProfessionalAgent
                INNER JOIN      (Goal
                INNER JOIN      SystemTable  ON Goal.GoalId = SystemTable.GoalId)
                                        ON ProfessionalAgent.ProfessionalId =
                                SystemTable.ProfessionalId)
```

```
                                        ON    AlternativePlan.AlternativeID =
                                SystemTable.AlternativeId)
                                        ON    SkillAgent.SkillId = SystemTable.SkillId)
                                        ON    ExecutionPlan.PlanId = SystemTable.PlanId

                ORDER BY SystemTable.SystemId;
```

Figure 4.37: Example for SQL statement to retrieve user goal execution table.

Goal definition in XMLSchema (Example):

```
<?xml version="1.0" encoding="UTF-8"?>

        <SystemName>HEDMAS</SystemName>
        <SystemDomain>Education</SystemDomain>
                <Goal>
                        <GoalDomain>Education</GoalDomain>
                        <GoalId>ACHV-1</GoalId>
                        <GoalName>Acheiving PhD Degree</GName>
                        <GoalLocation>C:\acivingPhD\achv-1</GoalLocation>
                        <GoalSkillS>
                                <Skill-1>Enrolment</Skill-1>
                                <Skill-2>Research</Skill-2>
                                <Skill-3>Thesis></Skill-3>
                        </GoalSkillS></GoalSkill>
                        <GoalConstraints>
                                <GoalParameters>
                                        <Boundary>Information Science and
                                                Engineering</Boundary>
                                        <Subject>PhD</Subject>
                                        <Kind>Course</Kind>
                                        <Item>352AA</Item>
                                        <Limit>
                                                <DateFrom>17/02/2011</DateFrom>
                                                <DateTo>18/10/2014</DateTo>
                                        </Limit>
                                </Parameters>
                        </GConstrain>
                </Goal>
```

Agent UML (AUML) interaction diagram: Through the DMMAS analysis and design phases the AUML interaction diagram is used to represent the system agents communication and protocols. The AUML interaction diagram is FIPA standard. On the other hand for the protocols, ACL and KIF is recommended (Fasli, 2007). However, since DMMAS modelling is not interaction oriented therefore the protocols, and the message syntax are not within the research interest, instead the AUML interaction diagram is presented.

Objective: To accommodate the internal and the external agent interactions.

Resources: The system architecture design, *professional-agents* architecture, and *skill-agent* architecture.

Potential Techniques: Study the system requirements and determine the detail design level to explore the system and make it understandable to the implementation process. From that perspective survey the messages and type of messages that satisfy system interaction design. It is important to study the syntax of the AUML notation and for this purpose refer to the AUML website and a summery for the notations and diagrams in Appendix D.

Output: Figure 4.36 depicts the interaction diagram between three activities within the *skill-agent* "*Enrolment*" example. The diagram indicates to multi decision and conditional stats. For example if not paid then check if it is on hold or exceeds the waiting time. The structure of the AUML interaction diagram is approved to handle the all the agent emergent conditions in one diagram with multiple nested functional frames (refer to Appendix D).

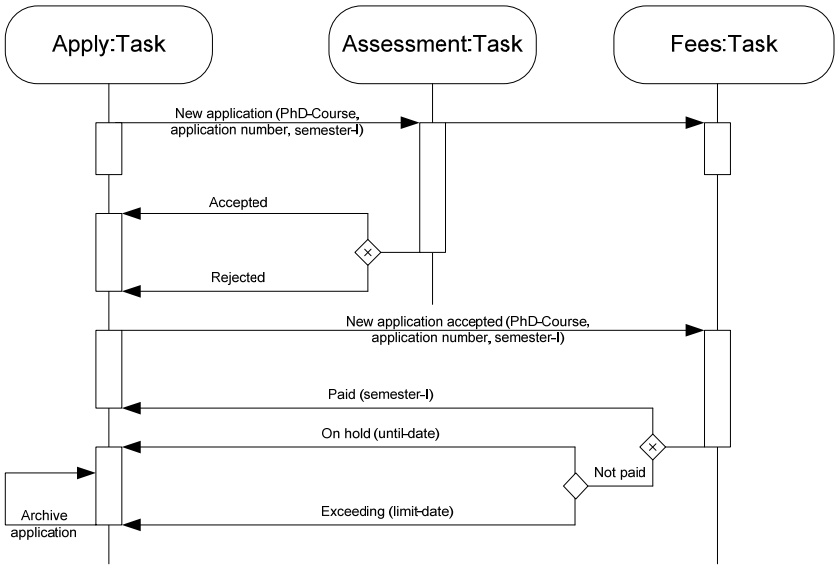

Figure 4.38: AUML interaction diagram for *skill-agent* "*Enrolment*" example.

Design Artefacts: The DMMAS design phase is divided into two phases complementing each other, the architecture design and the detailed design. In both phases the modelling processes customise the traditional software system design artefacts and ontology design artefacts and fit them together in one design model. It is a new experience to merge these two approaches in one development methodology. To reduce the design complexities DMMAS introduced a list of new diagrams and notations as part of its practice where the diagrams can simplify the design process.

Table 4.4 lists the output (components) of DMMAS design for components validation against completeness.

Step	Architecture Design	Detailed Design
1	*Skill-agent* task	Notation
2	*Professional-agents* architecture	*Skill-agent* internal structure
3	*Skill-agent*s search model	*Professional-agents* internal structure
4	Extended execution plan	System goal XML based Structure
5	Goal execution plan	Goal execution plan
6	Ontology of *skill-agent* functionality 1. Information component 2. Identification component 3. functionality component	Goal execution components
7	Search model	AgentUML interaction diagram

Table 4.4: Detailed design phase components.

4.9 Summary

This chapter explains in detail the DMMAS development process for multi-agent systems development methodologies based on new cooperative multi-agent systems architecture incorporating an ontology approach to define agents' functionalities. DMMAS introduce the *skill-agent* and *professional-agents* as a new concept to form MaS dynamic organisational structure. The *professional-agents* representing the system goals and the *skill-agent*s representing the skills are required to achieve the goal. The MaS dynamic structure is formed at system runtime in consequence of the goal requirement and agent adoption strategy between the *professional-agents* (goal owner) and *skill-agent*s (expertise).

This chapter introduces and explains DMMAS new diagrams and notations to familiarise the development process. An example "*Achieve PhD Qualification System*" is used to illustrate some of the DMMAS analysis and design steps. The DMMAS development approach is original and innovative, intending to transfer the multi-agent systems from the static independent to the dynamic cooperative. This claim will be proven in the next chapter, which will examine the DMMAS applicability by developing a real world e-commerce case study Travel Agency System (TAS).

Chapter 5 Travel Agency System: A Case Study

5.1 Introduction

This chapter examines the use of DMMAS as a proof of concept in building open cooperative multi-agent systems that operate in distributed heterogeneous software environments. A Travel Agency System (TAS) case study is developed by applying DMMAS modelling techniques. The nature of the TAS case study is in the DMMAS problem domain and in addition, TAS is characterised by coordination constraints that require sophisticated software techniques amenable to multi-agent systems problem solving strategies. In Chapter 4 the DMMAS analysis and design was explained in detail. This chapter illustrates the deployment of DMMAS analysis and design phases throughout the TAS development process. The process emphasises the ontology modelling for system goals definitions and skill-agent functionality.

5.2 Travel Agency System

To investigate and evaluate the potentiality of DMMAS analysis and design processes the research deploys DMMAS to develop a Travel Agency System (TAS) in a real world application from a commercial domain. A multi-agent systems problem solving strategy is an appropriate solution to TAS constraints. TAS consists of distributed standalone subsystems of car rental, hotel booking and flight reservations that operate in open distributed environments. These subsystems require coordination and integration to achieve the system goal cooperatively. In this regard TAS has criteria in terms of specifications and constraints that fit with open distributed multi-agent systems concepts. On the other hand, DMMAS methodology has been designed to function for such types of problem classes.

The travel industry is increasingly becoming internationalized. Airlines, accommodation providers and tour operators control the market, but these service providers face high costs in communication and personnel. Using today's IT technology and with the availability of the Internet services, the user himself can search the internet and access different web services then within a set of constraints

and preferences, the user is able to coordinate the results of individual web services in order to arrange his travel. Thus the sites (the systems) are physically independent and belong to different vendors and each vendor has its own target and system development standards. In large vast business companies for example airlines, TAS applications belong to one business institution and could be managed by different departments from different geographical locations with each system developed independently. Despite the development circumstances there is a functional requirement to integrate these individuals systems to provide a total solution to the user needs and isolate the user from the system coordination. AlHashel and Mohammadian (2008) illustrate the advantages of the multi-agent system concepts to create a cooperative system to isolate the user from the system to system coordination.

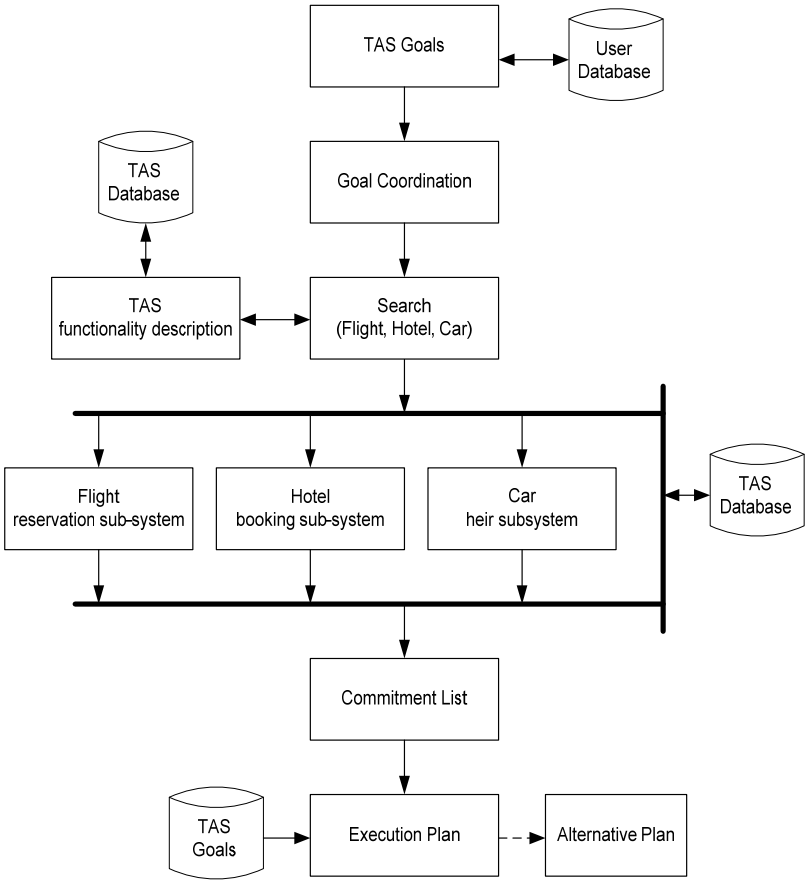

Figure 5.1: TAS main components.

Figure 5.1 illustrates the main components of TAS where the thick lines define the three main subsystems as independent components with same interaction level to the other system components. If the TAS subsystems car rental, hotel booking, and flight

reservation belong to different vendors and are built by different developers is it possible to create an emergent integration to answer the user goals combinations and constraints. For example, the user needs a Flight from Sydney to Bangkok on 23/04/2010 and needs to rent a car within his arrival to Bangkok at 24/04/2010 at 09:30 and a room in Bangkok hotel on the same day for five days, then on 28/04/2010 leaves Bangkok for Sydney, then travelling to Canberra to arrive on 29/04/2010 at 7:20 AM. Currently this type of problem is either solved by the user himself through accessing different web services and using his time and effort to coordinate between different system responses. Alternatively the user asks a travel agency to plan this trip sector for him. The travel agency will carry out the process similarly to the user with some additional business privileges. Furthermore, in most cases the travel agency will ask the customer to give some time to plan his preferences.

In such a computational environment, the multi-agent systems based problem solving concept has great potential for accessing an open distributed system then to search and integrate these systems using the required set of skill-agents to answer the user query on a travel package he/she selected. TAS is designed based on multi-agent systems concepts to obtain the emergent integration according to the user preferences. This class of problem domain is in the range of DMMAS potential where all the other existing multi-agent systems development methodologies are out of this problem class. However, to demonstrate this claim find below the requirements that the TAS can achieve when developed under the DMMAS design and analysis concepts:

- automate the travel process to provide various travel facilities that meet the customer needs as per the facilities combinations and constraints shown in the Table 5.1.
- the travel arrangement is fully automated and the travel sector is performed by the system to the degree the user is isolated from the systems coordination between flight, hotel, and car facilities;
- the system is scalable, therefore adding or removing skill-agents can be done without any system runtime exceptions;

- the system is distributed and consists of three independent subsystems, i.e. Flight reservation, Hotel booking, and car rental. Each system is maintained from different locations;
- the system must be designed using open architecture and can interact and operate in an open environment such as the Internet and web services.

		Facilities		
Constraints		Flight	Hotel	Car
Place	From			
	To			
Date	From			
	To			
Type				

Table 5.1: TAS facilities.

Assumptions: The research considers the TAS case study as a prototype system used to investigate the potential of DMMAS design approach. The emphasis of this experiment is not on designing an entire application system (reservation system or car rental system or hotel booking system) thus analysis and design of such a system is beyond the research scope. The main purpose is to demonstrate the potential of DMMAS development processes in analysing and designing open cooperative multi-agent systems. Therefore the focus is on building the system architecture and prototype of the system components and their connectivity forming the system concepts.

DMMAS and TAS Problem Class: Formalising a general multi-agent systems development methodology for all types of problem classes is not yet available. There are some reasons why existing MaS developments are domain oriented. Firstly, there is no general multi-agent systems architecture that can embrace all the existing problem domains (Nicholas et al., 1998). Secondly, there are wide ranges of different areas where the software agent specifications can be applied. Third, an agent-oriented system is characterised by sophisticated concepts that make it a total software solution.

"*Agents are being used in an increasingly wide variety of applications — ranging from comparatively small systems such as personalised email filters to large, complex, mission critical systems such as air-traffic control. At first sight, it may appear that such extremely different types of system can have little in common*" (Michael, 1999).

Luck et al. (2004) listed the existing methodologies and identified their application domain area. This list shows that none of the existing development methodologies addressed the TAS problem class as open distributed realising the cooperative approach. This claim is discussed in Chapter 2, and demonstrated in Chapter 3. It is also discussed in a recent survey conducted by Pokahr and Lamersdorf (2008) and a summary evaluation and comparisons presented by Eric et al. (2005).

TAS problem class presents a range of constraints and are application environment that is efficient to consider as a test-bed to assess DMMAS development process. Therefore TAS problem class fits DMMAS problem class characteristics for the following reasons:

1. The TAS case study comprises a coordination process to organise the flight reservation, the hotel booking, and car hire within a trip sector. DMMAS have strong coordination design concepts based on the professional-agent and goal execution plan.
2. TAS problem scenario provides a range of constraints that required advance planning to schedule the best possible options. The DMMAS characterised by the goal XML Schema, and goal execution plan that can link the goal constraints precisely.
3. It needs a sophisticated open architecture search to access and integrate three different systems (Flight Reservation, Hotel Booking, and Car Rental) that can be represented by three agents. The DMMAS two layer architecture of professional-agent and skill-agents integrated by ontology based open search is efficient for this situation.
4. The user goal is changeable. The agents that can achieve the goal are also changeable in parallel with the goal requirements. The changeable goal

requirement can be translated by an agent team formation process. Whenever the goal changed the required set of skill-agents accordingly is reformed. On other words the operations entail a dynamic multi-agent system team formation process. This is in the core concept of DMMAS architecture and is represented by the ability to dynamically reconfigure at runtime to satisfy the goal requirements.

5. Deploying DMMAS to develop TAS problem class will result in a number of different skill-agents teams that can form to satisfy all the system goals. The possible numbers of skill-agent team formation must be calculated to correctly estimate the system dynamic goals reconfiguration. For example, Figure 5.2 show TAS consists of three skill-agents (*Sa*) they are; flight (*f*), hotel (*h*), and car (*c*) and the set of TAS goals (*Sg*) ={f},{h},{c},{f,h},{h,c},{f,c},{f,h,c}

   The total number of skill-agent teams (*At*) = (((Sa)**2) – (Sa-1))

   TAS *Sa* = 3

   The total number of TAS *At* = ((3)**2) – (3 – 1) = 7

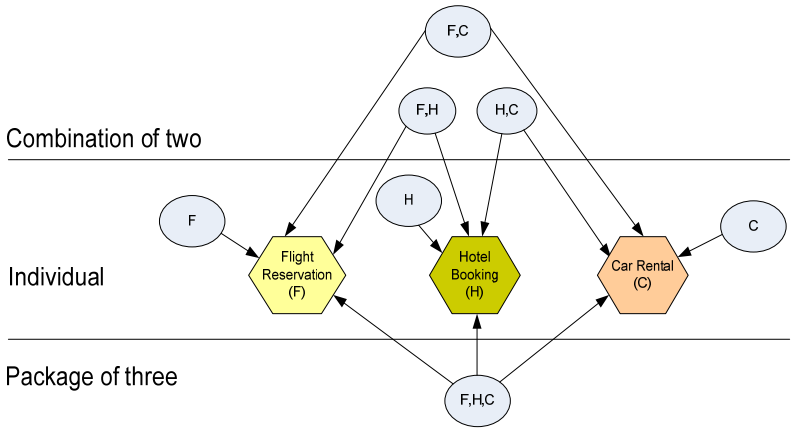

Figure 5.2: Numbers of skill-agents team in TAS.

5.3 TAS Requirement Analysis

As it mentioned earlier in Chapter 4, the first phase in DMMAS is the analysis phase which is divided into two sections. The first section is the system requirement and the second section is the system analysis. The system requirement is divided into two

steps. The first step is the system requirement at the high-level then this is followed by the system requirement goal analysis step. In the system requirements: high level the system actors, system goals, goals scenario, and system context diagram are all specified for TAS multi-agent system.

TAS System Requirements: Specify TAS actors: TAS consists of three main subsystems; i.e. the Flight Reservation (FlightRes), Hotel Booking (HotelBok), and Car Rental (CarRnt). In the initial requirement analysis, DMMAS visualised TAS as a system consisting of three independent subsystems each representing an independent unique service with no intersection between the services. Therefore specifying the system actors at this stage must be based on each system independently (see Figure 5.3). This will help in identifying every actor in the system. In the next step, if there are any similar actors found, they will be grouped together. Thus the analysis processes start to treat the system as one integrated multi-agent system.

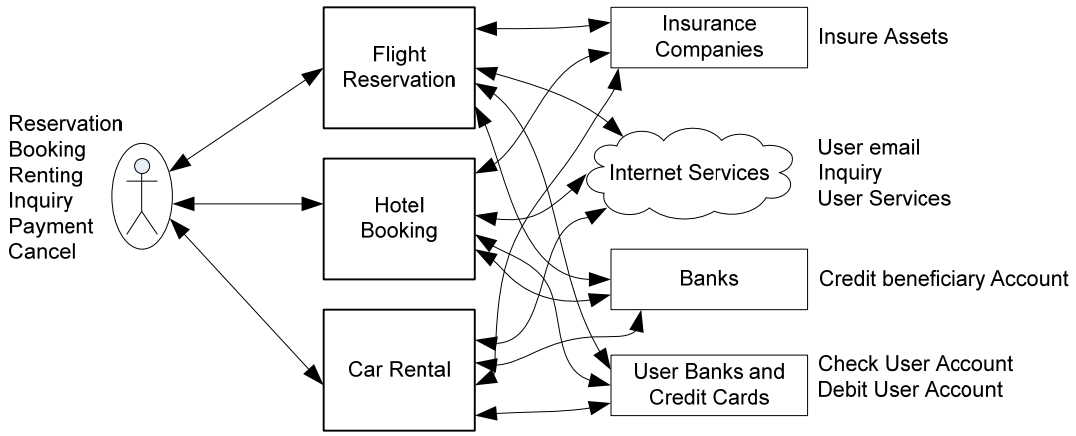

Figure 5.3: High level TAS over view including system external entities.

TAS context diagram: To clarify the system size or the high level entities involved in the system and to determine the application coverage area the DMMAS implement a context diagram which is depicted by the dotted line in Figure 5.3. The diagram dictates that TAS excludes the insurance company, the internet services, the banks, and the user payment institution. The operations are part of the user menu that has corresponding system activities inside the TAS body.

Specify TAS goals: From the TAS documentation scenario the possible three subsystems as in the context diagram; Flight Reservation (FlightRes), Hotel Booking

(HotelBok), and Car Rental (CarRnt) are the high level goals. Figure 5.4 reflects TAS goals including their initial is identification, sequence according to their priority to the users, and the goal names.

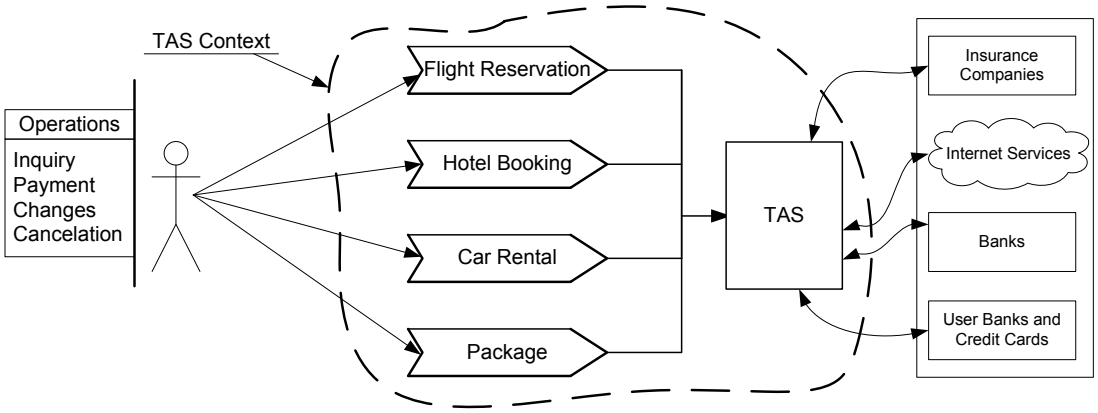

Figure 5.4: TAS context diagram.

TAS goals scenario: Following the DMMAS techniques, each goal must be defined according to the goal descriptor. Since TAS has three goals, these must all be defined. At this high level system analysis stage, the data-used and data-produced are only listed with the expectation this is only to develop the initial goal structure. The three TAS goals descriptors shown by Figures 5.5 (a, b, and c) are implemented as per the explanation in Chapter 4.

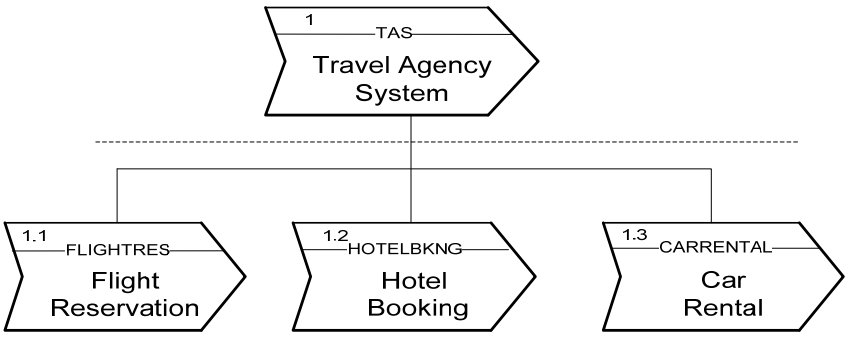

Figure 5.5: TAS goals diagrams.

TAS Goal-Descriptor: Flight Reservation

Goal Id	FLIGHTRES	Selection	Priority	Execution Sequence	I= Individual G= Group R= Random S= Sequential
Goal Name	FlightReservation	Individual	I	R	
		Group	G	S	
Description	Reserve a seat on flight according to the customer preferences				
Verification	seat reserve when payment successful				
Expected Data-Used	Passenger: Name, date of birth SeatClass; Departure: Place, Date, Time, Destination: Payment: CardNumber, Type, ExpiryDate.				
Expected Data-Produced	Reservation: Code; Arrival: Place, Date, Time, SeatClass; Return: Place, Date, Time; Payment: Amount				
Option-1	If seat not available				
Action-1	No reservation made, send message " Flight not available transaction faild"				
Option-2	At reservation available, successful payment is prior, If Payment unsuccessful				
Action-2	Reject request, send message: "Transaction failed payment not approved"				
Option-3	If the main goal is package and flight not available				
Action-3	Check goal priority; If goal priority is "Must" then Action-1, Else send message "hotel not available" and continue processing				

Figure 5.6: Goal descriptor for Flight Reservation.

TAS Goal-Descriptor: Hotel Booking

Goal Id	HOTELBKNG	Selection	Priority	Execution Sequence	I= Individual G= Group R= Random S= Sequential
Goal Name	HotelBooking	Individual	I	R	
		Group	G	S	
Description	To book a room in a hotel according to the customer preferences				
Verification	Inquiry, Room search, Payment, Book a room				
Expected Data-Used	Customer: Name personal details; Arrival: Date, Time; Payment: CardNumber, Type, ExpiryDate				
Expected Data-Produced	Hotel: Name, BookingNumber, LocationAdress, NumberOfNights; CheckIn: Date, Time; CheckOut: Date, time; Payment: Amount.				
Option-1	If the main goal is Hotel booking only and room not available				
Action-1	Reject request, send message: “Hotel not available transaction unsuccessful”				
Option-2	Unsuccessful payment				
Action-2	Reject request, send message: “Transaction failed payment not approved”				
Option-3	If the main goal is package and hotel not available				
Action-3	Check goal priority; If goal priority is “Must” then execute Action-1, Else send message “hotel not available” and continue processing				

Figure 5.7: Goal descriptor for Hotel Booking.

TAS Goal-Descriptor: Car Rental					
Goal Id	CARRENTAL	Selection	Priority	Execution Sequence	I= Individual G= Group R= Random S= Sequential
Goal Name	CarRental	Individual	I	R	
		Group	G	S	
Description	Rent a car according to the customer preference				
Verification	Inquiry, Receive payment, Reserve a car, Inspection, Car hired				
Expected Data-Used	Driver: Name, LicenceNumber, PersonalDetails; Car: Model, Type, Year Payment: cardNumber, Type, ExpiryDate				
Expected Data-Produced	ReservationNumber, Car: Model, Type, Model; Hired: PlaceAddress, Date, Time, numberOfDays; Return: PlaceAdress, Date, Time; Payment: Amount				
Option-1	Unsuccessful payment				
Action-1	Reject request, send message: “Transaction failed payment not approved”				
Option-2	If the main goal is car rental only and car not available				
Action-2	Reject request, send message: “Car not available transaction failed”				
Option-3	If the main goal is package and car not available				
Action-3	Check goal priority; If goal priority is “Must” then Action-2, Else send message “car not available” and continue processing				

Figure 5.8: Goal descriptor for Car Rental.

5.4 TAS Specification Analysis Phase

The first step in the system specification analysis is to identify the professional-agents, the skill-agents, then set up the execution plan and the alternative execution plan. These components are essential to construct the initial system specification.

TAS professional-agents: As it has been stated in Chapter 4 the professional-agent is the agent representing the user goal in a particular domain of skills. In the TAS case study there is only one professional-agent, i.e. the travel agency depicted by Figure 5.9 as one consolidated unit. This travel agency is the key goal agent that can embrace the set of skills to satisfy the user goal within the travel agency domain consisting of flight, hotel, and car facilities. Regardless the user preferences or if he keeps changing

his/her goal the travel agency professional-agent will gather the required skills to satisfy the goal.

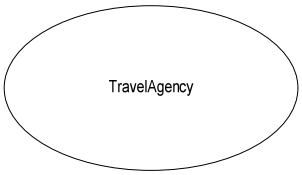

Figure 5.9: TAS Professional-agent.

TAS skill-agents: To identify the skill-agents that compose the TAS under layer, refine the initial system scenario. Form system scenario refinements process flight reservation, hotel booking, and car rental as depicted in Figure 5.10 have been identified and allocated to form the required individual skills that satisfy all the user goals. The identification process is straightforward and the number of skills-agent team combination (t) that can be formed to satisfy all the goal possibilities can be calculated by following the example mentioned earlier in the problem class; t = (((number of the skill-agents in the system)**2) – (number of the skill-agents in the system) – 1)) = ((3)**2) – (3-1) = 7 teams.

Figure 5.10: TAS skill-agents.

Figure 5.11 illustrates the verifications graph of TAS goals in corresponding to the system skill-agents. This verification graph is used to confirm the number of available goals. In fact it used as a checking tool specifically in large scale system.

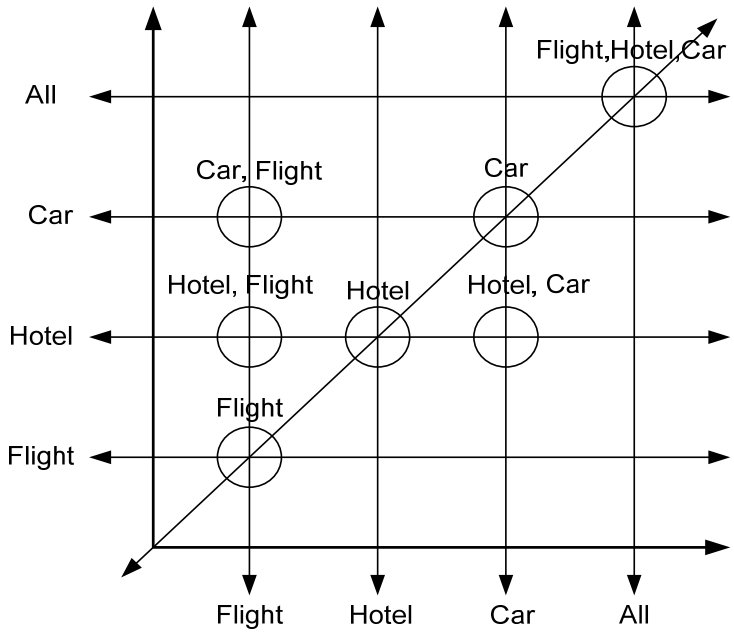

Figure 5.11: TAS skill-agent team graph.

TAS professional-agent execution plan: At this stage the relation between the TAS professional-agent (Travel Agency) and the three skill-agents (Flight, Hotel, and Car) are established from the execution perspective. The execution plan is the registry that holds the information about the execution in terms of: importance (must be executed or optional), sequence (sequential, random), and team (individual, group). The goal descriptor is the key input source to develop the execution plan.

TAS Goal Execution Plan Descriptor: Travel Agency

Professional-AgentId : TRVLAGNCY	Professional-Agent Name : TravelAgency

Professional-Agent
Description : Travel agency goal is responsible to create a travelling packages according to the customer preferences in combination of Flight, Hotel, and Car reservation arrangements

Execution Plan

Alternative Plan Id: TRVLALTPLN
Alternative Plan Name: TravelAltrPlan

Goal-ID	GoalName	Skill Agent Id	Skill Agent Name	Plan Id: / Plan Name	Team: Group	Team: Individual	Sequence: Sequential	Sequence: Random	Priority: Must	Priority: Optional
GLFLHT	Flight	SKFLIGHT	FlightReservation	PLFLHT / FlightPln		●	●		●	
GLHOTEL	Hotel	SKHOTEL	HotelBooking	PLHOTEL / HotelPln		●	●		●	
GLCAR	Car	SKCAR	CarRental	PLCAR / CarPln		●	●		●	
GLFLHTHTL	FlightHotel	SKFLIGHT SKHOTEL	FlightReservation, HotelBooking	PLFLHTHTL / FlightHotelPln	●			●	●	
GLFLHTCAR	FlightCar	SKFLIHGT SKCAR	FlightReservation, CarRental	PLFLHTCAR / FlightCarPln	●			●	●	
GLHTLCAR	HotelCar	SKHOTEL SKCAR	HotelBooking, CarRental	PLHTLCAR / HtelCarPln	●			●	●	
GLFULPKG	FlightHotelCar	SKFLIGHT SKHOTEL SKCAR	FlightReservation, HotelBooking, CarRental	PLFULPKG / FullPKGPln	●			●		●

Figure 5.12: TAS goal execution plan descriptor.

TAS goals alternative plan: Figure 5.13 depicts the alternative plan used as a backup in case any of the TAS goals experience a plan implementation failure (it is not system exceptions). In this case TAS can use the alternative plan as a recovery option. In TAS the recovery options are estimated to be simple and for this reasons and to avoid system implementation complexity all the goals cases are listed under same actions that send a message to the user.

Alternative Plan Descriptors: Travel Agency System

Alternative Plan Id: APTRVLAGNCY Alternative Plan Name: TravelAgencyAltrPlan	Backup For: PATRVLAGNCY

Goal-ID	Alternative Plan ID	Alternative Plan Name	Action ID	Action Name
GLFLHT	APFLHT	FlightAltPln	ACFLHT	FlightAction
GLHOTEL	APHOTEL	HotelAltPln	ACHOTEL	HotelAction
GLCAR	APCAR	CarAltPln	ACCAR	CarAction
GLFLHTHTL	APFLHTHTL	FlightHotelAltPln	ACFLHTHTL	FlightHotelAction
GLFLHTCAR	APFLHTCAR	FlightCarAltPln	ACFLHTCAR	FlightCarAction
GLHTLCAR	APHTLCAR	HtelCarAltPln	ACHTLCAR	HtelCarAction
GLFULPKG	APFULPKG	FullPKGAltPln	ACFULPKG	FullPKGAction

Figure 5.13: TAS alternative plan descriptor.

TAS organisation structure: The diagram in Figure 5.14 explains the relationship between TAS goals and their recovery plan for every individual goal in the system. Since the actions which flow from the alternative plan have similar actions which send a message “Unable to achieve the goal transaction failed” therefore it is not necessary to expand the diagram of the alternative plan actions.

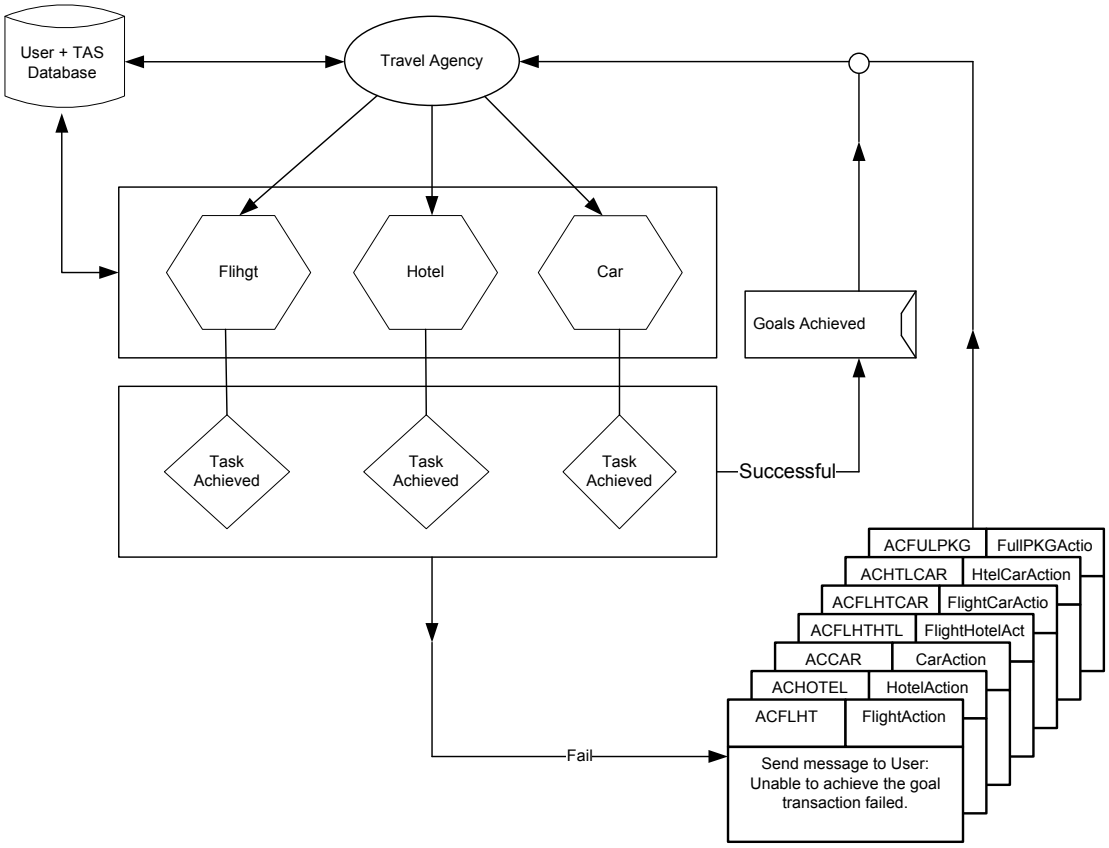

Figure 5.14: TAS organisation structure.

TAS data model (Data-used and Data-produced): The data-used and the data-produced are very important components to determine the TAS ontology domain which will be exposed in the next phase. The three TAS skill-agents (flight, hotel, and car) are defined in the diagrams represented in Figure 5.15, Figure 5.16, and Figure 5.17 consecutively. At this stage the data does not necessarily include every attribute in the system. Instead the building block is started and the in-status and out-status must be specified at this level. Thus to use this term in the ontology model for example, the flight reservation is In-Status = Customer, the Out-Status = Passenger, the hotel booking, the In-Status = Customer, the Out-Status = Dweller, and the car rental the In-Status = Customer and the Out-Status = Rental. These terms must be distinguished from each other particularly the out-status. This specifies each skill-agent with a unique functionality (services) subsequently the required skills for the goal achievement does not overlap and the ontology schema for each skill is defined and recognised independently.

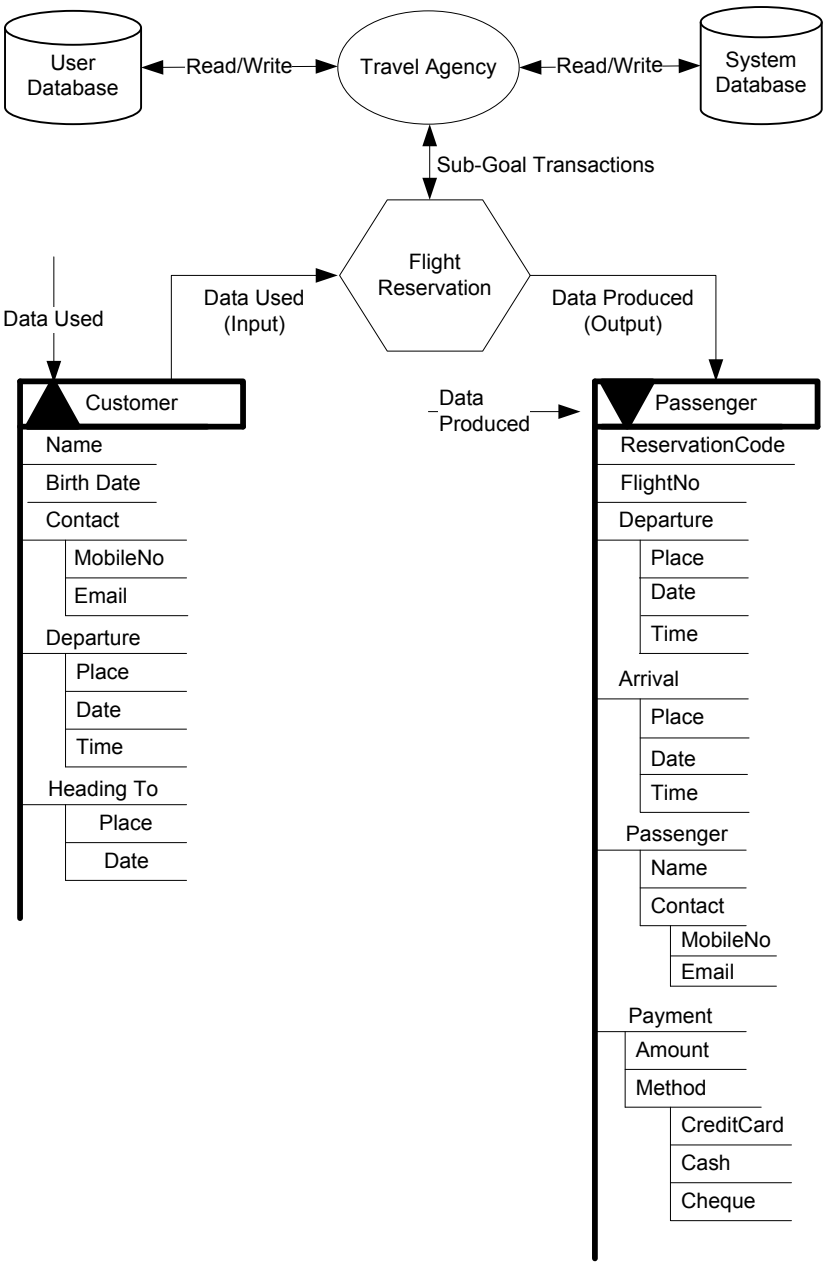

Figure 5.15: Flight Reservation skill-agent data model.

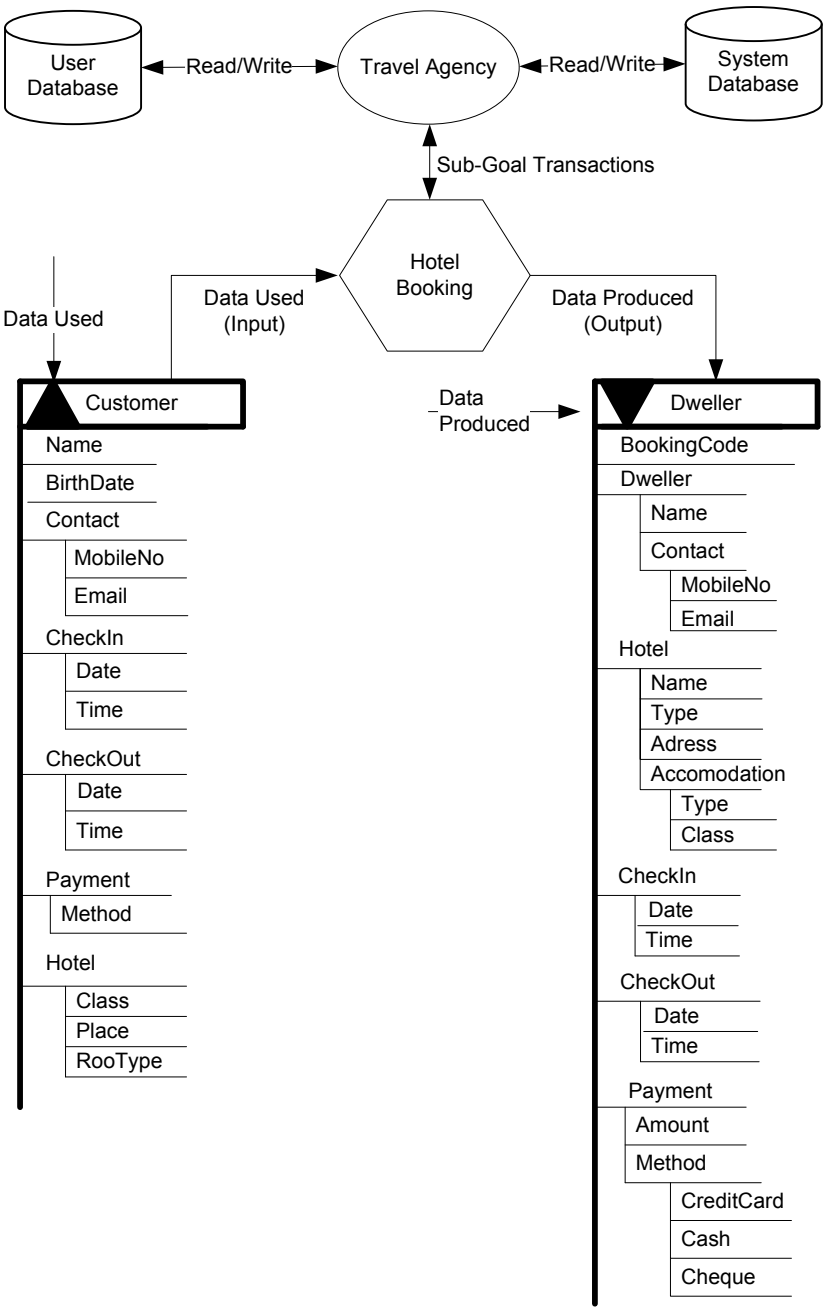

Figure 5.16: Hotel Booking skill-agent data model.

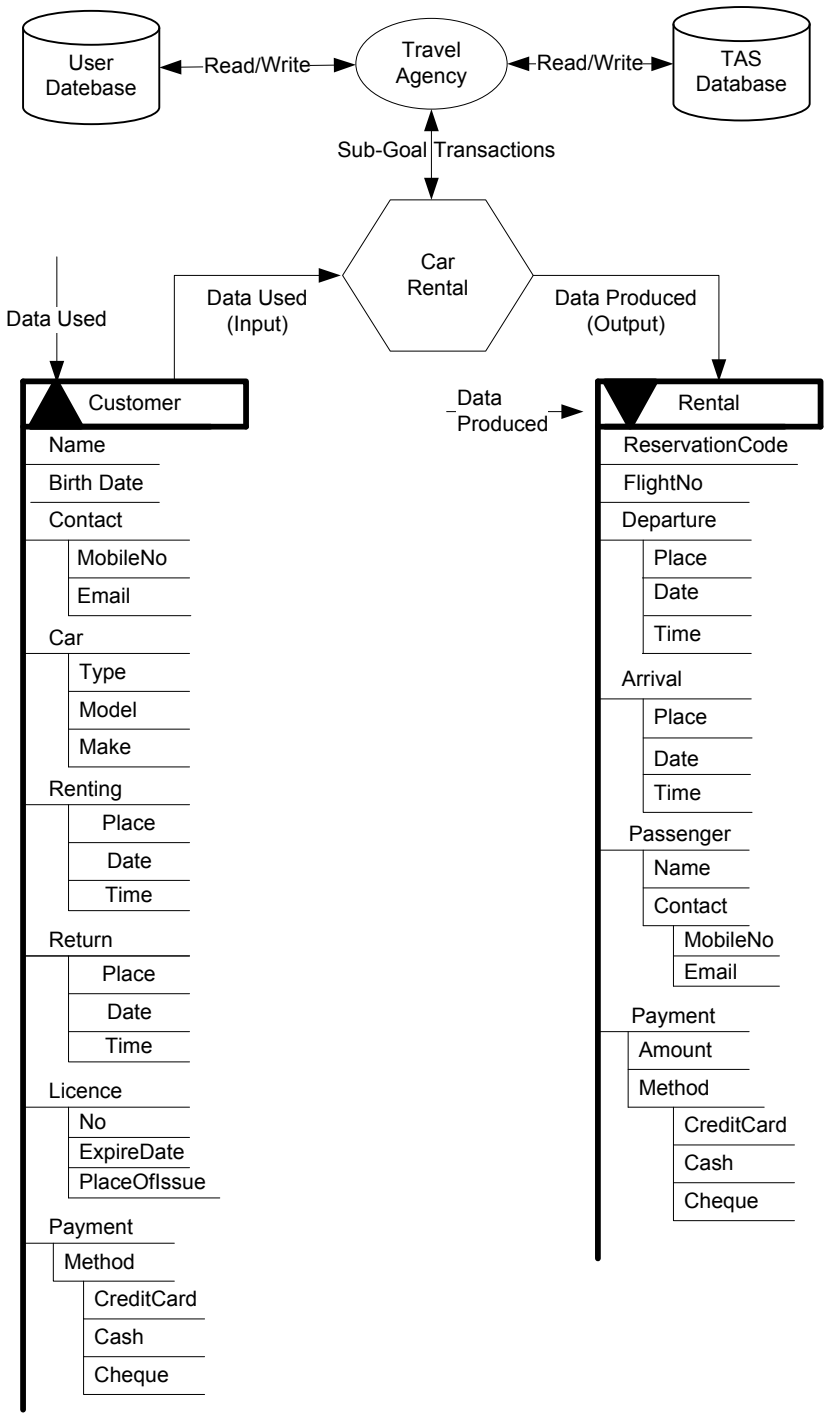

Figure 5.17: Car Rental skill-agent data model.

Identify TAS ontology domain: DMMAS frameworks guide the development of an ontology based search model for the skill-agent functionality. The primary steps to achieve this purpose are to identify the ontology domain. Thus the ontology is open ended and can be defined beyond the scope of application requirements. Identifying the TAS ontology domain and defining the ontology hierarchal levels must be decided at this analysis stage.

To identify the ontology domain follow the steps mentioned in Chapter 4 under ontology analysis section (page 121) then verify the data-used and data-produced model with the TAS initial scenario. Therefore the TAS ontology domain is identified in "travel" domain. Figure 5.18 defines the first step in defining TAS ontology domain, i.e. ontology scope descriptor. The ontology scope for TAS defines the aim of the ontology and usage then benefits or how it will use this ontology. This step establishes a basic foundation for TAS ontology depth or the length of the concepts tree and identifies if this ontology is part of another definition, in which case, who will use this ontology definition. Based on these inputs the TAS ontology scope is developed as in Figure 5.18.

Ontology Scope: Travel Agency System		
Aime	To Ontologies Travel Agency System skill-agent functionalities consist of three skill-agents: Flight reservation, Hotel booking, and Car rental	
Usage	Sharing	Non
	Reusing	Non
	Part of	Non
	Other	Non
	Description	Travel domain consist of: Flight reservation, Hotel booking, and Car rental skill-agents.
Beneficiary	Travel Agency System (TAS)	

Figure 5.18: TAS ontology scope.

TAS Domain Ontology High-level Diagram: The TAS domain ontology as represented by Figure 5.19 provides the first step in building the target ontology that can answer the query about the skill-agent functionality. The strategy to achieve this goal is to view each skill-agent as a process that has input and output to reflect the skill-agent internal processing. Following DMMAS methodology each skill-agent must have both input-statuses (single criteria that represent the input before processing and output-status (a single criteria in objective form) that represent the

output-status after processing. The different space between the input and output is called transition (changes). However, Figure 5.19 depicts the TAS skill-agents (flight, hotel, and car) with proposed input and output status. There are two important conditions in defining the input and the output. First, the input must be different from the output otherwise the process is neutral. Second, there are no two skill-agents with similar input and output; otherwise both offer the same functionality.

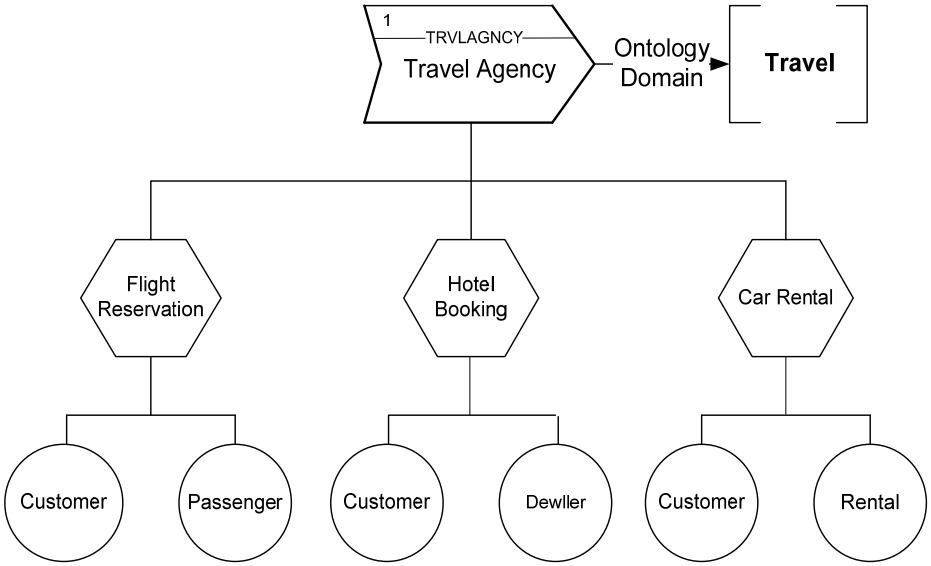

Figure 5.19: Input-status and Output-status within each TAS skill-agent.

Specify TAS domain ontology scope: The domain and scope ontology descriptor as illustrated in Figure 5.20 is an important component to initially establish the query or the ontology inference scenario. Specifying the system queries can help determine the ontology scope. It acknowledges the ontology criteria involved in the skill search to form the skill-agent team required to achieve the goal. The descriptor informs the developer if there is an ontology segment that can be used or shared with other applications. It also guides the ontology development process to work in parallel with the requirements.

Domain and Scope Ontology Descriptor: Travel Agency System					
Aime	To Ontologies Travel Agency System skill- agent (flight, Hotel, and Car) functionality, and make it knowledge-based searchable		Owners	Travel Agency System	
				PATRVLAGNCY	
Usage	Sharing	Non			
	Reusing	Available for future developments			
	Part of	Non			
	Other	Non			
Beneficiary	Description	Travel Agency System Skill-agents (Flight Reservation, Hotel Booking, Car Rental)			
	Others	Non			
Domain	**Travel**		Exist	Full	OWLOntologyLibrary
			NO Yes	Partial	
Queries	1	Domain	6	Package (Flight, Hotel, Car)	
	2	Skill-agent Functionalities	7	Combination of 3, 4, and 5	
	3	Skill-agent Flight			
	4	Skill-agent Hotel			
	5	Skill-agent Car			
Client	Travel Agency System				
	Owner				

Figure 5.20: Domain and scope ontology descriptor for TAS.

TAS ontology domain object diagram: According to DMMAS ontology analysis instructions, every skill-agent in TAS must be associated with one equivalent ontology domain object. For example the diagram in Figure 5.21 represents the flight skill-agent ontology object. The circles represent the concepts and the lines describe the attributes. It is important to clarify in this diagram the data-used and produced. Thus the data model focuses on defining the process input and output status from the data analysis perspective where the domain ontology object diagram views the system's skill-agent from a transition perspective in order to define the skill-agent object model and conceptual model to reflect the process in two views; the object

model: state the subject (domain) the process deal with. The other view is the transition model: state the change on the input by performing the process, here called the transition from the input to the output.

The flight reservation presented by Figure 5.21 focuses on the flight-skill agent ontology object. The ontology object structure consists of flight at the node and departure and arrival at the sub-node. The flight node represents the object model of the skill-agent flight reservation. This can be captured from the answer to the question "what are the attributes of the object flight". The answer is "flight that has departure and arrival attributes". Actually the basic idea is to structure the ontology model on the skill-agent ontology object model proposed in chapter 4 and the query target explained earlier. However, the right hand side of the diagram represents the transition depicted by the reservation on the top of the node and the customer, and passenger on the sub-node. The data element under each node represents the ontology primitives that could help in the implementation phase.

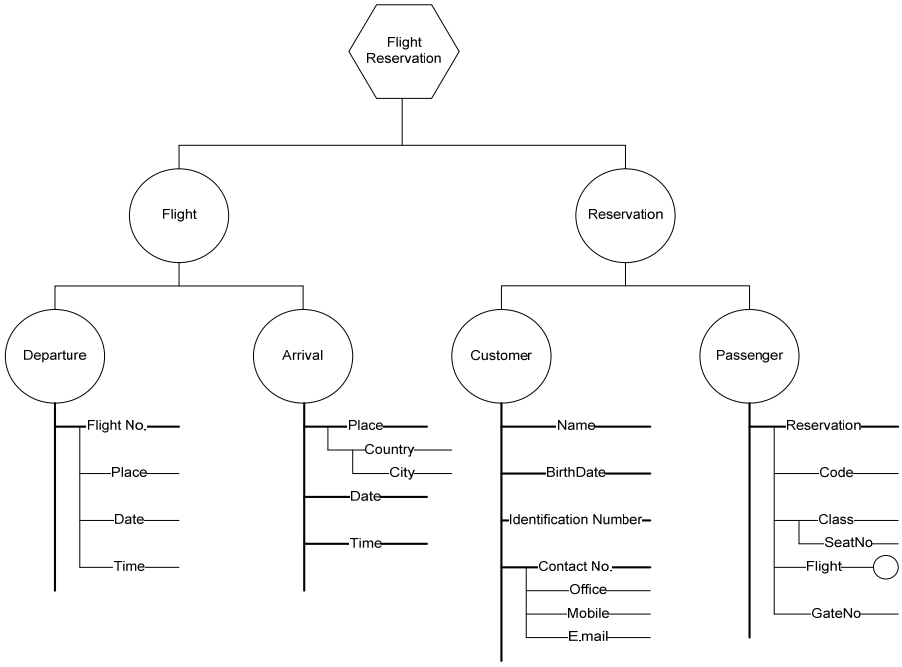

Figure 5.21: Flight Reservation skill-agent domain objects diagram.

The skill-agent "HotelBooking" in Figure 5.22 goes through the same analysis techniques explained earlier. The object model is represented by "Hotel" then defined

by accommodation and rooms as attributes to acknowledge the hotel concept in the "Hotel Booking" skill-agent. The transition model is represented by the "Booking" concept and defined by "Customer" concept which is with the transition becomes "Dweller" concept.

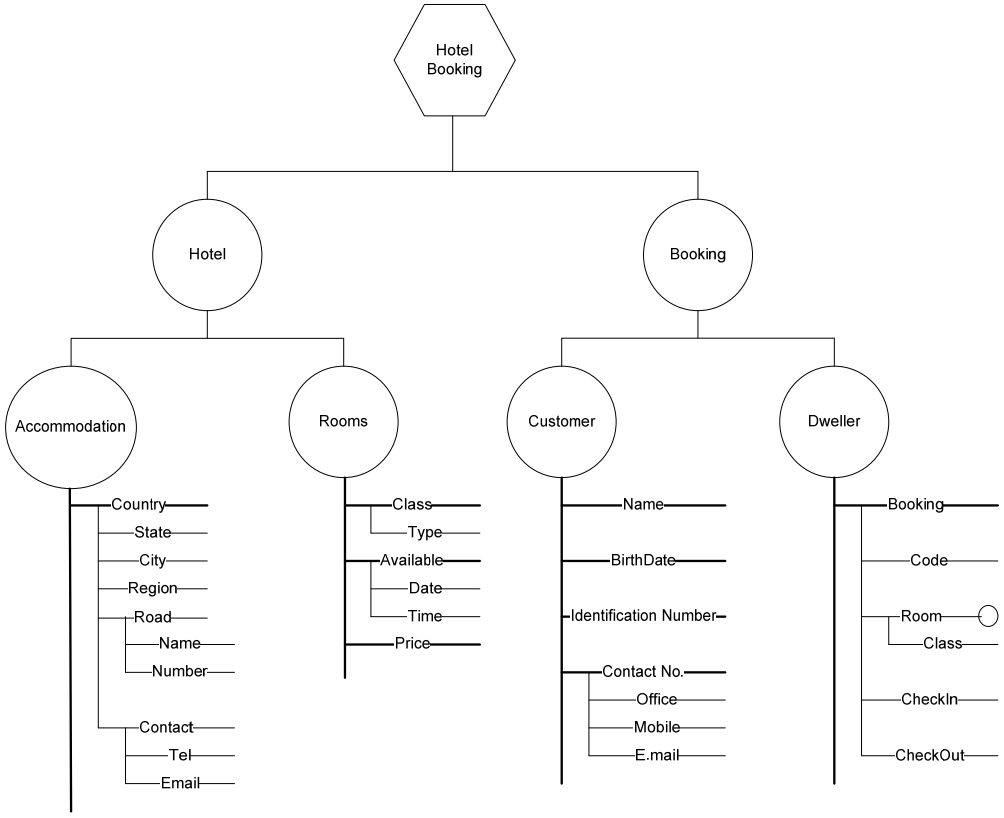

Figure 5.22: HotelBooking skill-agent domain object diagram.

The skill-agent "Car Rental" Figure 5.23 follows the same analysis techniques. The object model is represented by "Car" which is defined by vehicle and transport concepts to acknowledge the car concept for the "Car Rental" skill-agent ontology model. The transition model is represented by the "Hire" concept and defined by "Customer" concept which with transition, becomes "Rental" concept.

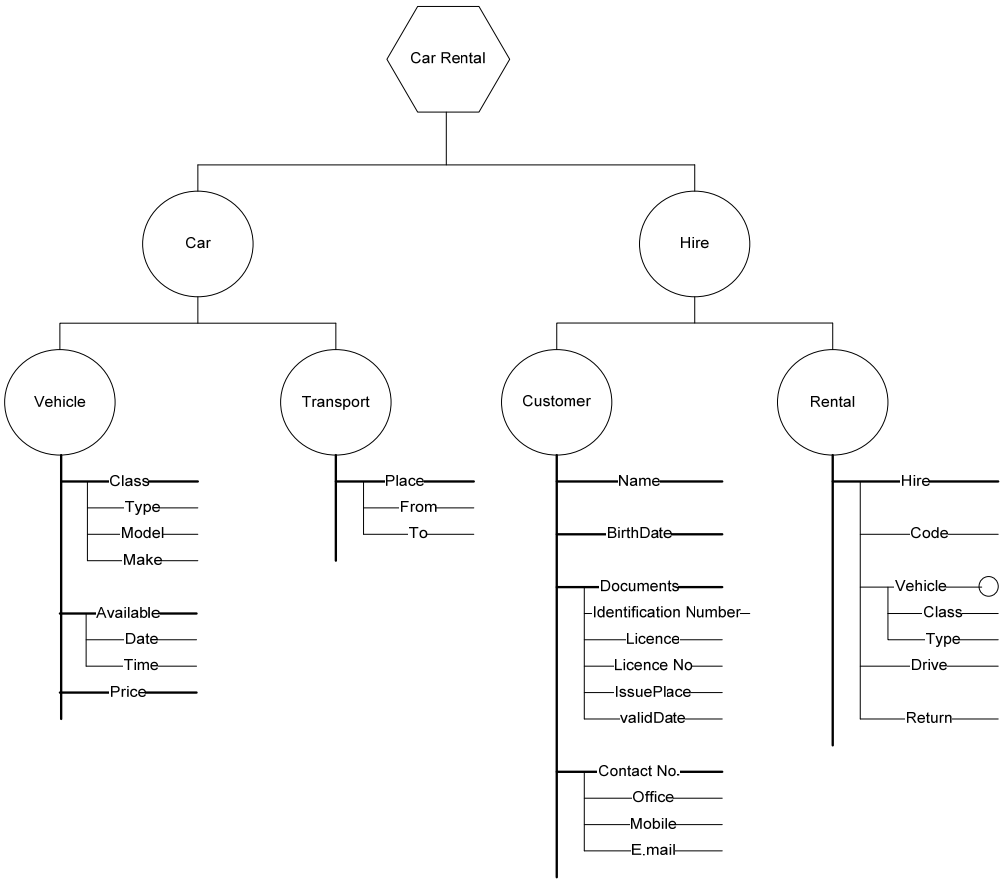

Figure 5.23: Car Rental skill-agents domain object diagram.

TAS skill-agents transition status: The last step in the analysis phase is to summarise the system ontology model in describable form. The skill-agent transition status is the main component that specifies each skill-agent in the system transition status. The skill-agent "Flight Reservation" transition status descriptor depicted by Figure 5.24 reflects the description of the concepts that construct the "Flight Reservation" skill-agent. The skill-agent transition status descriptor is in readable format design to help the developer to detect the ontology concepts and classes straightforward.

Skill-agent Transition Status: Flight Reservation				
Skill-agent	Name	FlightReservation	Note:	
	Code	FLIGHTRES		
Transition Status	No.	Process	Input	Output
	1	Reservation	Customer	Passenger
	2			
Object Definition	No.	Name	Attribute-1	Attribute-2
	1	Flight	Departure	Arrival
	2			

Figure 5.24: Flight Reservation skill-agent transition status.

Figure 5.25 exposes the skill-agent "Hotel Booking" transition status including the object and transition concepts that construct this skill-agent ontology model.

Skill-agent Transition Status: Hotel Booking				
Skill-agent	Name	HotelBooking	Note:	
	Code	HOTELBOKNG		
Transition Status	No.	Process	Input	Output
	1	Booking	Customer	Dweller
	2			
Object Definition	No.	Name	Attribute-1	Attribute-2
	1	Hotel	Accomodation	Rooms
	2			

Figure 5.25: Hotel Booking skill-agent transition status.

Figure 5.26 exposes the skill-agent "Car Booking" transition status including the object and transition concepts that construct this skill-agent ontology model. The concept "Customer" in the transition status under the input column has been repeated in all the three skill-agents transition status descriptors, according to the DMMAS rules it is not an error because both have different outputs status.

Skill-agent Transition Status: Car Rental				
Skill-agent	Name	CarRental	Note:	
	Code	CARRENTAL		
Transition Status	No.	Process	Input	Output
	1	Hire	Customer	Rental
	2			
Object Definition	No.	Name	Attribute-1	Attribute-2
	1	Car	Transport	Vehical
	2			

Figure 5.26: Car Rental skill-agent transition status.

At this stage the system analysis phase has been completed and furnished with all the artefacts, descriptors, definitions, and models that explain TAS requirements and specifications necessary to carry the system to the design phase. The DMMAS design phase is divided into two main steps; i.e. the architecture and the details where the former is preparing the artwork for the latter.

5.5 TAS Architecture Design

The DMMAS development phases complement each other, and develop the system as an iterative process. The architecture design will use the analysis phase artefacts and develop the system (TAS) architecture including the Ontologies model for the three skill-agents involved in TAS application. In this step, the professional-agent domain ontology and search mechanism ontology along with the skill-agent functionality ontology are set up.

The TAS three skill-agents have been specified and in this stage, these skill-agents need to develop further to extend their internal structures. As has been stated in Chapter 4, the skill-agent architecture is a high level abstraction and shows the main components of the skill-agent.

Flight reservation skill-agent state diagram: The flight reservation skill-agent intra processing design shown in Figure 5.27 depicts the high level main components that compose TAS flight reservation skill-agent. The purpose of this diagram is to specify

the flight reservation skill-agent intra processing in order to expand it in the next phase to become the skill-agent detailed design diagram. DMMAS view the skill-agent as a standalone service provider that has a transitional course of action consisting of input status, set of processes and output status. The effect of the intra processing transfers the input status to output status. Therefore the skill-agent functionality is the set of processes that transfer the input to output and this is used to define the TAS flight skill-agent functionality which DMMAS ontology schema intends to define.

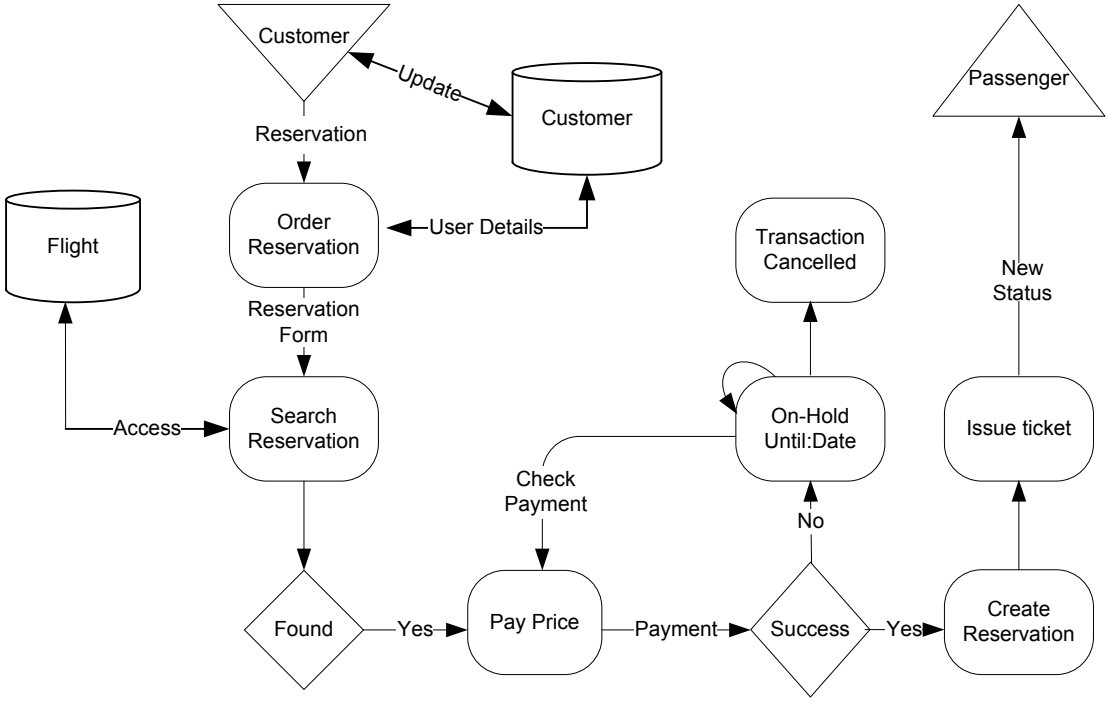

Figure 5.27: Flight Reservation skill-agent state diagram.

The state diagram illustrated in Figure 5.27 has been developed using UML activity diagram with additional triangles graphical notation to represent the input and the output status. The input status is defined by “customer”, and then when the flight reservation activities process the reservation request, the output status becomes “passenger”. Therefore the TAS flight reservation skill-agent functionality is defined by the transition ability to change the status from “customer” to “passenger” following these main processes.

Hotel Booking skill-agent state diagram: Figure 5.28 provides a diagrammatical state for the TAS hotel booking skill-agent. The diagram exposes the flow of the main process that changes the input status from “customer” to “dweller”.

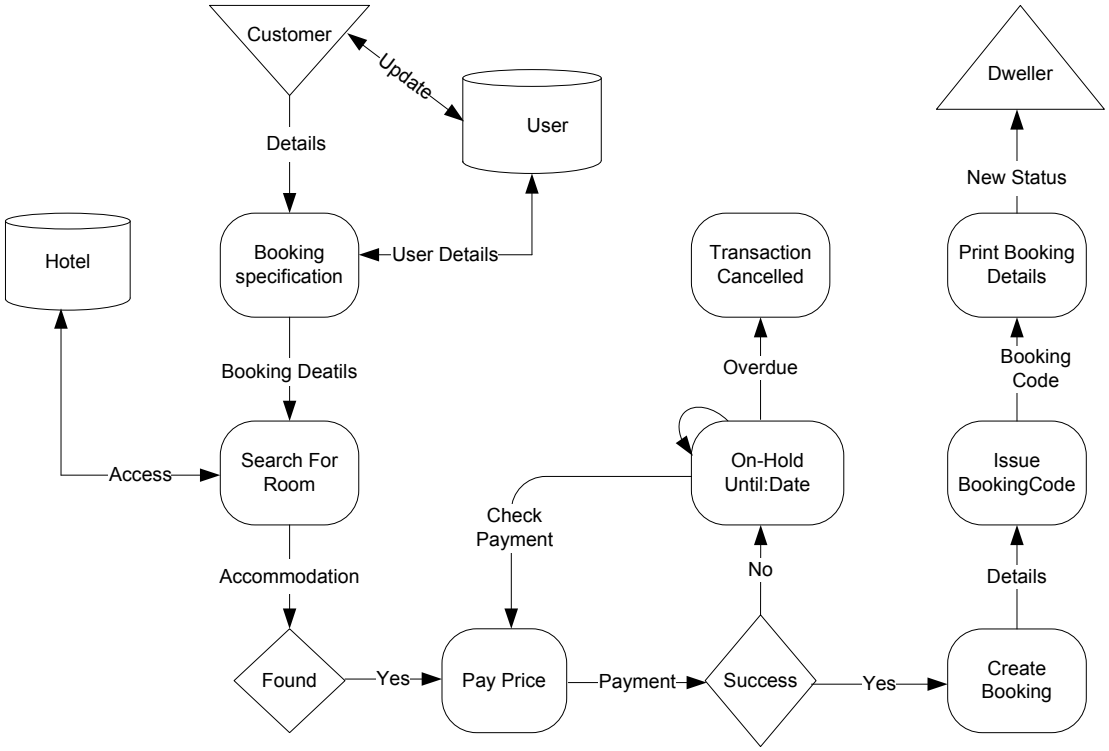

Figure 5.28: Hotel Booking skill-agent state diagram.

The hotel booking skill-agent state diagram shown in Figure 5.28 is triggered by the request of the input status “customer” then the booking specification will access the user database to update the search for room process. If the accommodation is found then the payment process is started followed by create booking process. In case the payment process fails then the booking transaction will go on hold according to the business rule. The booking code issue and the input status are changed to “dweller”. Therefore the hotel booking skill-agent functionality can be defined by the transition ability from “customer” to “dweller”.

Car Rental skill-agent state diagram: Figure 5.29 depicts the TAS car rental skill-agent state diagram. The process is initiated by the input status “customer” then this input is transferred to output status “rental” after passing through a set of processes. The scenario of car rental skill-agent is similar in concept to the flight reservation and hotel booking. The difference is that the car rental skill-agent functionality is defined by the ability of transfer the input from “customer” to “rental”.

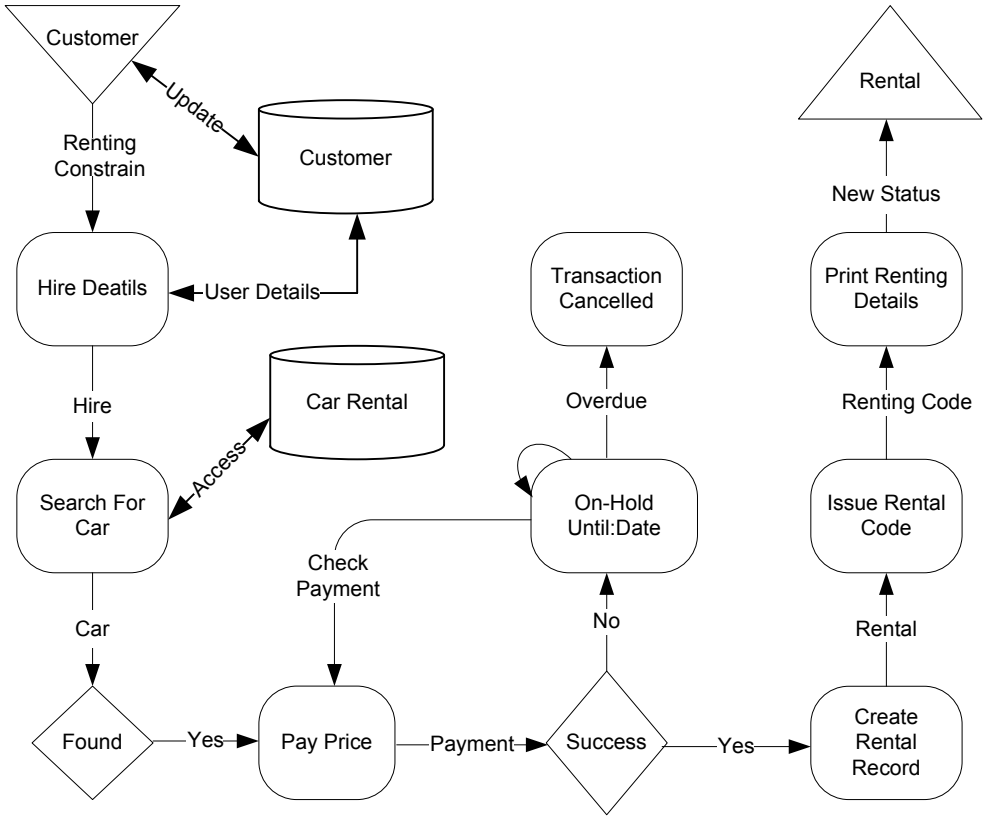

Figure 5.29: Car Rental skill-agent state diagram.

TAS professional agent processes diagram: Figure 5.30 shows TAS professional agent processes in the case of flight reservation skill-agent as an example. TAS professional-agent is responsible to execute all the system goals within its domain by adopting the skill-agents provided in the goal execution plan. DMMAS set the professional agent as a node of a particular domain for example in TAS case study there are three skill-agents so the domain is represented by TAS professional-agent.

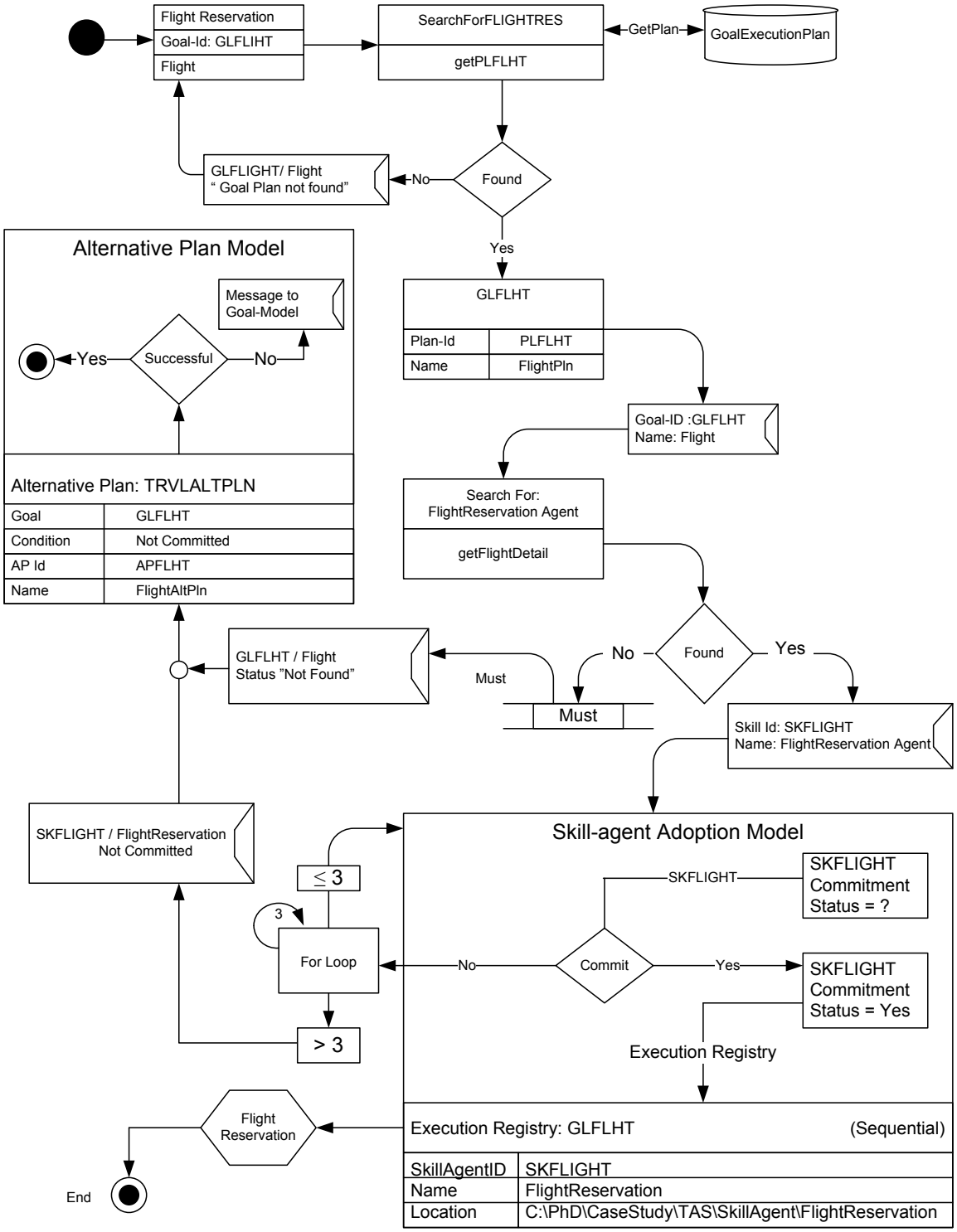

Figure 5.30: TAS professional-agent process diagram for flight reservation example.

The TAS professional-agent processes for the flight reservation goal illustrated in Figure 5.30 consists of three main components. First is the goal execution plan set previously (see page 208) and converted to a database table, the second component is the skill-agent adoption model that builds the execution registry and commits the

skill-agent to the team. The third component is the alternative plan that works as an option in case the skill-agent is not reachable. The skill-agent adoption model shows that there is a send and receive request for a maximum of three times to adopt the flight skill-agent. In case the skill-agent accepts the deal, the flight skill-agent commitment status will change to "Yes" and the skill-agent Id, name, and location are loaded in the execution registry. To simplify the professional-agent diagram the flight reservation goal is used as the template for the hotel booking and car rental which follow the same processing. The alternative plan is sending a message in case the flight reservation skill-agent does not respond after three requests. This assumes TAS is open distributed multi-agent systems and removing or adding skill-agent is predicted.

TAS skill-agents functionality structure: At this stage it is important to establish the grounding to the ontology schema. This can be achieved gradually, starting by determining the ontology context. The ontology context will help to avoid building costly ontology, because developing ontology is an open ended process where the ontology classes' relationship can run into a long continuous chain that could reach beyond the development purpose. Subsequently it costs time and effort. To set up the scene around the required ontology for TAS the functionality structure is defined at this stage.

The skill-agent functionality architecture is defined in three components, i.e. the identification, information, and functionalities components. The identification and the information for TAS skill-agents are depicted in Figure 5.31. These components are designed to help the ontology developer in case there is a predefined ontology or there are any reuse ontology segments.

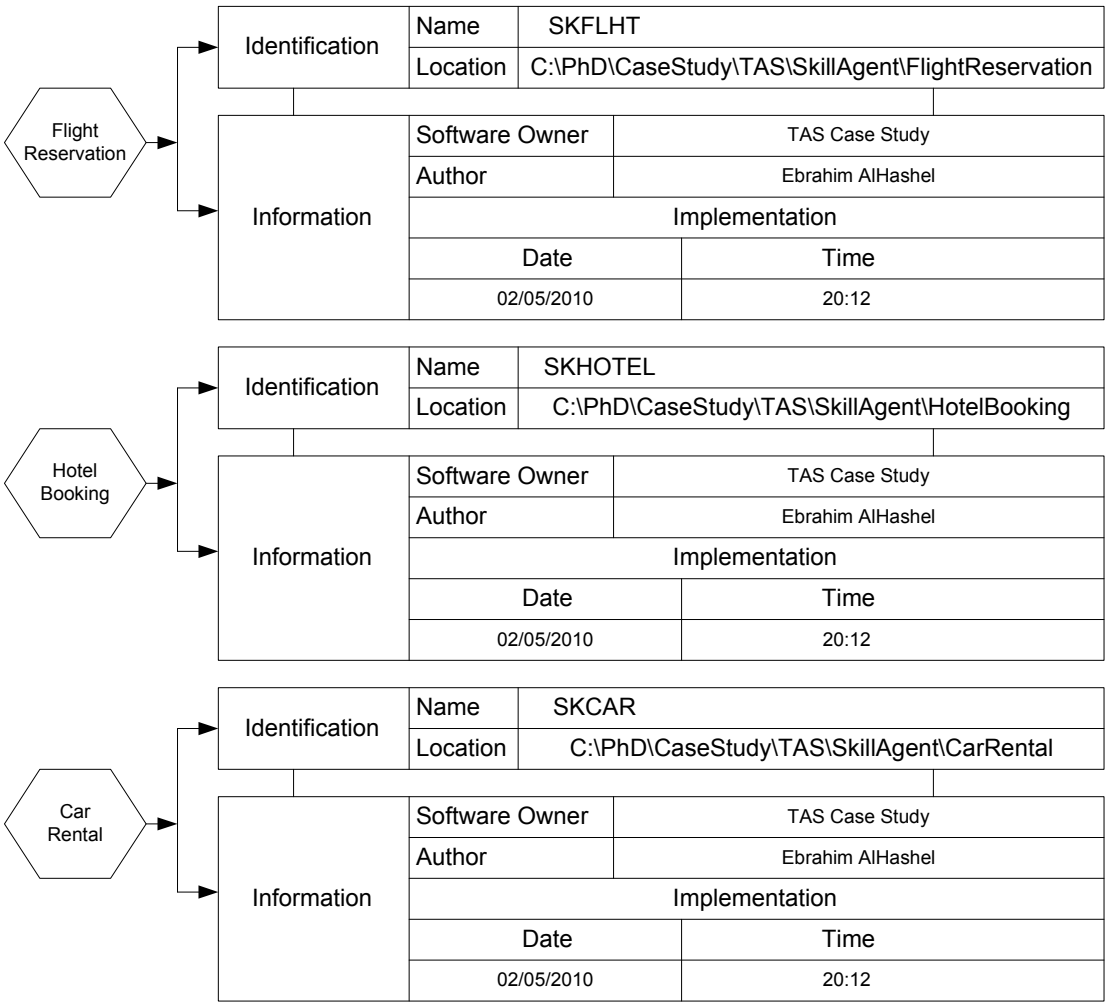

Figure 5.31: TAS skill-agents functionality identification and information components.

Functionality component: The functionality component is structured in two main nodes, descriptive and behaviour. Both these nodes are defined in one XML-based schema regardless of functional representation. The descriptive node details the functionality from the role perspective (what) and on the other hand the behaviour node defines the skill-agent functionality from the processing perspective (how). Generally, the functionality component consists of a list of optional attributes enabling the developer to decide on the number of attributes usage to define the functionality. In addition to the functionality component DMMAS identifies the query for the ontology model to answer. This helps to put a boundary around the required ontology model and also reconciles the ontology depth with the requirements.

According to DMMAS instructions, before setting up the ontology queries, the functionality criteria proposed in DMMAS architecture design phase in Chapter 4 must be defined a priori. This simplifies the query arguments and statements definition process.

Flight skill-agent functionality component: Figure 5.32 represents the flight reservation skill-agent functionality description component. The descriptive branch is defined by a list of single words that describes the functionality in the hierarchy structure. The descriptive schema structured in a conical hierarchy that started with a wider related process name then narrows down the description to the unit level. In this flight reservation skill-agent the process is based on "Reservation" and the aim or the lowest unit in the reservation is the seat that the skill-agent aims to reserve. Based on this concept the descriptive branch is developed. The Behaviour branch represents the flight reservation activity or the process. The terms that explain the rows are identified using a question refinement process. For example to find the action, one can ask what the skill-agent tries to do (aim); the answer is to reserve a seat. The action "ReserveSeat" represents the flight reservation skill-agent functionality transition state. Generally, the terms used in both branches are finally merged in one ontology schema that can reason and infer the flight reservation skill-agent main functionality.

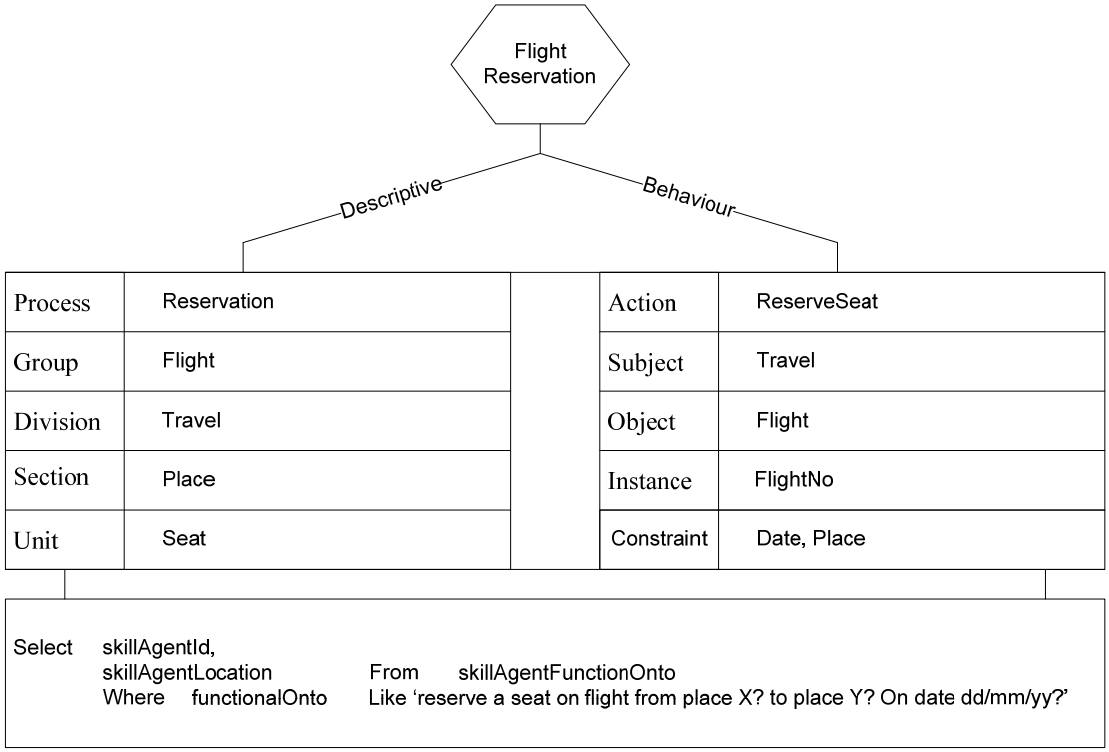

Figure 5.32: Flight Reservation skill-agent functionality component.

Hotel skill-agent functionality component: Figure 5.33 illustrates the Hotel booking skill-agent functionality description component. The concept used to describe the hotel booking skill-agent is the same as the techniques presented in Chapter 4. The constraint of the behaviour branch contain three attributes; Place, From-date, and Number-of-nights. These attributes are directly correlated to the hotel booking process and form key elements or critical elements in the booking process. There are other important elements, for example, payment data, but the aim is to focus on the booking functionality, not on the entire booking process.

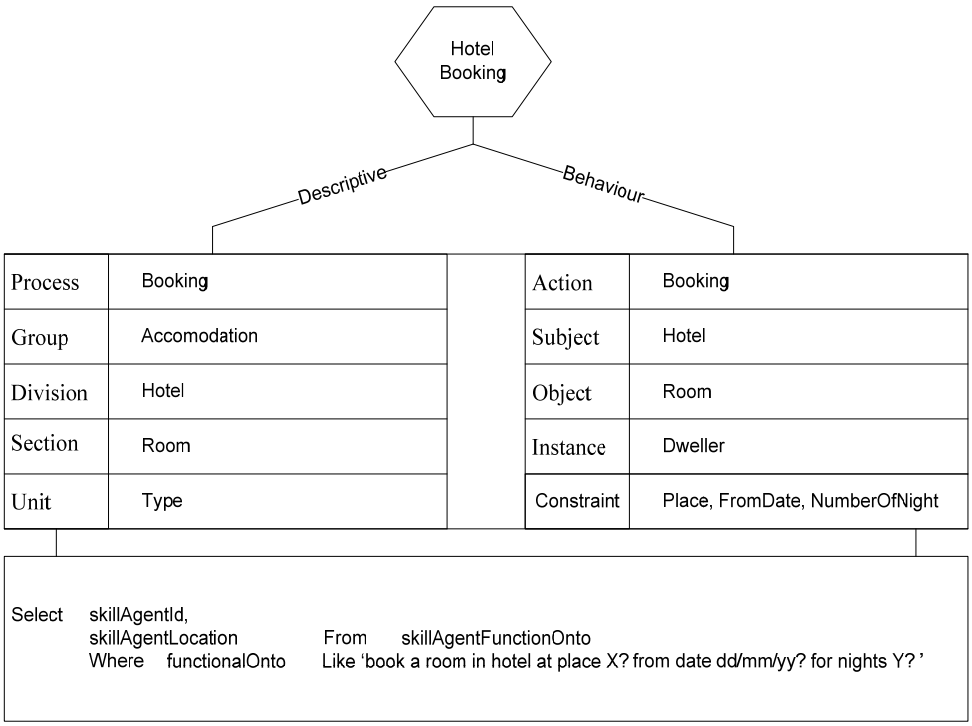

Figure 5.33: Hotel Booking skill-agent functionality description component.

Car skill-agent functionality component: The car rental skill-agent set up following the same previous techniques used in flight reservation, and hotel booking. The select query statement demonstrates the possible query that the ontology model must answer. The question mark "?" represents the variables of the constraint values.

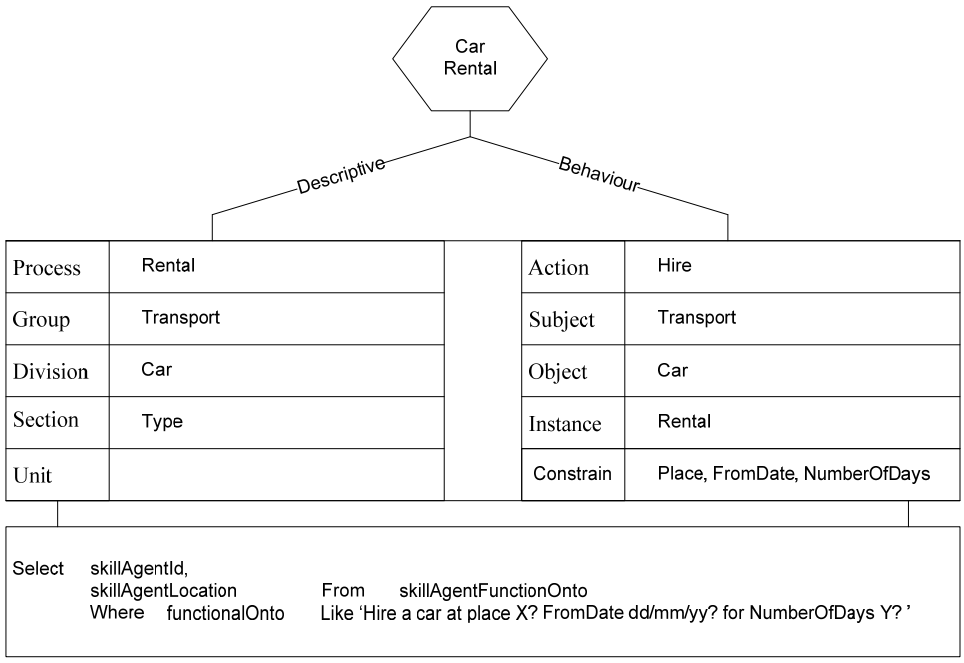

Figure 5.34: Car Rental skill-agent functionality component.

The functionality model is an ontology based schema developed to answer queries related to the skill-agent purpose or services that it can provide (through its functionality). The functionality model is part of the search model. It interfaces with the system skill-agent in the form of one unit of process or service provider which has input, some processing, and output. The functionality model is built using the class notation proposed in Chapter 4. Furthermore, the implementation language abstraction is XML-based ontology, and also considers the ontology modelling to attain implementable model. To demonstrate the ability of DMMAS development concepts all TAS skill-agents (flight, hotel, and car) in the ontology structure will be developed at this stage with consideration to the seven steps mentioned in Chapter 4 page 147 under System Architecture Design.

Flight reservation skill-agent functionality ontology model: Figure 5.35 presents the flight reservation skill-agent functionality ontology in hierarchal classes and subclasses relationships. This model structured base on the flight reservation skill-agent domain object diagram and skill-agent transition status descriptor are explained on page 188. The flight reservation skill-agent functionality ontology model shows the classes and the main instance that transfers the skill-agent input status from "customer" to "passenger". The flight reservation ontology model defined in "travel" domain where the car and the hotel skill-agent also share the same domain, so the final model represents the entire TAS classes' relationship as one ontology schema.

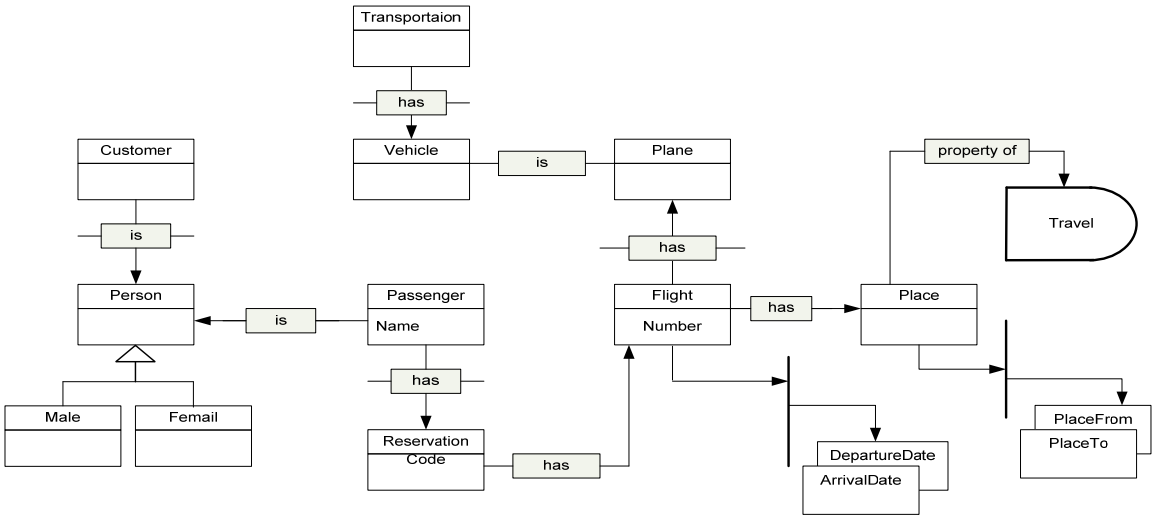

Figure 5.35: Flight Reservation skill-agent functionality ontology classes diagram.

The flight reservation ontology model shown in Figure 5.35 depicts the model that defines two concepts, i.e. the customer and passenger in respect of flight reservation in the travel domain. The passenger class is related to the reservation class where the reservation has a flight that has two instances, departure date and arrival date. The flight class has two classes' relations, the place and the plane. This relation defined as flight has a plane which is type of transport and place class. The place class in this purpose is presented by two instances "place-from" and "place-to" which reflect the meaning of place for the flight reservation functionality. The customer concept or class in relation to the passenger class is type; therefore both classes are correlated.

Hotel reservation skill-agent functionality ontology model: Figure 5.36 depicts the hotel booking skill-agent functionality ontology classes relationships. The ontology domain and the concepts are similar to the flight reservations but the classes' structure set up to reflect hotel as type of accommodation in the travel domain.

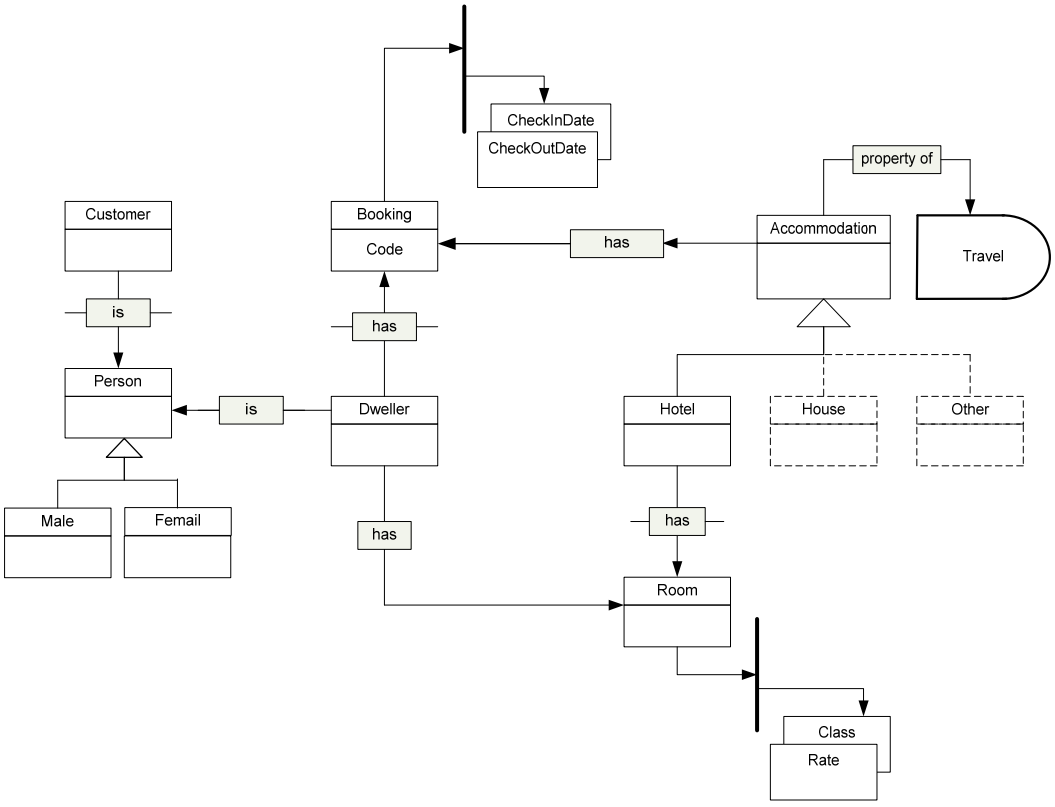

Figure 5.36: Hotel Booking skill-agent functionality ontology classes diagram.

The hotel booking defined by booking class generates two instances, i.e. the check-in-date and check-out-date and has attribute name "code". The accommodation class defined as super class for hotel which has room as subclass. This hierarchal

relationship defines room as subclass (belong to) hotel and hotel is type of accommodation and room has class and rate instances. The classes denoted by the dotted lines to "house", and "others" are included to improve the schema reading, however these classes are excluded from ontology implementation.

Car rental skill-agent functionality model: Figure 5.37 presents the car rental skill-agent functionality ontology classes' relationships. The ontology domain and the concepts are similar to the flight reservations and hotel booking but the classes' are structured to reflect car rental as ontology concepts in travel domain.

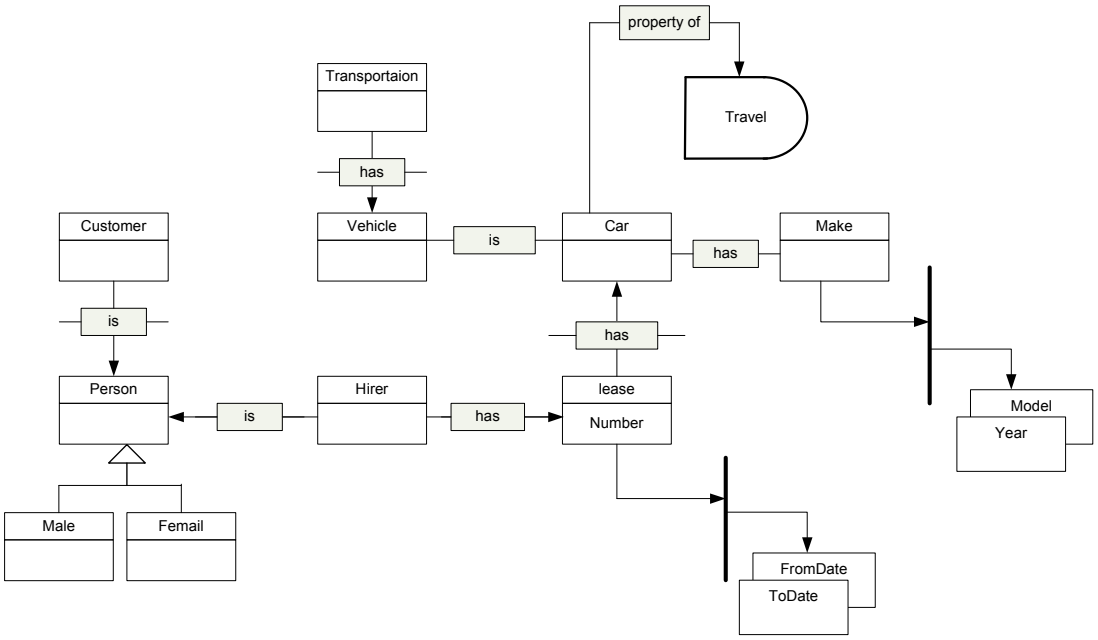

Figure 5.37: Car Rental skill-agent functionality structure.

The instances of "make class" and "lease class" are included in the car rental ontology because these are important constraints for the "car rental" skill-agent. Another advantage of these instances is that they provide additional options for the ontology schema reuse. The hirer class is intended to represent the output status and the customer class represents the input status. Both of these classes are related by "is" relation. On the other hand the customer class is defined as super class for person but the hirer class is defined in relation to the lease class which is part of the main

ontology classes. The lease class has an attribute “number” which provides information about lease class.

Goal execution model: The goal execution diagram is are unnecessary component for the TAS case study architecture design because the system goals are dynamic and the goal execution plan descriptor contains adequate information about the goals plans. Generally, at this stage all the TAS architecture components have been specified along with the system descriptors and the ontology building blocks. The architecture design artefacts are ready for further development to construct the TAS details design that will be carried out in the next section.

5.6 TAS Detailed Design

The detailed design phase uses the artefacts provided by the previous phases then develops the detailed specification for each component in the required system. This detailed design will describe the system interactions, interfaces, system functions, and ontology details design model to serve the system implementation phase. The first step in DMMAS detailed design is to set the skill-agent internal processing details. To perform this task, the skill-agent architecture presented in the architecture design phase will be expended further to expose all the components details.

Flight reservation skill-agent internal structure: All skill-agents have the same structure and concept but realise different functionality. Figure 5.38 depicts the flight reservation skill-agent classes and databases and the internal system messages. DMMAS deal with the skill-agent as high abstractions to give more space for decentralisation design and development. In addition it allows multi different structure agents to join the systems provided that it has the main components (request/reject, alternative plan, and execution plan) and Ontologies their functionality as per DMMAS standards. However, the external entities shown in Figure 5.38 indicate that this component is not designed in this step but it interacts with this skill-agent.

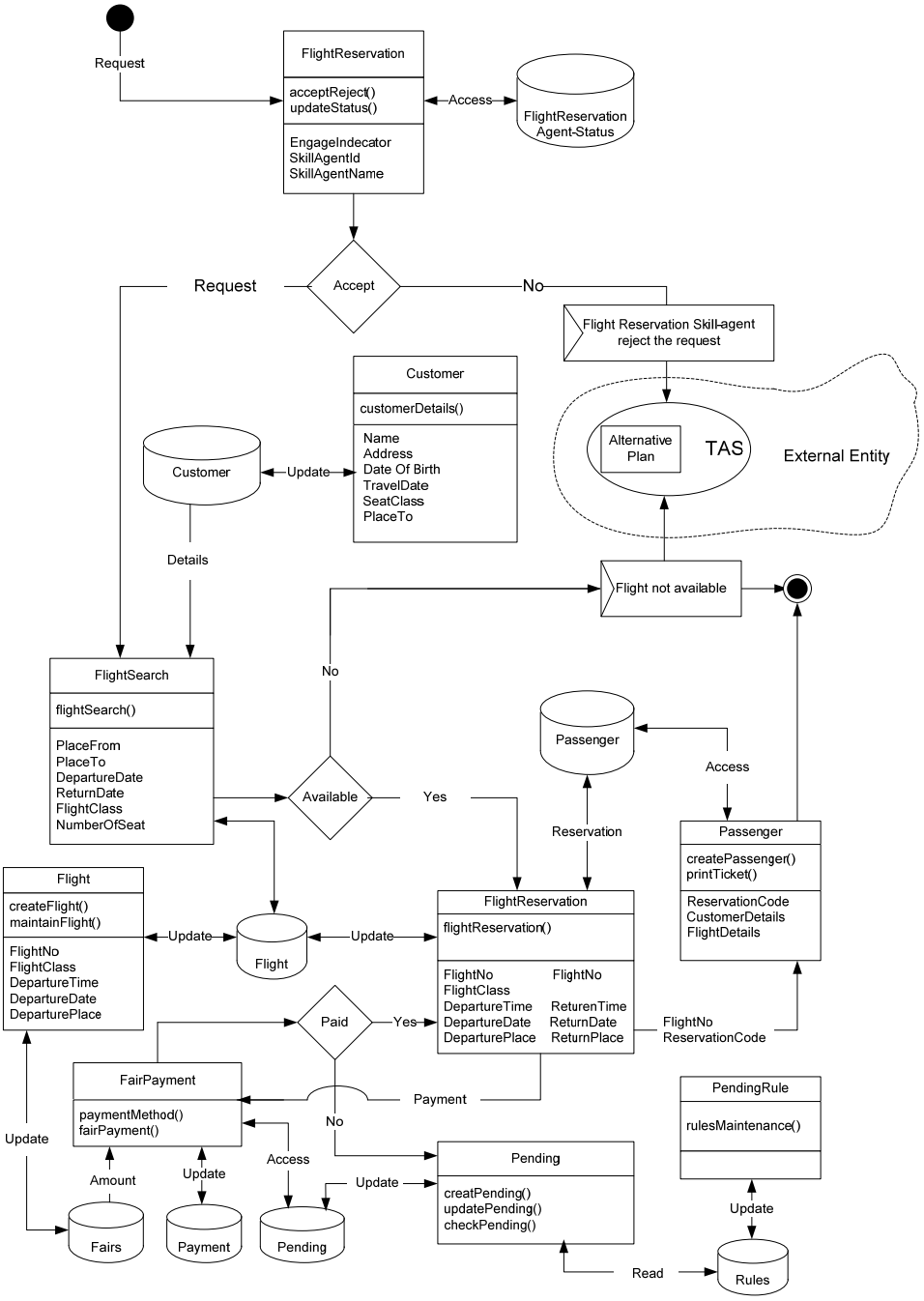

Figure 5.38: Flight reservation skill-agent detailed design.

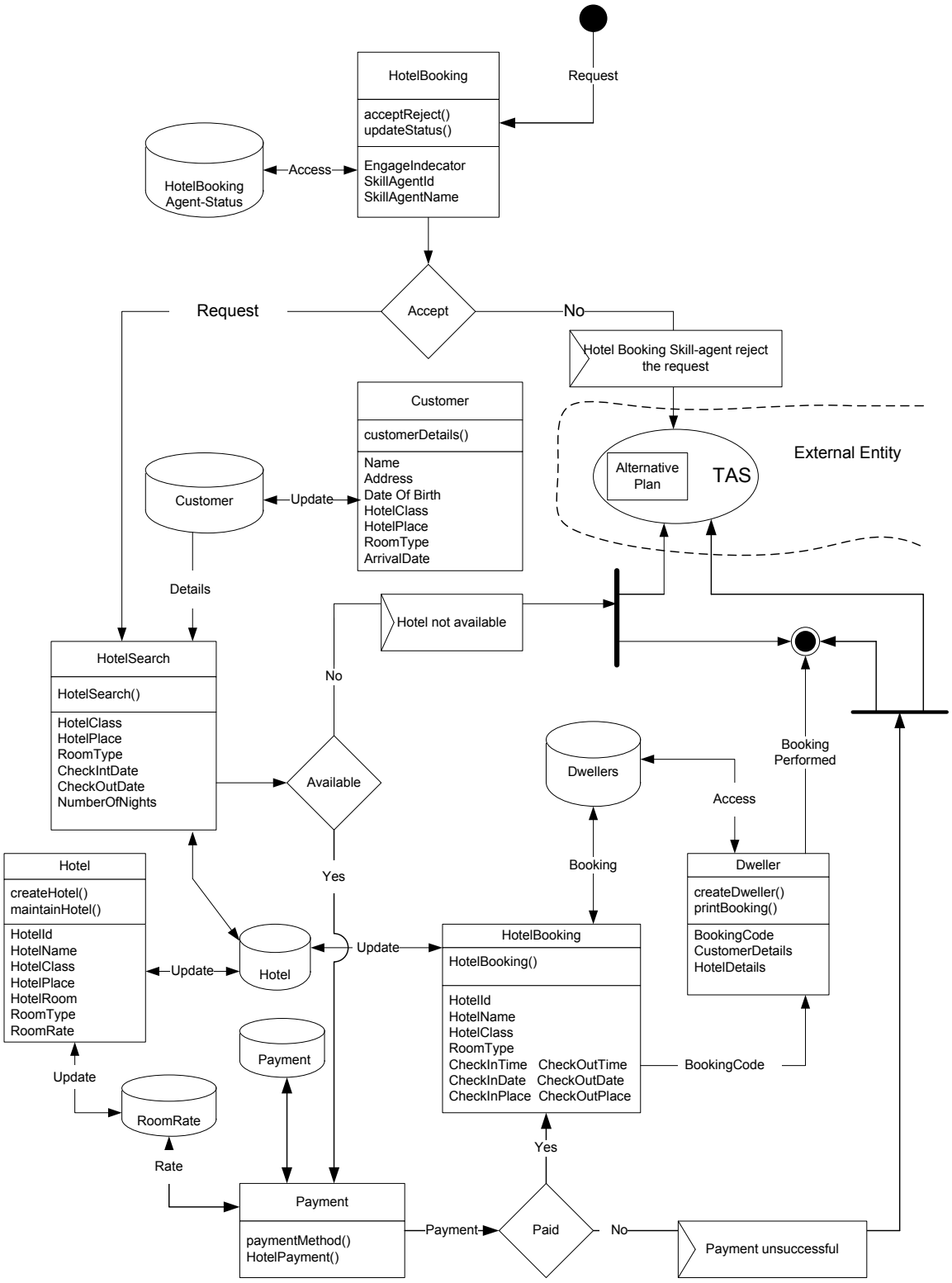

Figure 5.39: Hotel booking skill-agent detailed design.

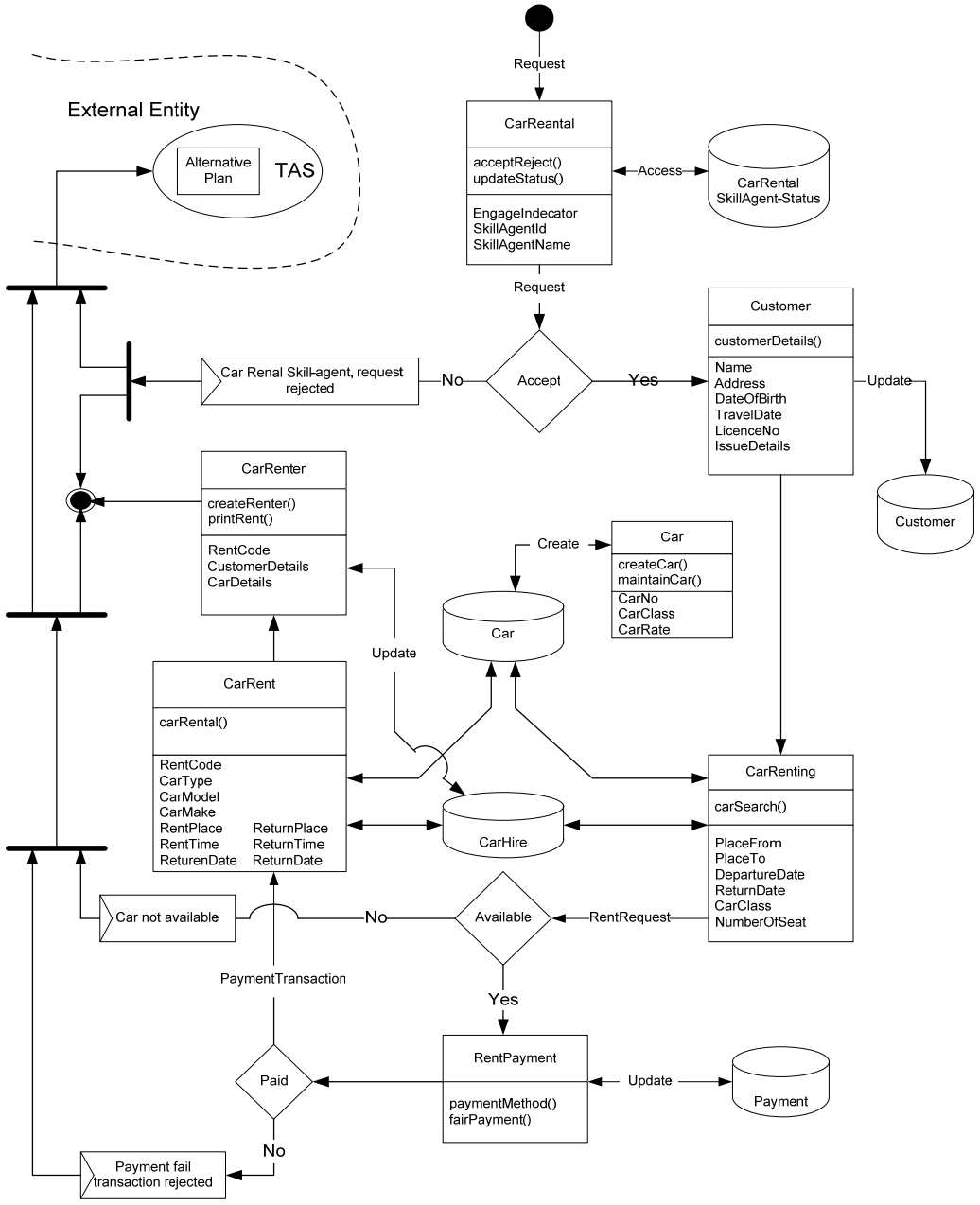

Figure 5.40: Car rental skill-agent detailed design.

TAS professional-agent detailed design: There is only one professional-agent in the TAS application. Therefore the architecture of TAS professional-agent does not include a search model for professional-agent. Figure 5.41 illustrates the TAS professional-agent detailed design. The skill-agents are shown by the hexagon notation abstraction and interact with the professional-agent through request and reject components. This deals with the skill-agent as an autonomous entity and provides the system with maximum flexibility. Figure 5.41 presents TAS

professional-agent main components and shows that there is a loop for the request while building up the required skill-agent team based on the skill-agent identification, name and location (address) because the system is built on the assumption that it interacts with skill-agents in the distribution environment.

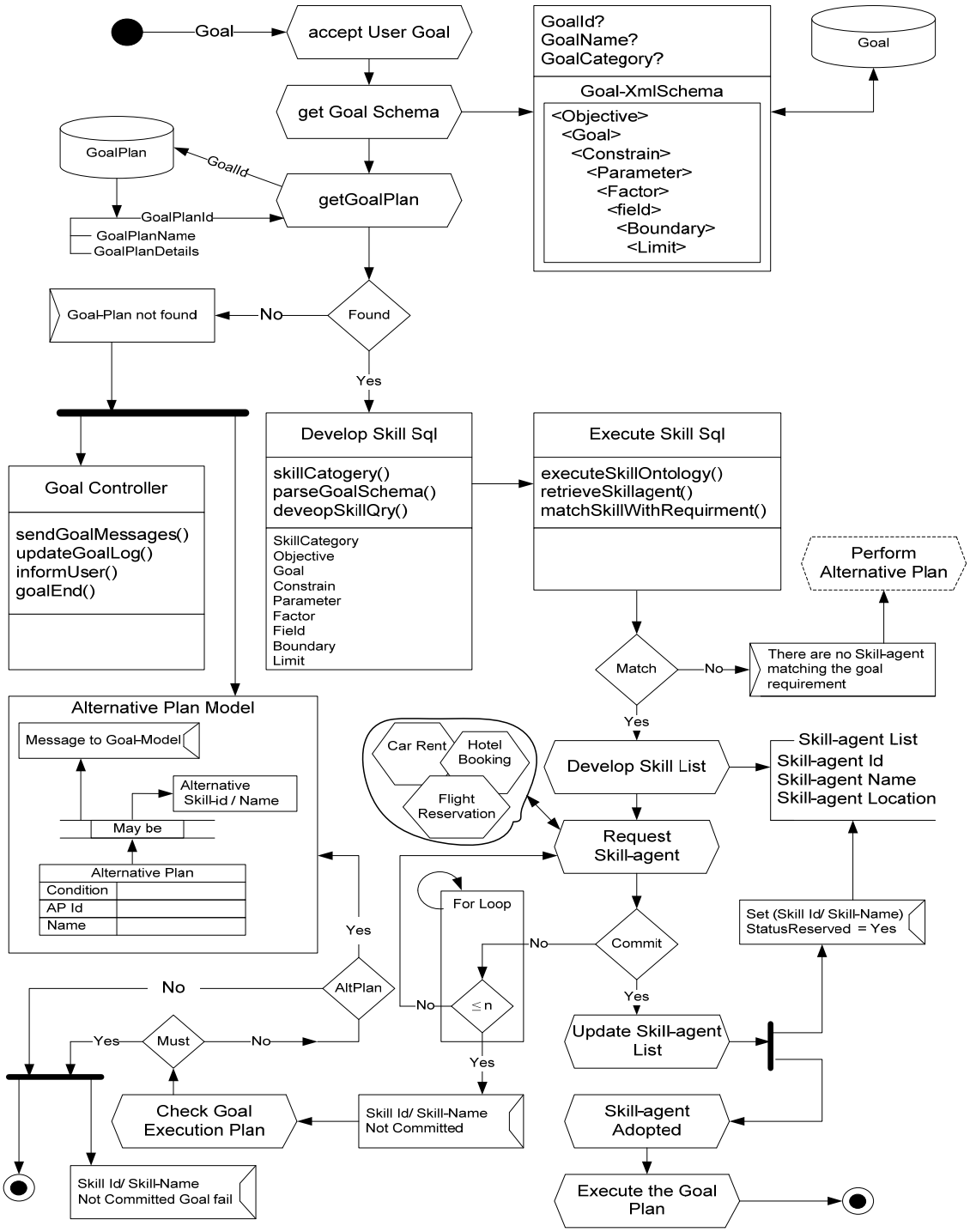

Figure 5.41: Professional agent detailed design diagram.

TAS goal execution plan: To improve the development guidelines the execution plan schema presented in the design architecture phase needs to be supported by a graphical diagram for clarification on the processing orders. Since the three skill-agents will follow the same processing flow the diagram in Figure 5.42 depicts TAS goals execution plan as a general diagram for all TAS goals. The goal execution plan diagram (Figure 5.42), extends the "Execute the goal plan" task defined in the professional-agent diagram (Figure 5.41).

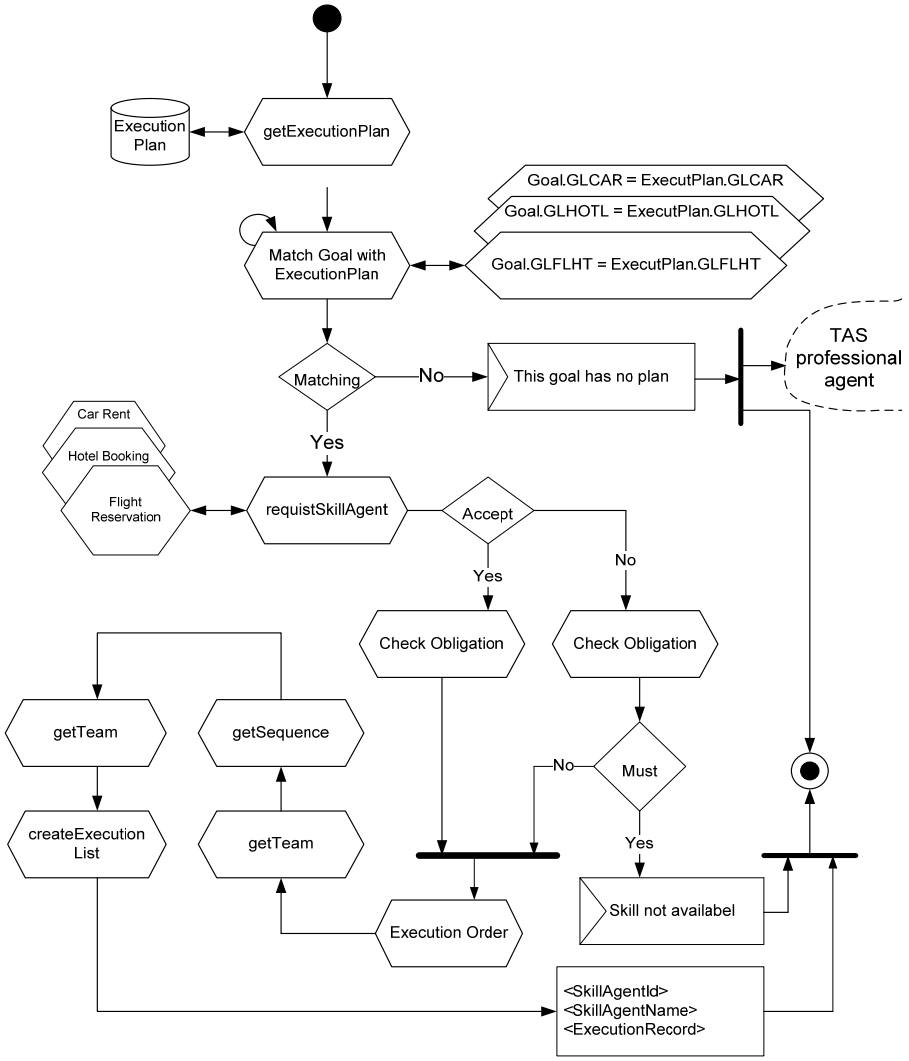

Figure 5.42: TAS Goals execution plan detailed design.

The dash line around the TAS professional-agent indicates that its external entity is outside this execution plan diagram context. The "create execution registry" task will build the execution attributes that form the entire plan where the execution records

hold all the attributes related to the plan and refer to the execution plan descriptor architecture design phase.

TAS flight goal XML-Schema structure diagram: The main purpose for the goal XML schema diagram is to agree on the goal elements design to simplify the implementation process. In Chapter 4 DMMAS proposed an adjustable goal XML-schema that can be modified to mirror the goal and its constraints. The diagram represented in Figure 5.43 demonstrates the basic XML elements for TAS flight goal. The elements represent the flight goal main constraints. There are other constraints can be also introduced for example the departure time or the arrival time, but according to the design assumption these constraint are optional. It is important to remember that this schema is to represent the goal and it is different from the previous schema which represents the skill functionality. Appendix A defines the TAS goals schema in XML document.

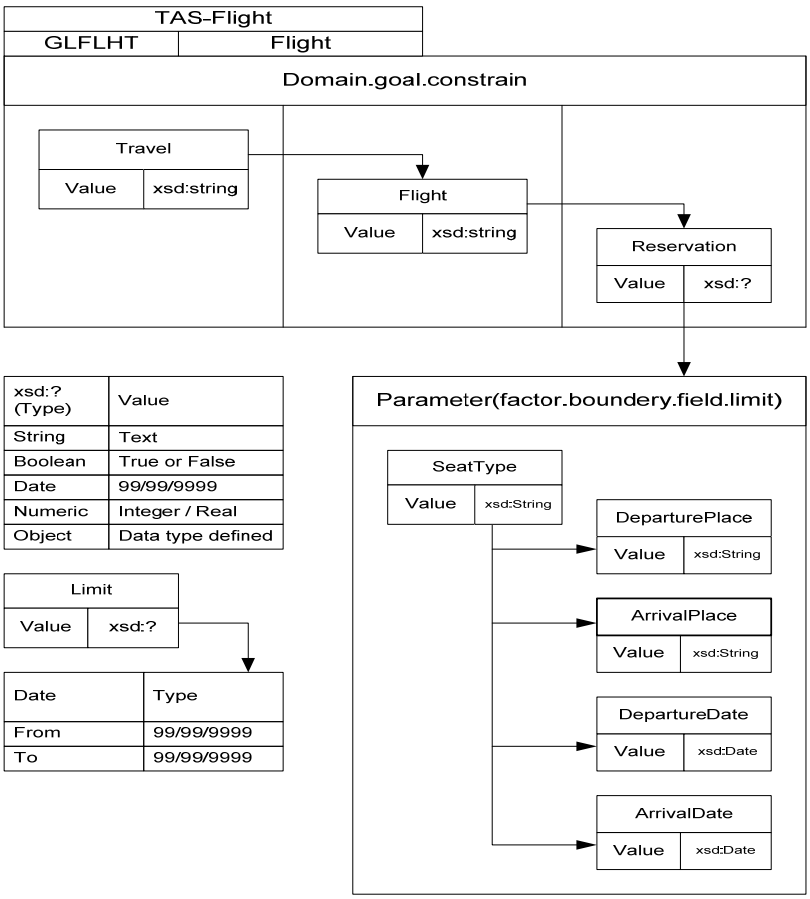

Figure 5.43: XML Schema for flight goal.

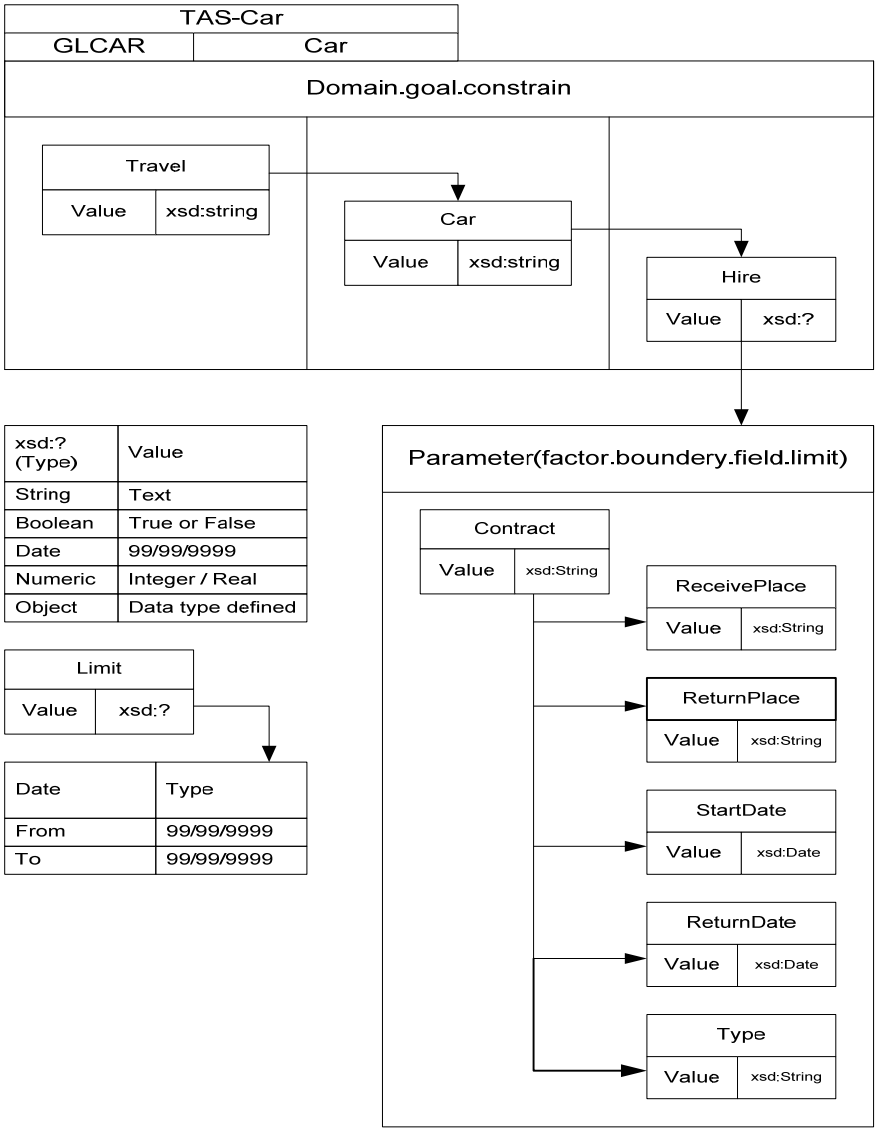

Figure 5.44: XML Schema for Hotel goal.

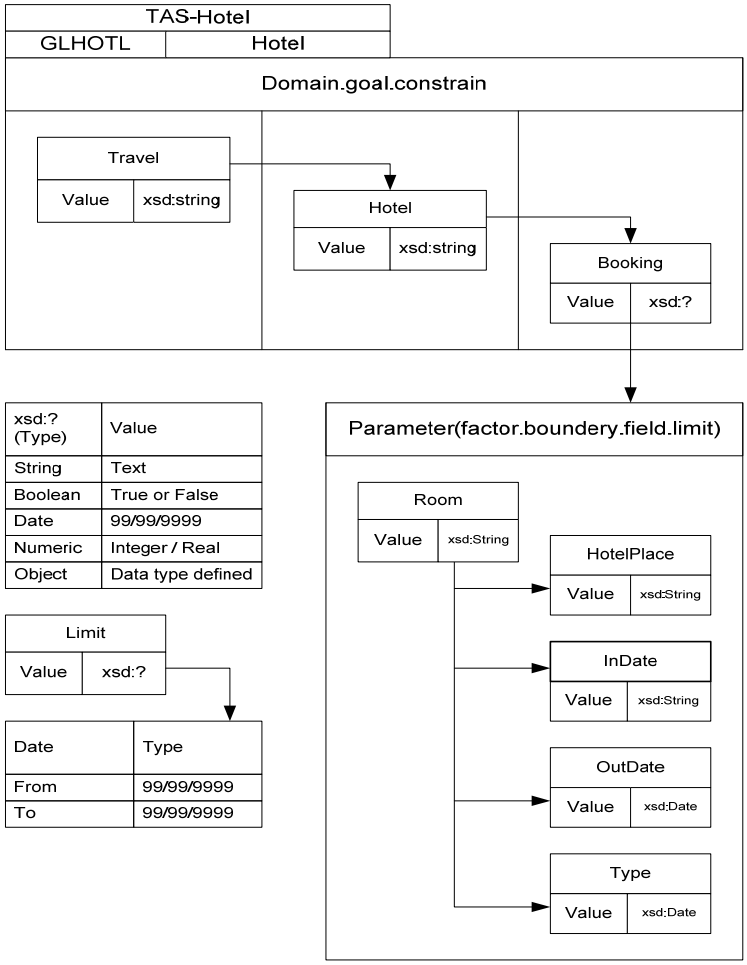

Figure 5.45: XML schema for Car goal.

The SQL statements defined below are structured to retrieve the TAS goals execution information for each skill-agent in the system including: goal identification, goal name, the system that the goal belongs to, the execution team, the sequence identification and the goal priority. These data will be used by the professional-agent to execute the goal selected by the user. Therefore at every goal in the system there is one SQL statement to retrieves its execution data. For example, the TAS has seven goals, below are seven SQL statements in equivalent to each goal:

```
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
```

```
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      =  Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  =  Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  =  Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      =  Goal.GoalId

            WHERE  SystemGoals.GoalID     LIKE  "GLFLHT";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      =  Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  =  Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  =  Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      =  Goal.GoalId

            WHERE  SystemGoals.GoalID     LIKE  "GLHOTL";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      =  Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  =  Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  =  Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      =  Goal.GoalId

            WHERE  SystemGoals.GoalID     LIKE  "GLCAR";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
```

```
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      = Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  = Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  = Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      = Goal.GoalId

            WHERE SystemGoals.GoalID      LIKE "GLFLHT"    OR
                  SystemGoals.GoalID      LIKE "GLHOTL";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      = Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  = Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  = Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      = Goal.GoalId

            WHERE SystemGoals.GoalID      LIKE "GLFLHT"    OR
                  SystemGoals.GoalID      LIKE "GLCAR";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId      = Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId  = Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId  = Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID      = Goal.GoalId
```

```
        WHERE SystemGoals.GoalID      LIKE " GLHOTL"    OR
              SystemGoals.GoalID      LIKE "GLCAR";
-------------------------------------------------------------------------------------------------------
SELECT      SystemGoals.GoalID,
            Goal.GoalName,
            SystemGoals.TeamId,
            Team.*,
            SystemGoals.SequenceId,
            Sequence.*,
            SystemGoals.PriorityId,
            Priority.*

            FROM (((SystemGoals
                  INNER JOIN  Team      ON  SystemGoals.TeamId     = Team.TeamId)
                  INNER JOIN  Priority  ON  SystemGoals.PriorityId = Priority.PriorityId)
                  INNER JOIN  Sequence  ON  SystemGoals.SequenceId =  Sequence.SequenceId)
                  INNER JOIN  Goal      ON  SystemGoals.GoalID     = Goal.GoalId

            WHERE SystemGoals.GoalID      LIKE "GLFLHT"    OR
                  SystemGoals.GoalID      LIKE "GLHOTL"    OR
                  SystemGoals.GoalID      LIKE "GLCAR";
```

Figure 5.46 depicts the database relationship between the skill-agent identifications data and the skill-agent transition states. This database relationship represents the functionality of the skill-agent in terms of input and output status, therefore it can be used as part of the skill-agent identification but from functionality definition.

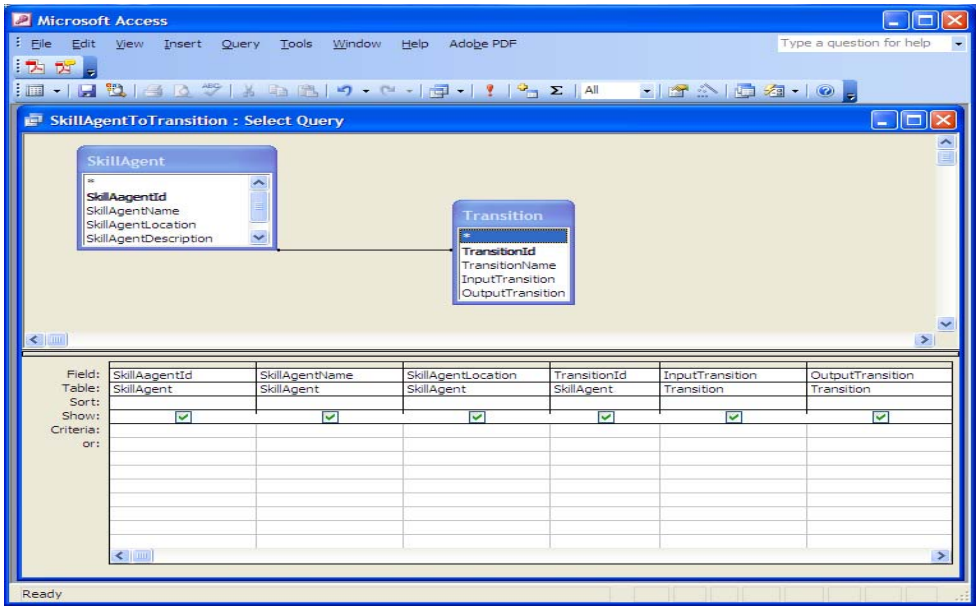

Figure 5.46: Skill-agent and transition status database relationship.

Figure 5.47 illustrate the database relationship between TAS skill-agents and the execution plan for each goal in the system including the alternative plans and the transition status. The database integrate the TAS entire components and their professional-agent to provide a main plan for the goal execution based on the selected skill-agents that identified through the ontology search based.

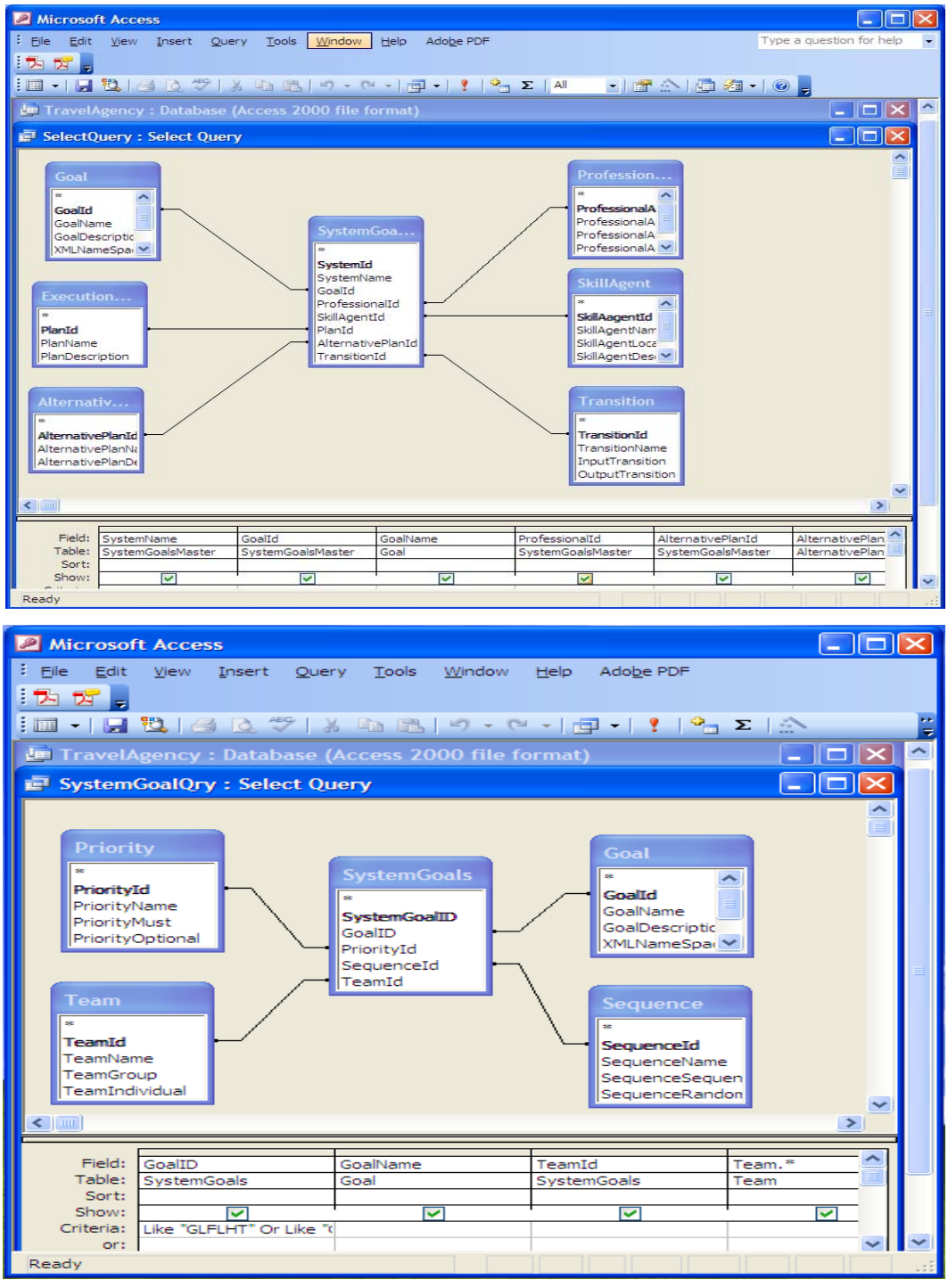

Figure 5.47: TAS goals, priority, sequence, and team database.

System interaction diagram: The last step in the detailed design is to set up the system interaction diagrams. The interaction diagram is to reveal the multi-agent systems main components interaction messages and related conditions. DMMAS interaction diagrams follow AUML standard with minor additional modifications. The AUML have a set of tailor made notations to the software agent concept (appendix D explains AUML notation and syntax). In contrast to the object-oriented system, the multi-agent systems and from DMMAS point of view the AUML syntaxes, is not for the intra component interaction but instead, the UML can be used for this level of design. However, the interaction diagram presented in Figure 5.46 plots the messages between the user, the professional-agent TAS, the flight skill-agent flight reservation, and the execution plan component. For the actors representation DMMAS use the same proposed notation to maintain the consistency in the modelling. For example, TAS professional-agent, and the flight reservation skill-agent are both presented using their notations. Another changes applied by DMMAS interactive diagram is not to use the rectangular boxes, but instead to use the operation fork lines only.

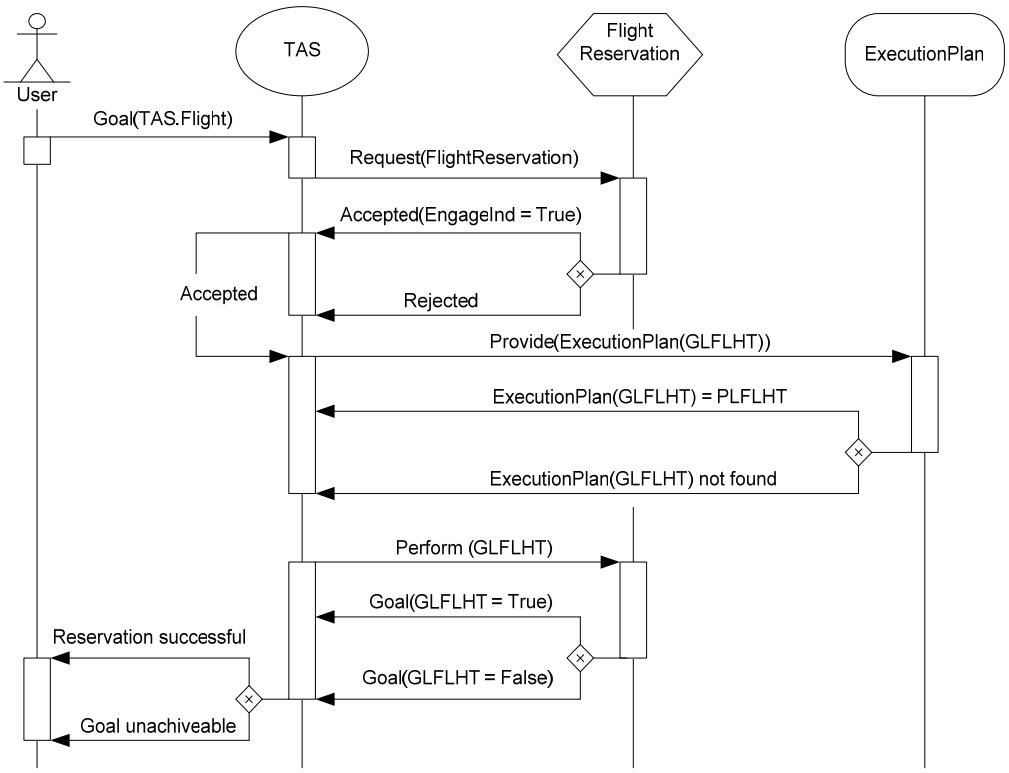

Figure 5.48: Interaction diagram for flight reservation skill-agent.

Both the Hotel booking skill-agent and the Car rent skill-agent are identical to the Flight reservation interaction scenario. The only change is the skill-agent and the goal.

It is also unnecessary to print the combination's goals because they can be driven from the individual goals. Figure 5.47 depicts the Hotel booking skill-agent and below Figure 5.48 plots the Car rental skill-agent.

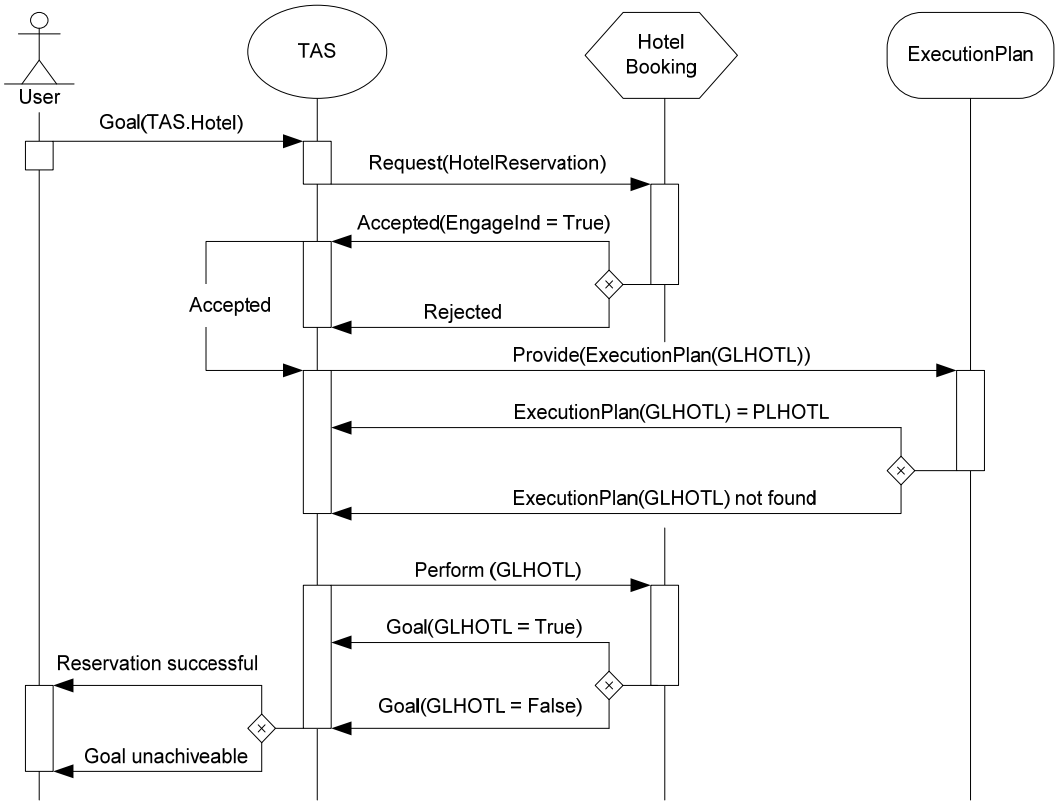

Figure 5.49: Interaction diagram for hotel reservation skill-agent.

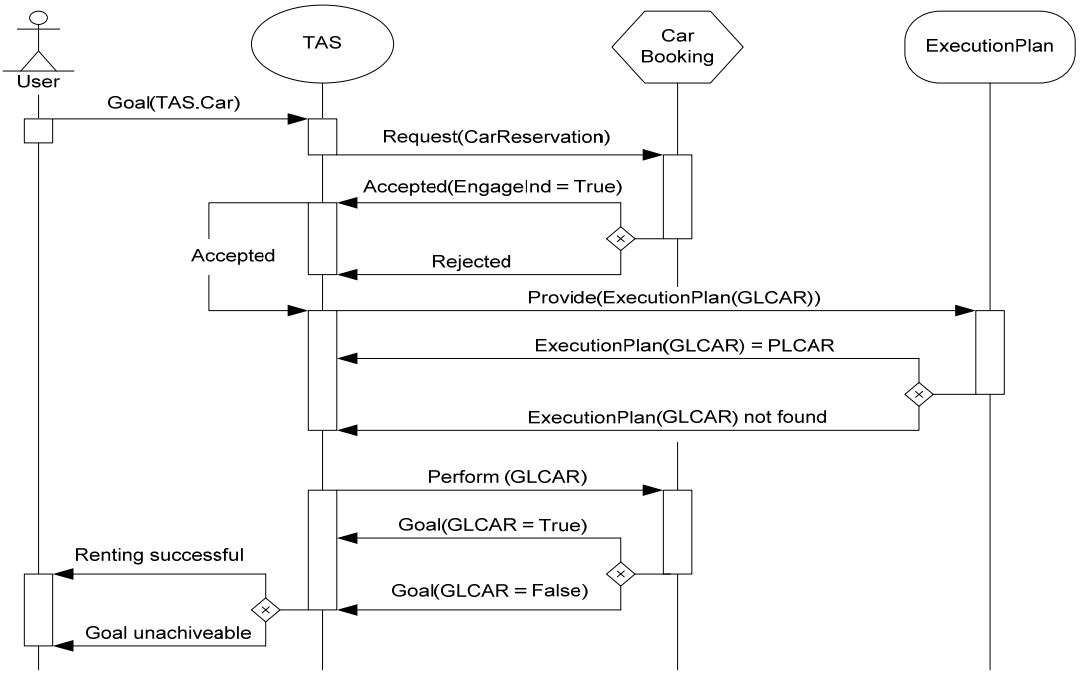

Figure 5.50: Interaction diagram for car rental skill-agent.

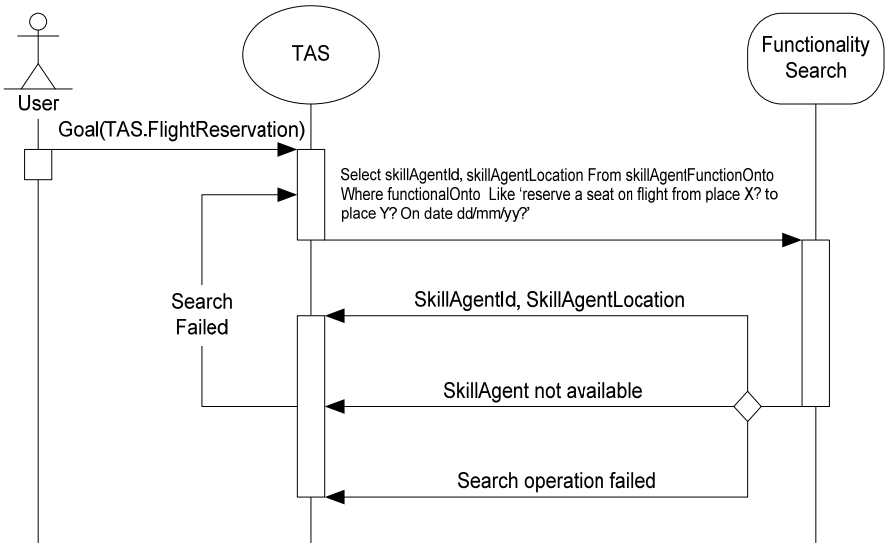

Figure 5.51: Interaction diagram for TAS professional-agent and functionality search.

At this stage of the DMMAS development methodology, all the necessary TAS analysis and design artefacts have been modelled. Some design artefacts are detailed up to implementation phase, for example the goal SQL sentences, the goal execution plan database relationships and the goal execution schema. This because these models must be verified before further details are included. These system plans are the basic core models in the system development and system runtime therefore it is essential to acknowledge the modelling process prior to the implementation phase. For these reasons, DMMAS enforce an overlapping slot between the detailed design and the system implementation.

Select code for TAS skill-agents functionality: This step demonstrates some selected implementation code for TAS skill-agents functionality. The selected XML codes listed in Figure 5.50 and an example for skill-agents (Flight, Hotel, Car) functionality definition which is divided into descriptive and behaviour as per the functionality components set up in the DMMAS design phase. Each functionality node is defined by the same tags' description and attributes to form a standard definition. This XML code will be translated using XML-Schema to set in a document description, then developed further to meet the skill-agent functionality ontology definitions. The functionality definition, the search, and the skill selection are the core components that facilitate DMMAS unique approach overall the existing agent-based development methodologies.

```
<?xml version="1.0"?>
<TASSkillAgentFunctionality>
	<FlightReservation>
		<descriptive>
			<process>Reservation</process>
			<group>Flight</group>
			<division>Travel</division>
			<section>Place</section>
			<unit>Seat</unit>
		</descriptive>

		<Behaviour>
			<action>ReserveSeat</action>
			<subject>Travel</subject>
			<object>Flight</object>
			<instance>FlightNo</instance>
			<constrain1>Date</constrain1>
			<constrain2>Place</constrain2>
		</Behaviour>
	</FlightReservation>

	<HotelBooking>
		<descriptive>
			<process>Booking</process>
			<group>Accomodation</group>
			<division>Hotel</division>
			<section>Room</section>
			<unit>Type</unit>
		</descriptive>

		<Behaviour>
			<action>Booking</action>
			<subject>Hotel</subject>
			<object>Room</object>
			<instance>Dweller</instance>
			<constrain2>Place</constrain2>
			<constrain1>FromDate</constrain1>
			<constrain2>NumberOfNights</constrain2>
		</Behaviour>
	</HotelBooking>

	<CarHire>
		<descriptive>
			<process>Rentar</process>
			<group>Transport</group>
			<division>Car</division>
			<section>Typr</section>
			<unit></unit>
		</descriptive>

		<Behaviour>
			<action>Hire</action>
			<subject>Transport</subject>
			<object>Car</object>
			<instance>Reter</instance>
			<constrain2>Place</constrain2>
			<constrain1>FromDate</constrain1>
			<constrain2>ToDate</constrain2>
		</Behaviour>
	</CarHire>
</TASSkillAgentFunctionality>
```

Figure 5.52: XML selected code for TAS Skill-agent functionality.

5.7 Summary

This chapter demonstrates DMMAS deployment on the Travel Agency System (TAS) case study. This practical example provides proof of concept of DMMAS and its potential for developing open cooperative multi-agent systems that operate in distributed heterogeneous computational environments. The TAS case study with three subsystem architectures is situated in distribution environment and characterised by rich coordination constraints that interface with DMMAS design and development multi-agent systems. Throughout, the DMMAS development lifecycle (requirement analysis, specification analysis, architecture design, and detailed design) was able to deliver TAS internal structure and its component details forming open cooperative multi-agent systems.

This chapter also examines the DMMAS unique approach in incorporating an ontology approach for defining skill-agent functionality, system goals and problem domain knowledge. Incorporating ontology engineering into the DMMAS development process was attempted successfully the new diagrams and graphical notations introduced for this purpose. This chapter sets out the experimental DMMAS development approach and processes, then concludes that DMMAS is able to deliver TAS into the proposed multi-agent systems architecture. The results are presented and further evaluation and assessment of DMMAS discussed in the following chapter.

Chapter 6 Evaluation and Assessment

6.1 Introduction

This chapter evaluates DMMAS with respect to analysis and design processes from technical and modelling perspectives. The technical perspective includes process and technique related criteria. The modelling perspective includes model related and supportive features criteria. This evaluation perspective is organised, based on agents development methodologies evaluation framework (AMEF) proposed by Seller and Giorgini (2005). AMEF evaluation criteria are formed out of identifying and integrating various analysis features of several methodologies evaluation frameworks. To reinforce the evaluation framework additional tabulated comparison criteria are added to expose DMMAS modelling support, modelling strength and weaknesses, application domains, and usability using diagrams and notations. These tabulated criteria are proposed by (Alhashel et al., 2007, Luck et al., 2004). The chapter lists criteria using graphical means to assess the direction of the methodology in relation to the MaS typology based on whether it is independent or cooperative.

Since these measurements are qualitative, the benchmarking test bed method is an efficient instrument to apply. In this regard DMMAS is compared with the closest five existing MaS methodologies (Gaia, Tropos, Prometheus, PASSI, and MaSE). The second part of this chapter discusses the research limitations and the possible future enhancements to improve DMMAS performance.

6.2 The Evaluation Framework

The structure shown in Figure 6.1 highlights the major components of the evaluation framework. Every component targets particular assessment criteria and is accomplished by a dedicated criterion that relates to the assessment aim. The evaluation framework has been selected for its comprehensive test and accuracy measurements along with robustness and clarity. In addition, the AMEF has been tested and implemented on a large number of development methodologies for example “Comparison of Ten AOM” Quynh and Low (2005). The feature analysis framework was formed by identifying and integrating the evaluation criteria from

various feature analysis frameworks, including those for assessing conventional systems development methodologies represented in Hederson-Sellers and Giorgini (2005).

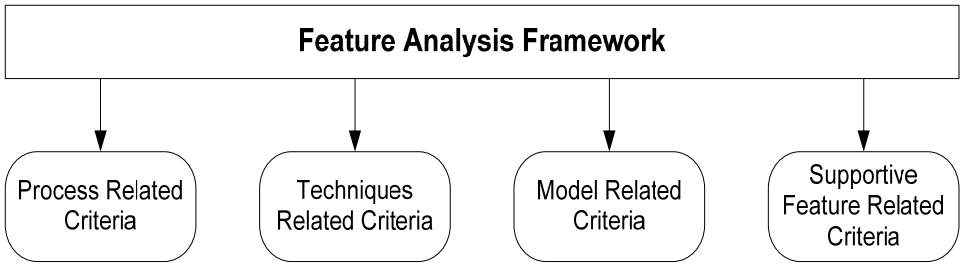

Figure 6.1: Structure of the evaluation framework.

The four major components of AMEF are:-

Process-Related Criteria: evaluating DMMAS in comparison to the existing multi-agent systems development methodologies focusing on the processes involved in each phase.

Technique-Related Criteria: These criteria for assessing the methodology techniques to perform development steps and/or to produce models and notational components.

Model-Related Criteria: test the potentiality and characteristics of the methodology models and notational components.

Supportive-Feature Criteria: examining a set of attributes of high-level methodology capabilities.

6.2.1 Process Related Criteria

The process related criteria are complemented by three modelling qualities; the applicability of the methodology, the steps involved in the development process, and the methodology approach. To construct a measurement scale, each modelling quality was translated into a range of criteria within its assessment properties. Table 6.1 presents the process related criteria and the assessment values for modelling quality:-

Development lifecycle: Classified the development methodology building strategy or approach, for example waterfall, iterative, spiral.

Coverage of the lifecycle: Determine the coverage range of the development process in relation to the software development lifecycle phases.

Development perspective: The development strategy that followed, for example top-down, bottom-up, or hybrid.

Application domain: In AOSE as yet there is no common application domain methodology. However, this criterion specifies the class of the application domain that the methodology can build for example, robotics, business, and knowledge.

Size of MAS: Multi-agent systems are subject to a limited number of agents that can be managed and controlled.

Agent nature: Agent has a multiplicity of natures for example, heterogeneous, homogeneous, mobility. These criteria define the agent nature that the development methodologies generate.

Support for validation and verification: Is the methodology phases or sub-phases containing a validation or verification to maintain the design consistency.

Steps in the development process: the tasks and activities involved in the modelling process.

Ease of understanding the process steps: The level of understanding of the methodology steps and graphical support.

Usability: The extent of the consistency in the methodology steps, graphical diagrams, and key notations, that enable the developer to function smoothly and get familiar with its development processes.

Refinability: To what level the development process gradually and in clear hierarchal paths builds and constructs the required application.

Approach towards MaS development: defines the approach that the methodology is based on. For example (a) Object Oriented, knowledge engineering, or native agent-based. (b) Approach towards using “role” in MaS development, for example does the methodology deploy the concept of “role” in the MaS analysis and design practice. If yes, then what is the approach followed to identify the roles? There are three main approaches for roles representation - goal-oriented, behaviour-oriented, and organisation-oriented. However, DMMAS deploys a combination of organisational and goal-oriented roles representations.

	Existing Methodologies					DMMAS
Criteria	**Gaia**	**Tropos**	**Prometheus**	**PASSI**	**MaSE**	
Development lifecycle	Iterative within each phase but sequential between phases	Iterative and incremental	Iterative across all phases	Iterative across and within all phases (except for coding and deployment)	Iterative across all phases	Iterative within each phases
Coverage of the lifecycle	Analysis and design	Analysis and Design	Analysis and Design	Analysis, Design and Implementation	Analysis and Design	Requirement Analysis, to Detailed Design
Development Perspective	Top-down	Top-down	Bottom-up	Bottom-up	Top-down	Top-down
Application domain	Independent (business process management, GIS, traffic simulation)	Independent (e.business systems, knowledge management, health IS)	Independent (colonic manufacturing online bookstore)	Independent (distributed robotics applications, online bookstore	Independent (distributed planning, databases integration system, computer virus immune system automatic control)	Independent (e.business large scale open distributed heterogeneous system
Size of MAS	<=100 agent classes	Not specified	Any size	Not specified	<= 10 agent classes	Any size
Agent Nature	Heterogeneous	BDI-like agent	BDI-like agent	Heterogeneous	Not specified but possibly heterogeneous	Independent autonomous agent
Support for Validation	No	Yes	Yes	Yes	Yes	Yes
Understanding of process	High	High	High	High	High	High
Usability	Medium	Medium	High	High	High	High
Refine ability	Yes	Yes	Yes	Yes	Yes	Yes
Approach	a.OO b.RO (OrO)	a. I* modelling framework b. NRO	a. OO b. NRO	a. OO b. RO	a. OO b. RO (GO)	Combination

OO = Object-Oriented,
RO = Role-Oriented,
GO = Goal-Oriented,
NRO = Non Role-Oriented,
KE = Knowledge-Engineering,
BO = Behaviour-Oriented,
OrO = Organisational-Oriented.

Table 6.1: Comparison of Process-related criteria.

Table 6.1 highlight the DMMAS process-related criteria, and it also compares DMMAS with five existing multi-agent systems development methodologies. The comparison results reveal that there are new process-related approaches in the analysis and design of DMMAS as listed. Mainly, the agent nature criteria where DMMAS deal with independent autonomous agent and the others are dedicated to heterogeneity or belief-decide-intention (BDI) agent structure. DMMAS is not concerned with the agent intra structure, instead it deals with the application domain to create a cooperative environment regardless of the agent intra structure. Creating agents cooperative environment is the core of MaS concept where the existing methodologies instead are concerned with independent agent development.

Modelling agents using the “role” approach whereas in the conventional agent-based development strategy the role is represented by an agent. This trend is followed by most of the existing methodologies explicitly or implicitly. For example, Gaia,

Prometheus (in grouping of functionality), Tropos (merging role and position), PASSI, and MaSE which built on the assumption that the system goals will be satisfied if each goal maps to a role, and every role is played by at least one agent class (Hederson-Sellers and Giorgini, 2005). But DMMAS underlying system design is not role based instead it dealing with an agent as a unit of abstract. In DMMAS the system is dynamic organisational structure that defines agent as a coherent service unit to satisfy a sub-goal or a goal in the system. This approach emphasises agent independency consequently an agent is not necessarily to be developed locally or deployed locally, or follow the DMMAS development methodology. The only condition is that the agent functionality (for services) must be defined and identified in the functionality ontology component, and must be prepared with the Accept/Reject component. Generally, this approach works as a standardization to incorporate heterogeneous agents to operate in distributed environment.

6.2.2 Technique-Related Criteria

Technique-Related criteria are divided to two parts. This first part is organised to formulate a measurement for the methodology's usability and availability. The criteria extended to examine the methodology deployment steps including graphical diagrams and formed notation. The technique related criteria originated with two refinement questions: What are the techniques used to perform each step? What are the techniques used to develop each graphical diagram and notation used? On the other hand, for the usability assessment the test focuses on the extent of ease of use in the context of:

- understanding the techniques (it reflects how the techniques easy to understand),
- ability to follow the techniques,
- whether there are examples to illustrate the techniques and simplify the modelling process.

Tables 6.2 and 6.3 are complementary and present the evaluation framework for DMMAS. The criteria in both tables are prepared to stress the methodology usability and availability in individual forms.

The ease of understanding and usability of the methodology's steps are ranked as High "H", Medium "M", or Low "L". The Inputs/Output columns specify the methodology supported steps either input or output or both, consecutively donated as "I, or O, or B". Finally, the usability of the techniques are summarised in Table 6.3 to provide equivalency comparisons. In Table 6.2 and Table 6.3, the first column list a set of criteria that serve as an assessment checklist and the first top row presents the techniques, steps, and usability attributes measure. These steps are not necessarily performed by the methodology or it may not perform then directly or in some cases the step is performed implicitly or is stretched over a number of processes.

DMMAS								
Steps	Supported?	Models/ Notational components?	Inputs Outputs?	Techniques for step	Techniques for modelling	Ease of understanding	Usability	Examples
Identify system goal	Yes	System goals diagram, Goal refinement diagram	B	In early requirements, identify goals, subgoals, using refinement techniques focusing on how and why.	Show goals dependencies among stakeholders and system in system context diagram, Why and How refinement as object and assignment	H	H	Y
Specify use cases scenario	No	Instead using goal and stakeholder	B	Perform goal scenario from the system scenario	MAS is Goal oriented not case oriented. Compiling each goal and perform Goal discriptor	H	H	Y
Identify roles	No	DMMAS is Skill oriented not Role base	B	From system goals and subgoals, derive the system skill-agent	Develop the skill-agent descriptor by answering the descriptor template	H	H	Y
Identify system task/ behaviour	Yes	Goal Execution Plan	B	Developing a database for every goal in the system	Developing goal execution plan database schema derived from Goal descriptor	H	H	Y
Identify agent classes	Yes	Professional-agent and skill-agent	B	Using professional-agent architecture diagram and defining the skill-agents involved in the domain	Professional-agent domain diagram along with the professional-agent architecture diagram	H	H	Y
Model domain conceptualization	Yes	Skill-agents, Functionality	B	Defining Skill-agents that involves in the system reflect the system domain	Each Professional-agent diagram represent one complete system domain	H	H	Y
Specify acquaintances between agent classes	Yes	Replace by Ontology modelling	B	Ontologies the skill-agent functionality, and using Goal SQL statement structure to search	Skill-agents functionality descriptor, and functionality model Ontology domain model	H	H	Y
Define interaction protocols	Yes	AUML	B	Capturing the accept and reject interaction scenario between professional-agent and skill-agent	AUML (agent UML) techniques and standard	H	H	Y
Define contain of exchanged messages	No	AUML	B	No predefine agent communication language	Treated as True or Fails or Accept or Reject	H	H	Y
Specify agent architecture	Yes	Skill-agent Architecture / Professional-agent Architecture	B	Professional-agent has predefine architecture, the skill-agent treated as independent architecture	DMMAS Professional-agent architecture, Accept and reject model used as interface the skill-agent	H	H	Y

Table 6.2: Evaluation of DMMAS Steps and Techniques.

The DMMAS early requirement process deploys the stakeholder goal and sub-goal scenario to capture and build the multi-agent systems goals. Thus the DMMAS is goal oriented, subsequently the UML use-case is reshaped to deliver goal-case. The idea continues in the role model deployed by all the existing MaS methodologies. The DMMAS proposed a new concept namely skill (services) represented by the skill-agent which replaces the role model. DMMAS introduces the skill-agent to build a loosely coupling open system that can stand as an independent unit by itself and can be invited for goal achievement. The role as designed in Tropos, Gaia, and MaSE is interconnected and must be designed locally. This technique results in designing close agent systems. It is obvious in this assessment schema that DMMAS contains these steps but utilises different original techniques.

DMMAS								
Steps	Supported?	Models/ Notational components?	Inputs Outputs?	Techniques for step	Techniques for modelling	Ease of understanding	Usability	Examples
Define agent mental attitudes (goal, belief, plans, commitment)	Yes	Plan descriptor Data used and Data produced model	B	Follow a predefine plan template, and database schema. For commitment the accept and reject skill-agent architecture must be predesigned	Fill the plan database template, skill-agent commitment model, and goal domain ontology model	H	H	Y
Define agent behavioural interface (e.g., capabilities, services, connections)	Yes	The commitment model, and skill-agent search model	B	The skill-agent search model used as services to provide the professional-agent with the skill for the goal achievement	The relationship between the professional-agent and the skill-agent serve by the services (skill) search model, and Skill-agent commitment model (connection)	H	H	Y
Specify system architecture (i.e., overview of all components and their components	Yes	DMMAS: Multi-agent System Architecture	B	Organisational three levels architecture links by ontology definitions	DMMAS MAS architecture, Ontology domain for linking.	H	H	Y
Model MAS environment (resources, facilities, characteristics)	Yes	Ontology Model	B	The environment defined by the skill-agent functionality ontology model and search	The skill-agent ontology model, the search ontology-based model work as environment	H	H	Y
Agent-environment interaction mechanism	Yes	Skill-agents, Functionality Ontology	B	This ontology is the main link with the external skill-agent which behave in open distributed environment	Each Professional-agent diagram represent one complete system domain	H	H	Y
Specify organisational structure	Yes	Professional-agent, skill-agent model	B	Professional-agent allowed to adopt the skill-agent base on the goal needs.	The professional-agent to the skill-agent adoption structure form an emergent organisational structure	H	H	Y
Instantiate agent classes	No	Irrelevant	Is object-oriented techniques, out of the DMMAS focus. Applicable for methodology that deal with conceal or independent MAS.					
Specify agent instances deployment								
Specify agent inheritance and aggregation								

Table 6.3: Evaluation of DMMAS steps and Techniques.

In equivalent steps in both tables, DMMAS is not performing all of these steps because it has its own unique development approach and design techniques. DMMAS follows a unique approach that models MaS as a cooperative system facilitated by dynamic team formation process. In such situation the agent class techniques as appears at the end of Table 6.3 is irrelevant because it focuses on object-oriented software development techniques rather than traditional AOSE design methodology.

It is important to note that DMMAS pursues a new approach and introduces new abstractions subsequently requiring special measurement criteria equivalent to these new abstractions which are not available within the other methodologies. For this reason the framework original criteria have been extended to stress the DMMAS technical steps with the intention to avoid the unique individual features and emphasis on the generalisations as possible.

6.2.3 Model-Related Criteria

The second part of Model-Related Criteria is the steps and usability of techniques that are depicted in Table 6.3. The table criteria cover all the usability techniques that are used in each methodology. It is not possible to give the rows of the Model-Related criteria table direct values because each methodology has its own approach and could not contain or share every step with the other methodologies. However, the criteria listed in Table 6.4 contain the most common dominator between the methodologies modelling. The ordinal scale (high “H”, medium “M”, and low “L”) is used to assign the values and if the methodology does not support the criteria it takes a not applicable “N” value or is replaced by “R” in case there is alternative usability technique.

Criteria	Existing Methodologies					DMMAS
	Gaia	Tropos	Prometheus	PASSI	MaSE	
Identify the system goals	N	H	H	N	H	H
Identify system tasks/behaviour	M	H	H	H	H	H
Specify use case scenarios	N	N	H	H	H	R
Identify roles	H	N	N	H	H	R
Identify agent clasess	H	H	H	M	H	H
Model domain conceptualization	N	N	N	M	N	H
Specify acquaintances between agent classes	M	M	H	H	H	N
Define interaction protocol	H	N	H	H	H	H
Define content of exchange messages	N	N	L	H	H	N
Specify agent architecture	N	H	H	H	M	Y
Define agent mental attitudes (e.g. capabilities, services, contracts)	N	H	H	M	N	N
Define agent behavioural interface (e.g. capabilities, services, contract)	H	N	H	H	N	N
Specify system architecture (overview of all components and their connections	N	N	H	H	N	Y
Specify organisational structure/control regime/inter-agent social relationships	H	H	N	N	N	H
Model MAS environment (e.g. resources, facilities, characteristics)	M	H	M	N	N	H
Specify agent-environment interaction mechanism	N	N	H	N	N	H
Specify agent inheritance and aggregation	H	N	N	N	N	N
Instantiate agent classes	M	N	L	N	H	N
Specify agent instances deployment	N	N	N	L	H	H

N = Not applicable
H = High
M = Medium
L = Low
R = Replaced by

Table 6.4: Comparison regarding steps and usability of techniques.

The results reflect some positive indications for the usability techniques of DMMAS. The use case scenarios identify roles taking “R” values, thus DMMAS replaces these techniques by the goal scenario at an early stage. Furthermore, DMMAS replaces the roles component with the skill-agents component. As has been stated earlier,

DMMAS is goal oriented in its analysis and design and dynamic organisational in its structure, so the use case scenario is not an option in this modelling direction. Table 6.3 reflects the high value assigned for the use of organisational structure employed in DMMAS. Overall, the DMMAS usability techniques within the criteria and value presented in Table 6.4 provide some advantage in usability and availability.

6.2.4 Supportive-Feature Criteria

The criteria presented in Table 6.5 reflect the multi-agent system analysis and design features (concepts) that presume any MaS methodology must provide a level of support to those features and their modelling. Again, because of different and multiplicity development approaches employed by each development approach it is not necessary to fulfil all the features. However, the assigned values are illustrated in Table 6.5 and the criteria are clear and straightforward.

Criteria	Existing Methodologies					DMMAS
	Gaia	Tropos	Prometheus	PASSI	MaSE	
Completeness/Expressivness	M	H	H	H	H	H
Formalization/ Preciseness	a. H b. Y	a. H b. Y	a. H b. Y	a. H b. Y	a. H b. Y	a. H b. Y
Model derivation	Y	Y	Y	Y	Y	R
Consistency/ Complexity	a. Y b. Y	a. Y b. Y	a. Y b. Y	a. Y b. Y	a. Y b. Y	a. Y b. Y
Ease of understanding	H	H	H	M	H	H
Modularity	Y	Y	Y	Y	Y	Y
Abstraction	Y	Y	Y	N	Y	Y
Autonomy	Y	Y	Y	Y	Y	H
Adaptability	P	N	N	N	N	Y
Cooperative behaviour	Y	Y	Y	Y	Y	Y
Communication ability	N	Y	Y	Y	Y	Y
Inferential capability	N	Y	Y	Y	P	N
Reactivity	P	Y	Y	Y	Y	N
Deliberative behaviour	Y	Y	Y	Y	Y	N
Personality	N	N	N	N	N	N
Temporal continuity	N	N	H	N	N	Y
Concurrency	N	N	N	Y	Y	Y
Human Computer Interaction	N	Y	Y	Y	N	Y
Models Reuse	Y	P	P	Y	Y	Y

H = High
M = Medium
N = Not applicable
P = Possibly
R = Replaced by
Y = Yes

Table 6.5: Comparison regarding model related criteria.

The assigned value in Table 6.5 show a high score for completeness and expressiveness criteria for all the methodologies including DMMAS which have quality documentation support in this thesis. In addition, DMMAS's phases include a coherent number of steps complementing each other to gradually developing the required system. The "replace by" value assigned to "model derivation" criterion indicate to that it is not applicable to DMMAS but replaced by other criterion in the list.

Despite DMMAS's different development approach there is complete support for abstraction, autonomy, adaptability, and cooperative behaviour. These concepts form core of DMMAS. On the other hand concepts like inferential capability, reactivity, deliberative behaviour, and personality are not within DMMAS modelling techniques. There are an important clarification to the existing methodologies (Gaia, Tropos, Prometheus, PASSI and MaSE) assigned values "Yes" for the criteria; autonomy, cooperative behaviour, and communication ability. These criteria assessed these methodologies from a perspective of developing independent close MAS. On the other hand if the criteria are defined to evaluate DMMAS approach then there will be unbalanced comparisons, because the existing methodologies follow different design approach than DMMAS. For example, criteria (inferential capability, reactivity, deliberative behaviour, and personality) are not useful to evaluate DMMAS but they are useful to the rest.

6.3 Comparison of Concepts

The list of concepts depicted in Table 6.6 comprises the most appropriate concepts related to the agent-oriented software engineering field. Any MaS development methodology must cover some of these concepts directly or indirectly. The purpose of this evaluating is to understand the degree of maturity of the methodology by counting the number of native agent-oriented concepts covered by each methodology. The assigned values in the table are descriptive thus the criterion reflects the component name or the model that is capturing or representing the concept. It is difficult to evaluate each criterion for each methodology because these methodologies entailed a different set of measurement criteria that matches with its application domain and the type of agent architecture it designs. Based on this fact the value "not specified" indicates to either the concept is not present or is impeded within the modelling components.

	Existing Methodologies					DMMAS
Criteria	**Gaia**	**Tropos**	**Prometheus**	**PASSI**	**MaSE**	
System goal	Not specified	Actor diagram, Rational diagram	Goal diagram		Goal hierarchy diagram	System goal diagram
System task/ behaviour	Role model	Actor diagram Rational diagram	Functionality descriptor	System requirement model	Extended role diagram	Execution plan descriptor,
Use case scenario	Not specified	Not specified	Use case descriptor	System requirement model	Use case diagram	User goal model
Role model	Not specified	Not specified	Not specified	Agent society model	Role diagram	Goal discriptor
Domain conceptualisation	Not specified	Not specified	Not specified	Agent society model	Not specified	Skill-agent functionality, descriptor, diagram
Agent goal/task	Not specified	Actor diagram	Not specified	System requirement model	Not specified	Goal Skill-agent diagram
Agent-role assignment	Agent model	Not specified	Not specified	Agent society model	Agent class diagram	Skill-agent diagram
Agent architecture	Not specified	BDI archetecture	Agent overview diagram	Agent implementation model	Agent class diagram	Professional, skill agents diagram
Interaction protocol	Interaction model	Sequence diagram	Interaction protocols	Agent society model	Communication class diagram	AUML
System architecture	Not specified	Not specified	System overview diagram	Agent implementation model	Not specified	Organisational architecture model
Organisational structure/inter-agent social relationship	Organisational structure model	Non-functional requirements framework	Not specified	Not specified	Agent class descriptor	Ontology based
Agent-Autonomy	- - - - ►	- - - - ►	- - - - ►	- - - - ►	- - - - ►	Skill-agent ontology domain model
Ontology model	- - - - ►	- - - - ►	- - - - ►	- - - - ►	- - - - ►	Complete development lifecycle

Table 6.6: Comparison regarding concepts.

The DMMAS column reveals the name of the component or the model that capture the standard criteria listed in the first column. In comparison, the DMMAS includes new concepts; professional-agent, skill-agent, ontology (functionality), and goal execution plan. DMMAS also encompass goal model, and UML interaction protocols. There is a clear focus on building MaS in emerging organisational structure dynamically (at runtime) which provides a main advantage to DMMAS. Another advantage for DMMAS is that the concept; agent-autonomy is highly supported by the ontology functionality model which is a unique approach. The dotted line mapped

these two concepts directly to DMMAS because the other methodologies do not employ these concepts. In general, this evaluation schema indicates to the advantage in DMMAS development approach in introducing new concepts for MaS analysis and design.

Performance Measure: The performance measure is to reflect the quality of the methodology from the support and software tool support. The performance criteria measure the extent of supports provided by the methodology in term of helping and guiding the developer to achieve the design process. In fact, this criterion is a type of development quality measurement. Table 6.7 facilitates the purpose of performance evaluation by the nine most important criteria covering the MaS performance. The assigned values follow the binary scale "Yes" or "No". The word "partially" is also used to weigh the values in between.

	Existing Methodologies					DMMAS
Criteria	**Gaia**	**Tropos**	**Prometheus**	**PASSI**	**MaSE**	
Software development tools support	No	No	Yes	Yes	Yes	No
Open system	Yes	No	No	No	No	Yes
Dynamic structure	No	No	No	Yes	Partially	Yes
Agility and robustness	No	No	Yes	No	Yes	Yes
Support for conventional objects	No	No	Yes	No	No	No
Support for mobile agent	No	No	No	Yes	No	No
Support for ontology development	No	No	No	No	No	Yes
Support for agent autonomy	No	No	No	No	No	Yes
Support emergent system	No	No	No	No	No	Yes

Table 6.7: Comparison regarding supportive related criteria.

The DMMAS column shows that the software development tool is not supported. The research scheduled this facility into the future work. At this stage the software development tools for DMMAS are beyond the focus of this research. The support for conventional object criteria is not supported by DMMAS because it is a native agent-oriented concept. For further explanation, the next section discusses the importance of an agent-based system being facilitated by tailor made tools and techniques that can

map its abstraction. Finally, the agent mobility is not supported by DMMAS. In fact agent mobility is type of an agent system approach where DMMAS is the focus of the dynamic integration in organisational structure. The others performance criteria show the potentiality of DMMAS supports in terms of ontology, autonomy and emergent system.

6.4 DMMAS Strengths and Limitations

DMMAS provides guidelines starting from the system early requirements through detailed designs for building cooperative multi-agent systems that function in open, distributed, heterogeneous environment applications. The key element of DMMAS is the ability to dynamically (at runtime) crate a team of agents to achieve the user goal. The modelling principles of DMMAS formed around organisational system structure consist of professional-agents situated on the node adopting the required skill-agents that are situated on the leaf. This approach is unique in the field of the MaS software engineering and accountable to this research.

DMMAS employed an ontology approach to model the skill-agents functionality then inferred this ontology model to find the appropriate skills for achieving the goal. This technique is innovative in the MaS design in addition; it provides substantial system flexibility to the agent to act autonomously.

PASSI, MaSE, Tropos, and modified version of Prometheus have used ontology for agent communication models to share the domain knowledge as part of KIF, or KQML ontology base communication language. At the same time none describe how to develop the ontology domain in these methodologies. The DMMAS use the ontology to define the skill-agent and the goal functionality with full support on how to model this ontology starting from the early requirements to the detailed design, in parallel with the system development lifecycle facilitated by a new techniques and notation.

Based on the research experiment explained in Chapter 3 and Alhashel (2007), methodologies like Gaia, MaSE, and PASSI are difficult to map their detailed design into implementation. The final artefacts float between the design and implementation.

In addition only Tropos covers the software engineering development lifecycle as it appears in Figure 6.2 as in Bresciani et al. (2004).

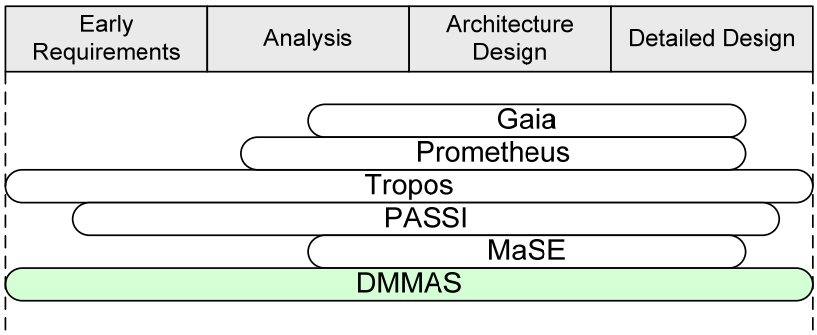

Figure 6.2: Comparison between DMMAS and rival development methodologies on software life cycle completeness.

In the case of DMMAS and as described in Chapter 5 the detailed design maps smoothly into the implementation process. The most important software engineering phases are the analysis and design. DMMAS covers the development phases and provides a full documentation to guide the software developer to build the system.

The last strength of DMMAS is that it does not guide the system to follow any agent architecture in the lower levels of the DMMAS architecture (skill-agents levels). There are no restrictions on which type of agents can be included in the system since it is defined in the ontology model. DMMAS is supported by an innovative dynamic organisation architecture which helps the developer decided where and when to deploy DMMAS.

Finally, DMMAS is configured based on multi-agent systems open cooperative concepts and facilitated with AUML interaction diagrams and the FIPA standard. DMMAS approach entirely focuses on the MaS abstraction and constraints. This originality improves the MaS analysis and design practice and delivers standard abstraction.

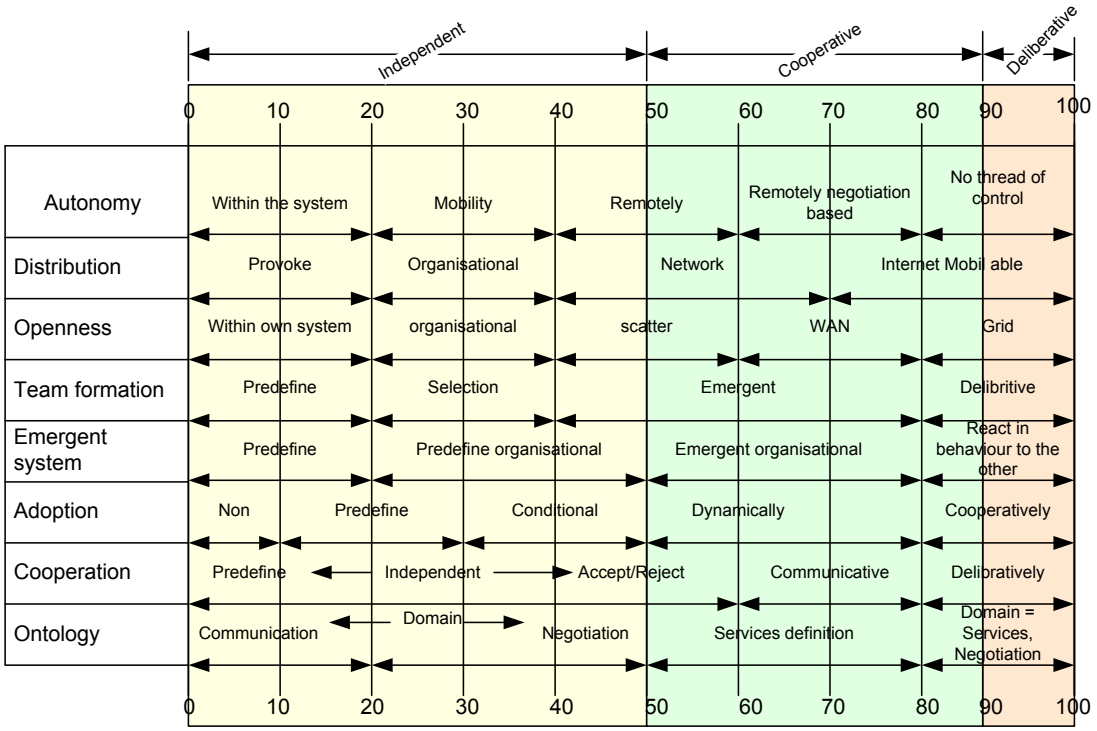

Table 6.8: Measurement scale for assessing multi-agent systems type.

Table 6.8 sets out the translation criteria for multi-agent system typology and classification. The criteria are scaled according to the importance of modelling within each methodology in relation to the multi-agent system classification.

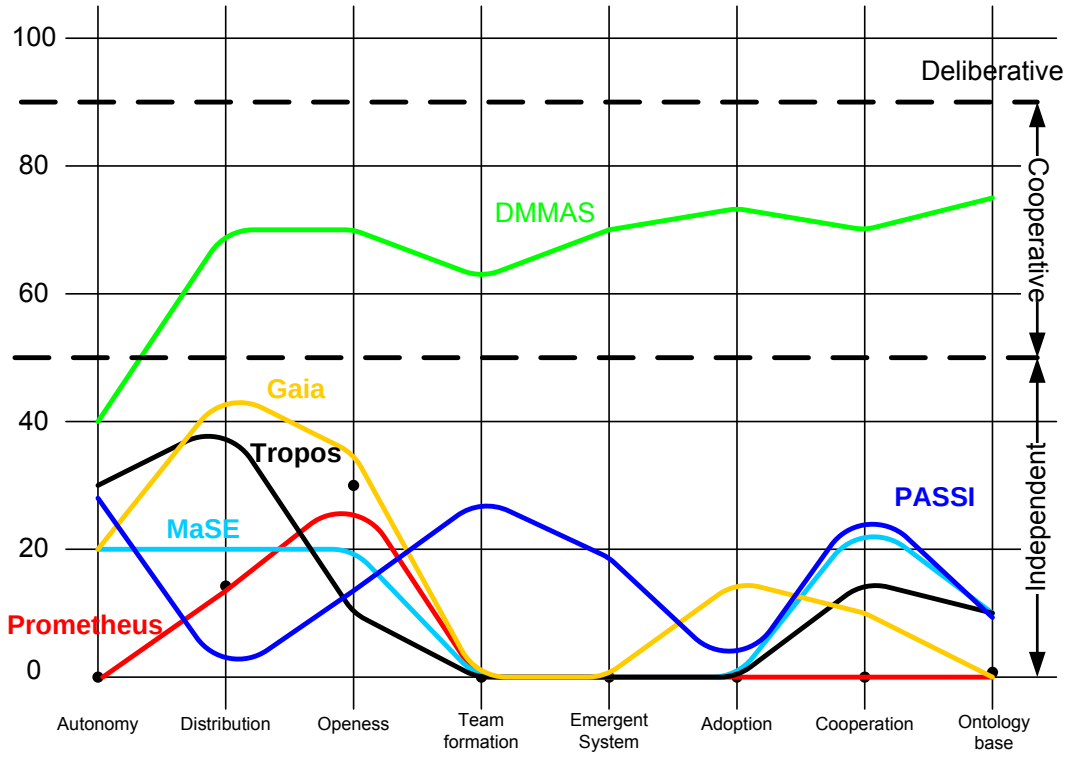

Figure 6.3: Graph classification of development methodologies.

The criteria are identified as qualities and components of multi-agent systems which use existing methodologies design inside one system boundary or close MAS. Figure 6.3 illustrates these criteria in a graph and indicates DMMAS design approach toward cooperative MaS. The graph provides a clear comparison between the existing MaS methodology approaches and the DMMAS development approach. The deliberative type appears on the top of the scale beyond the research scope and it relates to the agents cooperation based on deliberative behaviour and shared or mutual understanding between the software agents.

Weaknesses: While DMMAS provides significant advantages for building cooperative multi-agent systems, it is not without weaknesses. The goal execution plan in DMMAS is complicated to develop for each goal. In addition the execution plan is predefined component developed using the goal plan descriptor and the goal execution diagram. This means for any additional goal the execution plan must be

modified and reloaded. However, it could be very practical if the goal execution plan is standardised and automated.

The DMMAS accept/reject component that establishes communication between the professional-agent and the skill-agent can be enhanced if the professional-agent broadcasts its required skills (as invitation) to all/any skill-agents. The agent communication languages, for example KIF, KQML, or ACL, could be deployed to facilitate the invitation modelling. This design trend will provide additional flexibility to the system.

In an open, distributed, scalable system, the skill-agents could be replicated in terms of similar skills operated in the same system environment. In such cases conflict resolution utility is essential to coordinate between similar multi skill-agents. However, the current design of DMMAS presumed there is unique skill-agent in equivalent to a unique functionality domain in the ontology library.

With a large scale methodology like DMMAS with many development steps mistakes are likely to happen. Software development tools that govern the development process and detect errors and maintain consistency, will increase efficiency to the development practice. For example, Prometheus development tool (PDT) provides efficient facilities and options including skeleton code generation in JACK agent development language.

Finally, DMMAS introduced a set of new diagrams and notations that are not based on any existing modelling approach. This can be regarded as being either a strength or weakness depends on the practitioner background and experience. However, the diagrams and the notations used in DMMAS are able to translate the multi-agent systems unique abstractions and blueprint the components.

6.5 Summary

This chapter evaluated and assessed DMMAS analysis and design processes on four levels; technically, modelling, usability and potentiality for building open cooperative

multi-agent systems. The evaluation criteria were based on agents' development methodologies evaluation framework (AMEF) and extended by additional tabulated criteria that proposed to stress DMMAS new features. The assessment process deploys benchmark measurement scales for DMMAS performance in comparison to five existing MaS methodologies respectively, Gaia, Tropos, Prometheus, PASSI, and MaSE. Despite the differences between each individual method, the criteria of the evaluation framework facilitated have been selected to measure the common activities as possible. In addition, each method was evaluated individually then determined its strength and weaknesses.

The evaluation process reveals that DMMAS successfully presents a new and unique approach to design open cooperative multi-agent systems with the potentiality to span the design practice to utilise an ontology approach as part of the development processes. It also reflects that DMMAS new design techniques lead to transfer MaS from the independent to cooperative application domain. Moreover, the results provide scale of measure for the performance and approaches within each methodology and in overall comparisons. While some methodologies focus on modelling agent internal structures others focus on designing agent interaction protocols. In contrast the agent cooperation design activities are minimally covered. The evaluation process on DMMAS also determined some weaknesses, and subsequently suggested some improvements to enhance the development practice. These improvements proposed in the research future work are explained in the next chapter.

Chapter 7 Conclusion and Future Work

Several agent-based development methodologies have been proposed. However, all converge towards designing an independent closed multi-agent system emphasising the agent internal structure or the agent coordination and interactions protocols. The primary aim of this research was to transfer MaS from an independent state to a cooperative state (as summarised in Figure 7.1). For this purpose, the research proposed a new development methodology for multi-agent systems with potential to build cooperative MaS that can function in open, distributed, heterogeneous application environments.

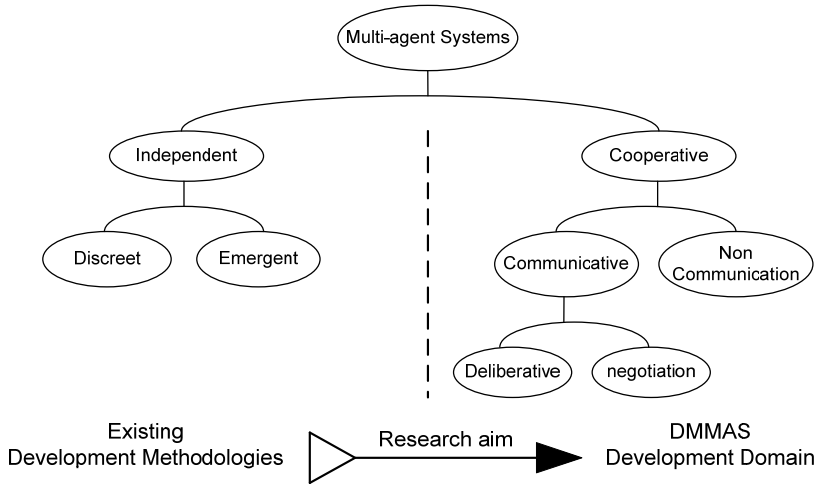

Figure 7.1: Multi-agent systems typology.

The thesis presented a development methodology for developing multi-agent systems (DMMAS). DMMAS provides a set of steps and guidelines for the software practitioner to design and develop a cooperative multi-agent systems based on software development lifecycle principles. DMMAS analysis and design life cycle, starting from early requirements analysis, are followed by system specification analysis, architecture design, and the detailed design phase. For every development step, there is sufficient explanation on what and how to progress in the building process, supported by unique set of diagrams and graphical notations. Since the multi-agent systems possess high level expressive abstractions, the research introduced a set of textual descriptors to express those abstractions.

The research proposed a new architecture for multi-agent systems constructed based on emergent organisational structure and an agent adoption strategy. The architecture consists of three levels - first level represents the goal and the coordination (goal execution plan), the second level represents the professional-agent the goal domain and team master and the third level is the skill-agent or the services required to achieve the goal. Every system goal or goals depends on their domain, there is a professional-agent responsible for the goal achievement. Each professional-agent is one unit coherent application domain acting as master manager with the ability to adopt any group of skill-agents required to achieve the goal.

Each single skill-agent represents one logical unit coherent service acting as a standalone autonomous subsystem or module that can exist at any place in the network. Each skill-agent's functionality is represented by a standard ontology schema designed to acknowledge the skill-agent functionality. The ontology schema defines the skills, or the services, that the skill-agent can offer for the goal achievement. The schema is also complemented by the skill-agents identification and location (address) on the network.

Unlike all the existing methodologies, DMMAS introduced an innovative approach to represent agent autonomous behaviour through the ontology mechanism. Using this technique the skill-agent can serve any goal depending on the professional-agent request. This technique improves the search mechanism and makes it functionality based instead of agent-name keyword based as that used by JADE, PASSI, JACK, Prometheus and Tropos. DMMAS do not focus on the skill-agent intra structure or agent communication protocols that require high bandwidth and form a bottleneck; instead it provides an Accept/Reject component for every skill-agent. For example, according to the JointPlan cooperative theory, if a set of agents aligned their plan in the correct execution order toward the user goal achievement, in fact they are in cooperation mode. Chapter 5 introduces the TAS case study and Chapter 6 demonstrates a comprehensive evaluation of DMMAS. Both chapters provide good evidence of the DMMAS use in transferring the multi-agent systems to the cooperative stage base on the goal execution plan and agent adoption strategy. The agent adoption strategy delivers the team formation process dynamically. This is an advantage feature for using DMMAS.

7.1 Main Contributions

This research reviewed the software system engineering practice and investigated the existing agent-based development methodologies to experience their techniques and approaches for modelling multi-agent systems features. The research introduced a new direction for the use of ontology in software design and development processes. In the field of agent-based software engineering, the research updates the discipline development tools with a new development methodology, namely DMMAS that has the potential to analysis and design cooperative multi-agent systems. The primary aim for this research is to fill the gap in the multi-agent systems to transfer the development process from the designing independent multi-agent systems to designing cooperative multi-agent systems. The research outcomes result in a set of steps and processes to guide the practitioner to deploy a multi-agent systems that is best applicable when the problem domain has an open system characteristic and consists of heterogeneous software agents located in distributed locations and needs to work together in a cooperative mode to achieve a particular system goal. Building such system specifications using agent-based software concepts and incorporating the ontology approach is an innovative addition to software engineering approach.

This research presents three main contributions;

1. This thesis enhances the multi-agent systems practice by providing a new development methodology with potentiality to transfer the multi-agent systems developments from designing an independent multi-agent system to cooperative multi-agent systems based on the goal plan, and agent adoptive strategy.
2. The research presents a new multi-agent systems architecture that informs the software body, how to design open distributed heterogeneous multi-agent systems.
3. The research informs the software development practice about how to incorporate ontology engineering into software engineering. The research presents a new method and diagrams notation to identify the model ontology concepts, objects, properties and classes relationships for functionality domains.

4. The research introduces a new agent structure that can behave in an autonomous mode. In this regard the research eliminates the predefined agent name (keyword) search in favour of functionality ontology search. This search approach is a new trend in the field that could enable the software practitioner to link and integrate a vast multi-disciplinary system, for example military, health care, and transportation.

7.2 Research Questions

This research is classified as qualitative research in information science design and has been conducted under combination of Hevner et al's (2004) framework for design science in information systems and Simon's (1996) iterative approach Generate/Test cycle. The research method was implemented successfully and was able to achieve the research aim and answered the research questions and states the hypothesis:

- What is a mechanism that has the ability to create agent team formation processes?

The proposed multi-agent system organisational architecture with the three portions layer, then incorporating the ontology modelling to create ontology-based search supported by a goal execution plan provide an ideal mechanism that identifies the required agents and forms the agents team.

- What is the software architecture of cooperative agent-based system?

The research published a research paper in this research question direction; "An Architecture for Agent-based Cooperative Systems" can be found in (AlHashel et al., 2009a). It is also covered in Chapter 4.

- What is the suitable software engineering model that seamlessly engineers cooperative agent-based systems?

The DMMAS with the unique ontological modelling, including the adoption strategy between the professional-agent when it overrides the skills-agents that is required to achieve the goal. Then using the goal plan to perform the goal, this intra processing with the support of the analysis and design phases depicted in Chapter4, and Chapter

5, is proof that DMMAS is an adequate methodology for engineering cooperative multi-agent systems.

The hypothesis is verified as follows:

- The existing agent-oriented development methodologies are not able to engineer multi-agent systems cooperation behaviour dynamically at runtime.

The evaluation and assessment processes for the existing development methodologies presented in Chapter 3 confirm this hypothesis. In addition, the research exposes the strengths and weaknesses of three well-known development methodologies and proves that the agent cooperation is not a runtime or emergent process. However, this research published a paper titled "Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE"; for confirmation refer to (Alhashel et al., 2007).

- DMMAS techniques for executing a goal is done in the following steps: first, access the goal execution plan, secondly, find the proper skill-agent required for the goal achievement, thirdly, the process is looped and at every iteration, the skill-agent identification, name, and location are registered to build the execution registry, that is this process is like gluing each skill-agent plan recipe with the other. This process exactly defined the Share Plan cooperative theory.

- The existing development methodologies need fundamental changes and enhancements to design cooperative multi-agent systems, therefore a new tailor made development methodology is the solution.

This hypothesis is stated to confirm that if the research can modify an existing methodology then it is possible to avoid reinventing the wheel. Based on this assumption the research modified Prometheus to incorporate agents cooperation; the result is published in "Enhancing Prometheus to Incorporate Agent Cooperation Process" (Alhashel et al., 2008).

7.3 Limitations

The research is driven by an appropriate research methodology with a potential to achieve the research goals and objectives and is able to address the research questions and test the hypothesis. Despite some success, throughout the research some limitations have been encountered, mainly:

1. The research timeframe: the nature of an agent-oriented paradigm is heterogeneous, composed of multi computational disciplines like AI, knowledge engineering, object-oriented, and DBMS. This unique combination required wide knowledge involving a variety of technologies and science including ontology paradigm. Moreover, research in designing such system is time consuming to overcome all the modelling and development aspects. For example Prometheus development methodology consumed seven years of group work to be established. However, the research could not proceed further to include a comprehensive implementation phase, because of timeframe limitation.

2. The manpower limitation: another restriction is that the research was restricted to the work of one researcher. On the other hand the agent-oriented paradigm includes a wide range of research areas and with one research student it could not cover every related aspect of the agent-oriented system. For example the research has not explained the implementation of the agent communication languages and the communication domain ontology or conflict resolution for agent coordination. Because these are research topics by themselves and need a larger research team if such implementation depth is required. However, the DMMAS covers the most important phases in any software engineering development process, the analysis and the design phases.

3. Development technologies constraint: the agent-oriented paradigm is immature and as yet there are no standard software development tools and infrastructure technologies to translate the designed intra- and inter-agent infrastructures into formal, computational specification languages or runtime environment. The software support can ease the implementation process and impose the development process to model the system components and abstractions to directly

align with the implementation libraries and classes. As a result of this limitation, the experiment depicted by Chapter 5 deployed DMMAS to engineer TAS then for the implementation phase, the research used database management system, object-oriented technology and web technology. This technology combination limited the research implementation from monitoring the agent behaviour to associate a smooth debugging and system exceptions handling, furthermore, it affected TAS system integration and system performance.

7.4 Future Work

Taking into consideration the research limitations in timeframe, manpower, and development technology, it was not possible to cover all the relevant software engineering requirements. However, the research has been driven by a well-defined plan and was successful in delivering the expected outcomes and able to answer the research questions. Further research work is needed to enhance and improve the DMMAS deployment and in addition, improve the multi-agent systems design and development practice.

To enhance this research and outcomes further extension is needed as below:

- Software tool support is very important to any development methodology (Lin, 2005). The DMMAS development process is characterised by implementation of the ontology which has several different descriptors in the design process. In addition, the nature of the multi-agent system entails a mixture of procedures including a database to define the execution plan. In this situation it is not practical to rebuild the entire process at every new system development, furthermore consistencies between different phases are essential to avoid fundamental run over mistakes. Therefore a software development tool to support DMMAS is highly recommended.
- The ontology is a premature discipline in the field of software engineering and subject to more research interest specifically in the web semantic area. This research uses the ontology to define the skill-agent functionality and the goal structure. Therefore, further study and investigation is required to enhance these models in two directions. First, further research is needed to define an ontology template that can be used as a general standard for all types of

functionalities and goals. Secondly, create a built-in class model for the ontology structure to be integrated with future development in order to establish a reuse ontology global model for DMMAS developers. This process will help in extending the multi-agent systems abilities to function in large scale wide generic systems for example, the Internet.

- The goal execution plan needs further investigation to automate the relationship between the system goals and the plan that executes the goal. Currently DMMAS uses a predefined plan incorporated in the system database which uses a SQL query statement to fetch the plan corresponding to the goal. However, instead of using predefine SQL select statement to search for the execution plan, it would be more efficient to generate this execution plan based on a particular algorithm then generate a coordination model for use by the system goal execution model.

The multi-agent systems is a complicated software paradigm and entails further research effort mainly to fill the gaps in the development tools that can ease deployment practice. This research introduces a new direction in defining the agent functionality (what it does) using an ontology approach. This research provide rich starting point for creating a software system that is built by different people from different places at different times but still can integrate to solve the goal. To make this software environment possible, standardisations gain the most essential requirements. With the enhancement recommended above the research recommend that DMMAS analysis and design techniques lead to becoming a standard development methodology for multi-agent systems.

Appendices

Appendix A: XML and XML-Schema selected code for goal definition, skill-agent definition and skill-agent functionality structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<GoalPhd xmlns="c:\phdXMLProject\goalphd"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="c:\phdXMLProject\goalphd GoalPhd.xsd">

<Domain>Education</Domain>
<GoalId>GOL-ACHV-1</GoalId>
<Name>AchievingPhDDegree</Name>
<Constraints>
        <Boundery>Enrolment</Boundery>
        <Subject> Information Science and Engineering</Subject>
        <Kind>PhD</Kind>
        <Item>A2235</Item>
        <Limit>
                <DateFrom>2011-02-17</DateFrom>
                <DateTo>2014-10-31</DateTo>
        </Limit>
</Constraints>
</GoalPhd>
```

Figure A-1: XML document for "*GoalPhD*" of *AchievingPhD* example.

```
<?xml version="1.0" encoding="UTF-8"?>

<SkillPhd xmlns="c:\phdXMLProject\skillphd"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:xs="http://www.w3.org/2001/XMLSchema"
          xsi:schemaLocation="c:\phdXMLProject\skillphd SkillPhd.xsd">
<Domain>Education</Domain>
<SkillId></SkillId>
<Name>Enrolment</Name>
<Status>
        <InputOutput inputName="Applicant">Applicant</InputOutput>
</Status>
<Status name="getStudent">
        <InputOutput name="Student">Output-Status</InputOutput>
</Status>
<OperationType name="enrolPhdCourse">
        <Function name ="getEnrolPhD">
        <Input Status="getApplcation"/>
        <Output Status="getStudent"/>
        </Function>
</OperationType>
</SkillPhd>
```

Figure A-2: XML document for skill-agent functionality, *Achieving PhD* example.

```
<?xml version="1.0" encoding="UTF-8"?>
<SkillFunctionality xmlns="C:\TASXMLProject\Ebrahim"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="C:\TASXMLProject\Ebrahim SkillFunctionality.xsd">

<Operation name="Input">
        <Status>Applicant</Status>
        <Data>Used</Data>
</Operation>
<Operation name="Output">
        <Status>Student</Status>
        <Data>Produced</Data>
</Operation>
</SkillFunctionality>
```

Figure A-3: XML document for *TAS* skill-agent transition status.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2009 sp1 (http://www.altova.com) by Ebrahim Alhashel (University of Canberra) -->

<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
           xmlns:ns1="c:\TASXMLProject\goalTAS"
           targetNamespace="c:\TASXMLProject\goalTAS"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

<xs:element name="GoalDefinition">
      <xs:annotation>
        <xs:documentation> Standard goal definition for “TAS” </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Domain"/>
          <xs:element name="GoalId"/>
          <xs:element name="Name"/>
          <xs:element name="Constraint">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Boundery"/>
          <xs:element name="Subject"/>
          <xs:element name="Kind"/>
          <xs:element name="Item"/>
          <xs:element name="Limit">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="DateFrom" type="xs:date"/>
          <xs:element name="DateTo" type="xs:date"/>
        </xs:sequence>
        </xs:complexType>
        </xs:element>
        </xs:sequence>
        </xs:complexType>
        </xs:element>
</xs:schema>
```

Figure A-4: XML-Schema code for *TAS* goals following DMMAS goal structure.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2009 sp1 (http://www.altova.com) by Ebrahim Alhashel (University of Canberra) -->
<xs:schema   xmlns="c:\TASXMLProject\skillTAS"
             xmlns:xs="http://www.w3.org/2001/XMLSchema"
             targetNamespace="c:\TASXMLProject\skillTAS"
             elementFormDefault="qualified"
             attributeFormDefault="unqualified">

    <xs:element name="SkillTAS">
      <xs:annotation>
        <xs:documentation> skill-agent functionality structure for “TAS” </xs:documentation>
      </xs:annotation>
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Domain"/>
          <xs:element name="SkillId"/>
          <xs:element name="Name"/>
            <xs:element name="Status" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="InputOutput">
                 <xs:complexType>
                   <xs:attribute name="inputName"/>
                 </xs:complexType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
        </xs:element>

        <xs:element name="OperationType">
        <xs:complexType>
          <xs:sequence>
        <xs:element name="Function">
        <xs:complexType>
          <xs:sequence>
        <xs:element name="Input"/>
        <xs:element name="Output"/>
          </xs:sequence>
        <xs:attribute name="Status"/>
        </xs:complexType>
      </xs:element>
      </xs:sequence>
    </xs:complexType>
    </xs:element>
      </xs:sequence>
    </xs:complexType>
    </xs:element>
</xs:schema>
```

Figure A-5: XML-Schema code for *TAS* skill-agent functionality.

Appendix B: Ontology selected code for TAS skill-agents functionalities. Figure B-1 listed RDF/OWL ontology definition selected code for the TAS skill-agents "*FlightAgent*", "*HotelAgent*", *and* "*CarAgent*". Figure B-2 listed RDF/OWL ontology definition implementation code for the TAS skill-agent functionality "*CarRental*". The others ontology coding is not included in this appendix due to the large volume of pages.

```
?xml version="1.0" encoding="UTF-8"?>
<!-- Edited with Altova SemanticWorks V2010 De1 -->

<!-- Case Study: Travel Agency System (TAS) RDF/OWL Ontology -->
<!--Developed by: Ebrahim AlHashel  (Faculty of ISE - University of Canberra) -->

<rdf:RDF xmlns:carAgent=http://www.w3.altova.com/ontologies/CarAgent#
         xmlns:flightAgent=http://www.w3.altova.com/ontologies/FlightAgent#
         xmlns:hotelAgent=http://www.w3.altova.com/ontologies/HotelAgent#
         xmlns:owl="http://www.w3.org/2002/07/owl#"
         xmlns:rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns#
         xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
         xmlns:travel=http://www.w3.altova.com/ontologies/TravelAgencySystem#
         xmlns:xsd="http://www.w3.org/2001/XMLSchemma#">

<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
    <rdfs:subClassOf>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TAS"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </owl:disjointWith>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </owl:disjointWith>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
    <rdfs:subClassOf>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TAS"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </owl:disjointWith>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </owl:disjointWith>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
    <rdfs:subClassOf>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TAS"/>
    </rdfs:subClassOf>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </owl:disjointWith>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </owl:disjointWith>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TAS">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
```

```
    </rdf:type>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#agentName">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#hasAgent">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#Agent"/>
    </rdfs:range>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </rdfs:domain>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#Agent">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#hasCode">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </rdfs:domain>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#Agent"/>
    </rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#reserveCode">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#Agent"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2000/01/rdf-schema#Datatype"/>
    </rdfs:range>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </owl:disjointWith>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
```

```
    </owl:disjointWith>
    <owl:disjointWith>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </owl:disjointWith>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/FlightAgent#flightPreference">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#fromPlace">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#toPlace">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/FlightAgent#seatClass">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/FlightAgent#departDate">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#date"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/FlightAgent#ArriveDate">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#date"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/FlightAgent#price">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
```

```
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#FlightReservation"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#decimal"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/HotelAgent#hotelPreference">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/HotelAgent#roomClass">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/HotelAgent#inDate">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#dateTime"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/HotelAgent#outDate">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#dateTime"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/HotelAgent#hotelRate">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#HotelBooking"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#decimal"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/CarAgent#carPreference">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
    </rdf:type>
    <rdfs:domain>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
    </rdfs:domain>
    <rdfs:range>
        <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#TravelPreference"/>
    </rdfs:range>
</rdf:Description>
<rdf:Description rdf:about="http://www.w3.altova.com/ontologies/CarAgent#carType">
    <rdf:type>
```

```
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#string"/>
        </rdfs:range>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/CarAgent#rentDate">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#dateTime"/>
        </rdfs:range>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/CarAgent#endDate">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#dateTime"/>
        </rdfs:range>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/CarAgent#carRate">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.w3.altova.com/ontologies/TravelAgencySystem#CarRental"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.w3.org/2001/XMLSchema#decimal"/>
        </rdfs:range>
    </rdf:Description>
</rdf:RDF>
```

Figure B-1: RDF/OWL ontology code for the *TAS* skill-agents *FlightAgent, HotelAgent, and CarAgent* .

```
<?xml version="1.0"?>
<!-- Edited with Altova SemanticWorks V2010 De1 -->

<!-- Case Study: Travel Agency System (TAS) RDF/OWL Ontology -->
<!--Car Rent Skill-agent functionality definition -->
<!--Developed by: Ebrahim AlHashel  (Faculty of ISE - University of Canberra) -->

<rdf:RDF     xmlns:owl="http://www.w3.org/2002/07/owl#"
             xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
             xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
             xmlns:rent="http://www.carrent.com/ontologies/Rent#"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema#">

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Rent">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Car">
    <rdf:type>
        <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
    </rdf:type>
    <rdfs:subClassOf>
```

```
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Rent"/>
        </rdfs:subClassOf>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Type">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Brand">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Model">
        <rdf:type>
<rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Made">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Price">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#PickUp">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Return">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#Class"/>
        </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasType">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Car"/>
        </rdfs:domain>
</rdf:Description>

<rdf:Description rdf:about="file:///C:/MyXML/renthasModel">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Car"/>
        </rdfs:domain>
        <rdfs:range>
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Price"/>
        </rdfs:range>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasBrand">
        <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
        </rdf:type>
        <rdfs:domain>
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#Car"/>
        </rdfs:domain>
</rdf:Description>
```

```
<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasMade">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasPrice">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasPricePerDay">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#PricePerMonth">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#DatatypeProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasPickUp">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
      </rdf:type>
      <rdfs:domain>
            <rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#PickUp"/>
      </rdfs:domain>
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasReturn">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/2002/07/owl#ObjectProperty"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasReturnAddress">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasPickUpAddress">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasReturnDate">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
      </rdf:type>
</rdf:Description>

<rdf:Description rdf:about="http://www.carrent.com/ontologies/Rent#hasPickUpDate">
      <rdf:type>
            <rdf:Description rdf:about="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
      </rdf:type>
</rdf:Description>

</rdf:RDF>
```

Figure B-2: RDF/OWL ontology code for the *TAS* skill-agent functionality "*CarRental*".

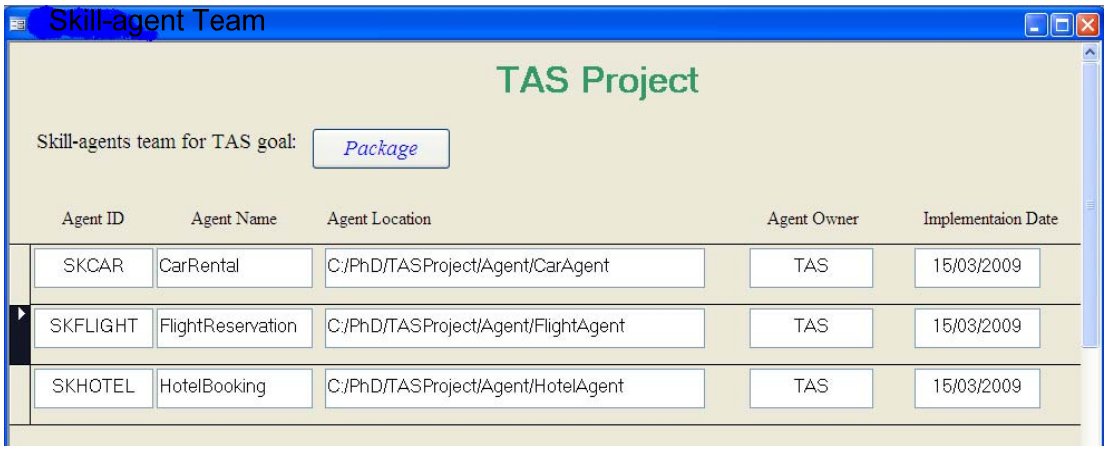

Figure B-3: Search result for *TAS* goal "*Package*" skill-agents team.

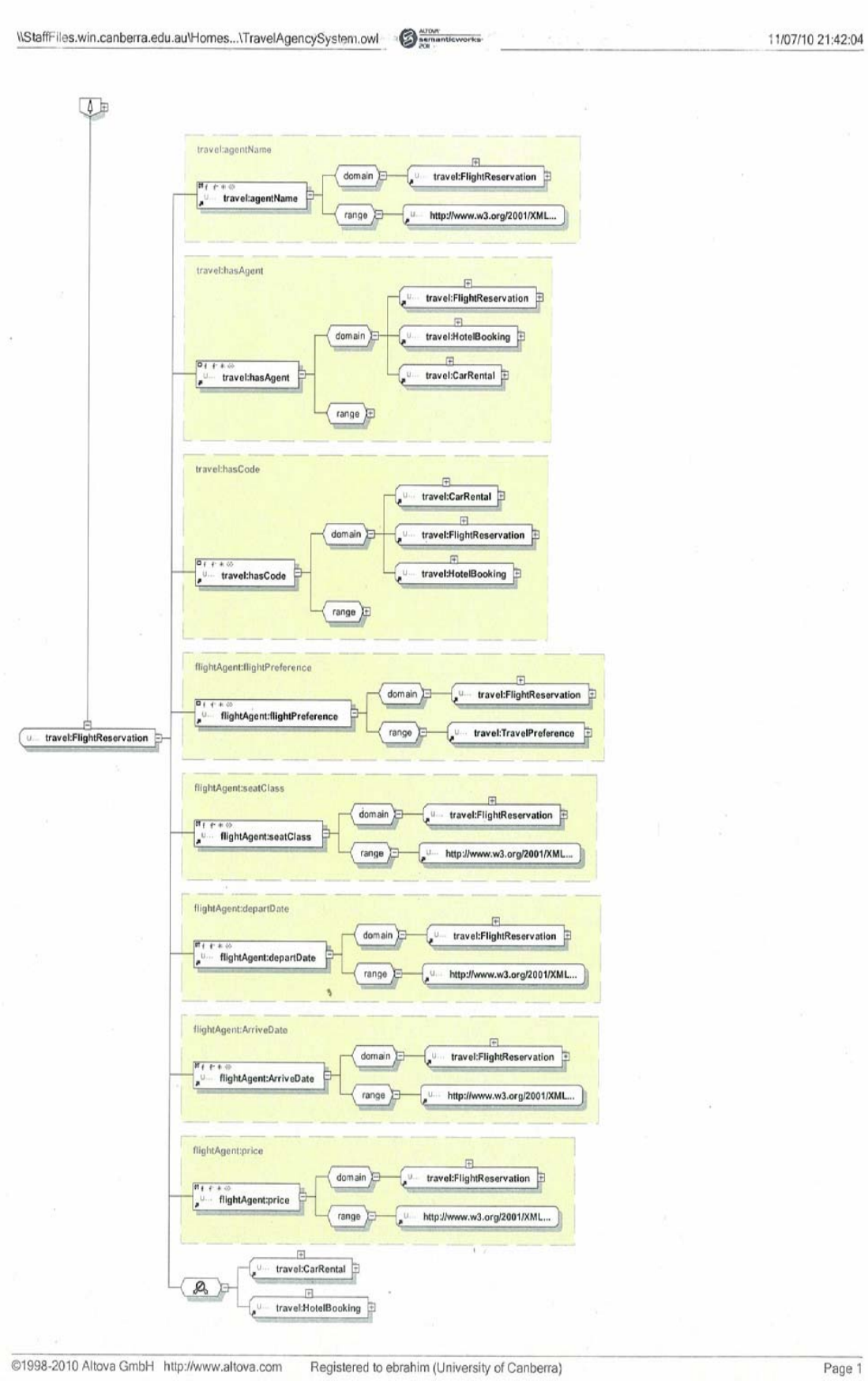

Figure B-4: Ontology concepts structure for TAS skill-agent "*FlightReservation*".

Appendix C: Goal execution plan database and SQL for "*AchivingPhD*" example.

Figure C-1 depicted the database design example for the goal execution plan. It illustrates the main table and attributes and their relationship. Each table is defined by its set of attributes integrated together in the system table which integrates the entire goal plan in one database schema. Figure C-2 illustrates the SQL statement set up to retrieve a record from the goal execution plan database. Figure C-3 illustrates a view screen for the goal plan database record for "*AchievingPhD*" example explained in Chapter 4.

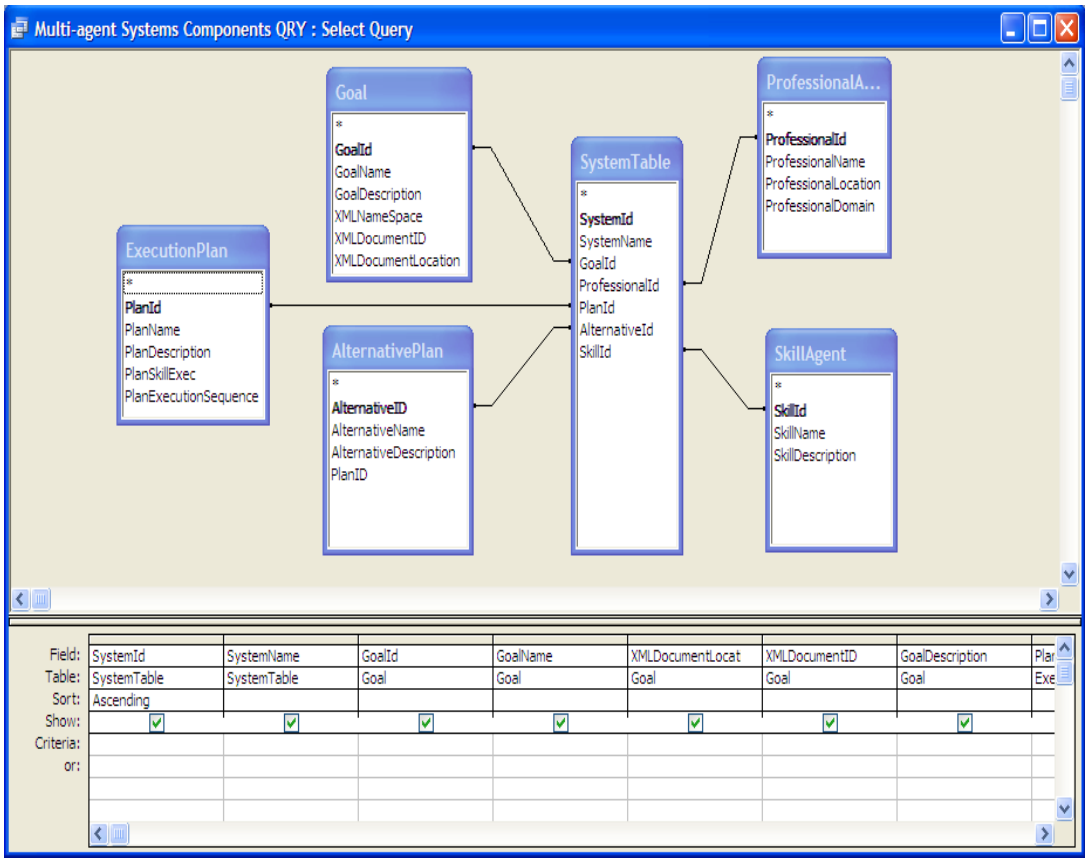

Figure C-1: Goal plan database.

```
SELECT     SystemTable.SystemId, SystemTable.SystemName,
           Goal.GoalId, Goal.GoalName, Goal.XMLDocumentLocation, Goal.XMLDocumentID,
           Goal.GoalDescription, ExecutionPlan.PlanId, ExecutionPlan.PlanName
           ExecutionPlan.PlanDescription,ProfessionalAgent.*,
           AlternativePlan.AlternativeID, AlternativePlan.AlternativeName,
           AlternativePlan.AlternativeDescription

      FROM          ExecutionPlan

       INNER JOIN   (SkillAgent
         INNER JOIN   (AlternativePlan
           INNER JOIN  (ProfessionalAgent
             INNER JOIN  (Goal
               INNER JOIN   SystemTable
               ON Goal.GoalId = SystemTable.GoalId)
             ON   ProfessionalAgent.ProfessionalId = SystemTable.ProfessionalId)
           ON   AlternativePlan.AlternativeID = SystemTable.AlternativeId)
         ON   SkillAgent.SkillId = SystemTable.SkillId)
       ON ExecutionPlan.PlanId = SystemTable.PlanId
       ORDER BY  SystemTable.SystemId;
```

Figure C-2: Goal plan SQL statement.

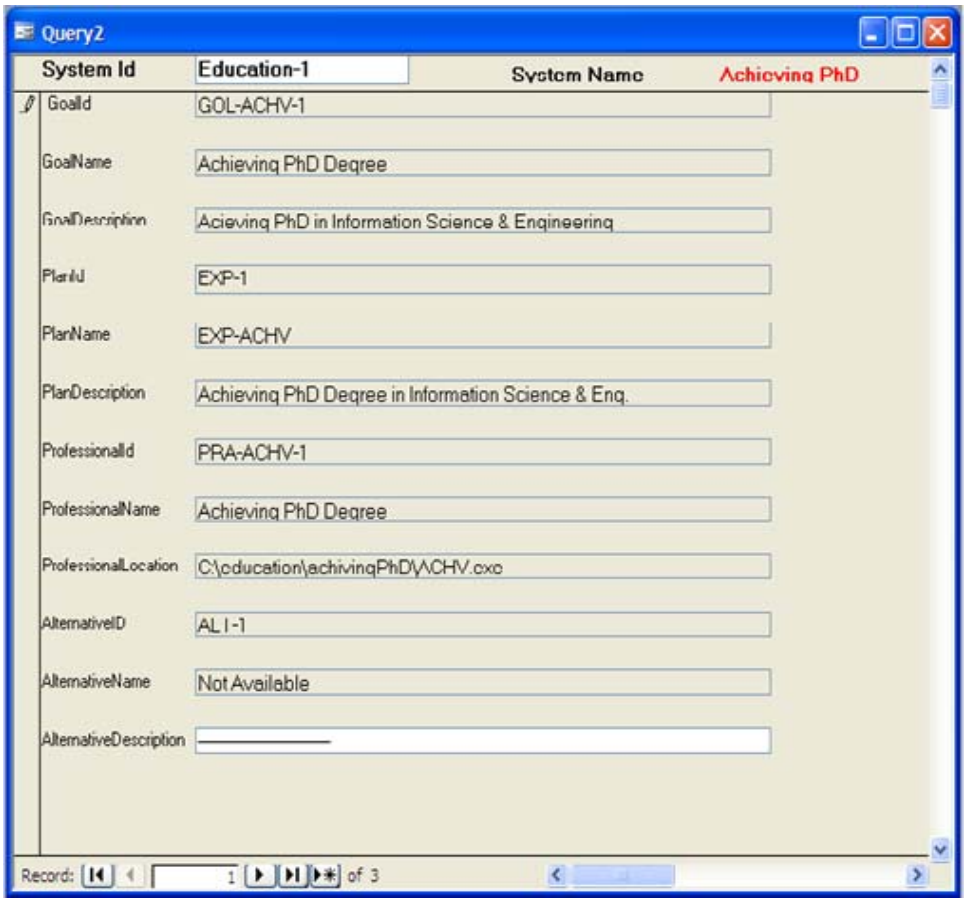

Figure C-3: Goal plan view example.

Appendix D: The AUML Interaction Diagram and Notation

This appendix demonstrates and explains the semantic of the agent unified modelling language (AUML) interaction diagram for multi-agent system communication and interaction protocols. The AUML interaction diagram is an extension of UML interaction diagram which by itself is a result of combination of both the activity diagram and the sequence diagram. Further details are available on http://www.auml.org.

Similar to UML, the AUML interaction diagram has lifelines and with time increasing as one move downwards. In addition, to represent the nature of an agent autonomous behaviour that has multiple thread of control the UML state diagram are also incorporated to denote the agent interaction specification (Odell et al., 2001). The other under lying idea behind the AUML interaction diagram is to represent a complete slot of an agent interaction in one diagram. For this reason the AUML interaction diagram has multiple regions in the form of nested boxes where each region (box) has its interaction functionality described in a label on the left top corner of each box. There are seven types of AUML interactions regions; sequence, optional, alternative, parallel, critical, loop, and reference.

Figure D-1 depicts an example for a basic interaction diagram where “sd” stands for sequence diagram follow by sequence diagram name. The rest is similar to the UML conventional model. The example shows the user agent sending a query message to the system and the system agent sending a response message to the user agent. The verticals lines or dashed lines represent the agent lifeline with the respect to the agent life cycle. If the agent is emergent and has a life limit, its lifeline is denoted by a dashed line. In contrast, if the agent is alive throughout the system operation then it can be denoted by a solid line.

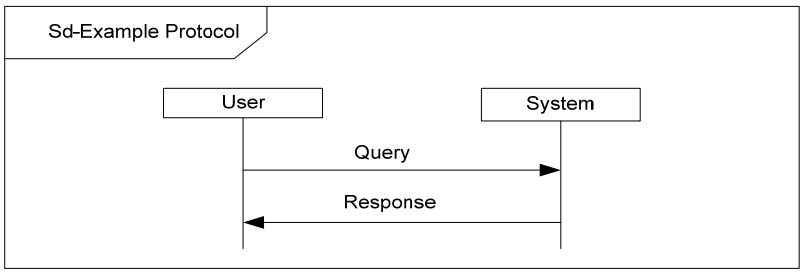

Figure D-1: AUML basic interaction diagram

Figure D-2 illustrates the agent interaction notation used in AUML and Figure D-3 depicts the deployment example. The agent interaction and the lifeline may split into two or more lifelines to show AND and OR parallelism and decisions, corresponding to branches in the message flow. Lifeline may merge at a particular instance of the interaction lifeline course. For example to handle “proposal” and “not understood” respectively the logical connectors (AND, XOR, and OR) are needed to handle such as these cases. The XOR can be abbreviated by interrupting the threads of interaction as shown in the example above. The thread of interaction is the processing of the income messages and it can be split up into different threads of interaction.

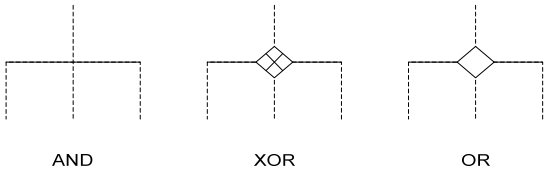

Figure D-2: Logical connector types

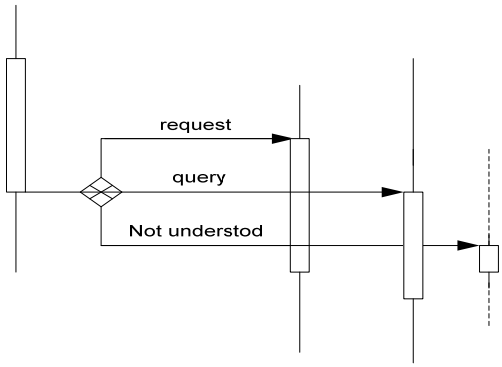

Figure D-3: Full and abbreviated notation of XOR connection.

The aim of the AUML interaction diagram is to represent a complete slot of the agent interactions in one diagram. For this reason the nested region applied. Figure D-4 depicts one complete AUML interaction diagram example described as following:

The **Loop** dictates that the region is repeated for some defined number of times depending on the range assigned, for example, Loop (1,3) means at least loop once for a maximum of three times.

The **Alternative** specifies that one of the interactions regions occurs. One of the regions may have "else" as guard.

The **Option** can only be executed if the guard is true; therefore it may or may not occur.

The **Parallel** specifies that both regions are executed at the same time slot.

The **Critical** region specifies that the messages encamps must occur sequentially and cannot intercepted by any other messages. It defines the condition of the critical action of the messages. For example, on the diagram shown above, the system shutdown message is the crucial action to take place by the system agent and the backup agent.

The **Reference** indicates to another interaction procedure defined outside the current context where it is defined.

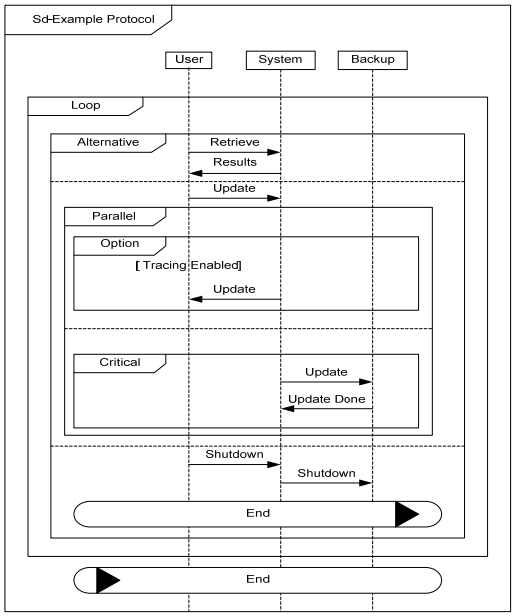

Figure D-4: AUML example showing notational elements.

Appendix E: List of Publications from Thesis Research

1. **Ebrahim Alhashel**, Bala Balachandran and Dharmendra Sharma, (2007). *A Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE*. Proceedings of Knowledge-Based Intelligent Information and Engineering Systems, Part III, (11th. KES-2007). Italy Vietri sul Mare, September 2007 pages: 909-916.

2. **Ebrahim Alhashel** (2008). *A Conceptual Agent Cooperation Model for Multi-agent Systems' Team Formation Process*. Proceedings Convergence and Hybrid Information Technology, Volume 1, (3rd. ICCIT-2008) South Korea, Busan, November 2008. pages: 12-20

3. **Ebrahim Alhashel**, and Masoud Mohammadian, (2008). *Illustration of a Multi-agent Systems Concept as User's Collaborative Software System*. Proceeding of International Conference on Intelligence for Modelling (CIMCA 2008) Austria, Vienna, December 2008. pages: 912-918

4. **Ebrahim Alhashel**, Bala Balachandran, and Dharmendra Sharma (2008). *Extending Prometheus with Agent Cooperation*. Proceeding of International Conference on Intelligence for Modelling (CIMCA 2008) Austria, Vienna, December 2008. pages:104-110

5. **Ebrahim Alhashel**, Masoud Mohammadian, Bala Balachandran, and Dharmendra Sharma (2009). *Architecture for an Agent-based Cooperative System*: Proceeding Computer Intelligent System and Agents (ISA 2009), Portugal, Algarve, June, 2009. pages: 219 - 240

6. Bala Balachandran, **Ebrahim Alhashel**, and Masoud Mohammadian (2009). *An Agent-Mediated Collaborative Negotiation in E-Commerce: A Case Study in Travel Industry*. Proceedings of Knowledge-Based Intelligent Information and Engineering Systems, Part II, (13th. KES 2009), Chile, Santiago, September, 2009, pages: 102-110.

7. **Ebrahim Alhashel**, Bala Balachandran and Dharmendra Sharma, (2009). *The Role of Ontology for Modelling Agent-based Systems*. Proceedings of Knowledge-Based Intelligent Information and Engineering Systems, Part II, (13th. KES-2009). Chile Santiago, September 2009 pages: 111-118.

8. **Ebrahim Alhashel** and Dharmendra Sharma, (2010). *Development Methodology for Cooperative, Distributed Multi-agent Systems*. An International Journal of Research and Innovation: Advances in Information Science and Service Science (AISS), (under review of December 2010).

Bibliography

ABRAN, A. & MOORE, J. W. (2004) Guide to the Software Engineering Body of Knowledge. IN BOURQUE, P. & DUPUIS, R. (Eds.), IEEE Computer Society. ISBN 0-7695-2330-7 Library of Congress Number 2005921729

ALESSANDRO, F. G., CARLOS, J. P. L., GRUPO, T., ANDREA, O., JAELSON, C., CENTRO DE, I. T. & FRANCO, Z. (2002) Software Engineering for Large-Scale Multi-Agent Systems.

ALESSIO, L., MICHAEL, W. & NICHOLAS, R. J. (2001) A Classification Scheme for Negotiation in Electronic Commerce. *Agent Mediated Electronic Commerce, The European AgentLink Perspective.*, Springer-Verlag.

ALHASHEL, E. (2008) A Conceptual Agent Cooperation Model for Multi-agent Systems' Team Formation Process. *Proceedings of the 2008 Third International Conference on Convergence and Hybrid Information Technology - Volume 01.* IEEE Computer Society.

ALHASHEL, E., BALA, B., MASOUD, M. & DHARMENDRA, S. (2009a) Architecture for Agent-based Cooperative Systems. IN REIS, A. P. D. (Ed.) *IADIS International Conference ISA 2009 (part of MCCSIS 2009).* Portugal - Algarve, IADIS Digital Library.

ALHASHEL, E., BALACHANDRAN, B. & SHARMA, D. (2007) Comparison of Three Agent-Oriented Software Development Methodologies: ROADMAP, Prometheus, and MaSE. *KES2007.* ITALY, Springer

ALHASHEL, E., BALACHANDRAN, B. & SHARMA, D. (2009b) The Role of Ontology in Agent-based Systems. *Intelligent Agent System.* Portugal - Algarve.

ALHASHEL, E., BALACHANDREN, B. & SHARMA, D. (2008) Enhancing Prometheus to Incorporate Agent Cooperation Process. *International Conference on Computational Intelligence for Modelling, Control and Automation - CIMCA.* Austria - Vienna, IEEE.

ALHASHEL, E. & MOHAMMADIAN, M. (2008) Illustration of Multi-agent Systems. *CIMCA.* Austria - Vienna, IEEE.

ANAL, A, A., ALEJANDRO, Z. & RAMIRO, I. (1999) Multi-paradigm Languages Supporting Multi-agent Development. *Proceedings of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World: MultiAgent System Engineering.* Springer-Verlag.

ANDREWS, D. (1996) Formal methods and software development.

AUSTIN, J. L. (1962) *How To Do Things With Words,* Cambridge, Massachusetts, Harvard University Press

AXEL VAN, L. (2000) Requirements engineering in the year 00: a research perspective. *Proceedings of the 22nd international conference on Software engineering.* Limerick, Ireland, ACM.

BACLAWSKI, K., KOKAR, M., KOGUT, P., HART, L., SMITH, J., LETKOWSKI, J. & EMERY, P. (2002) Extending the Unified Modeling Language for Ontology Development. *International Journal Software and Systems Modeling (SoSyM),* 1**,** 142-156.

BADR, I., MUBARAK, H. & GÖHNER, P. (2008) Extending the MaSE Methodology for the Development of Embedded Real-Time Systems. *Languages, Methodologies and Development Tools for Multi-Agent Systems.*

BAIRD, S. (2003) *Sams Teach Yourself Extreme Programming in 24 Hours,* The United State of America, Sams

BARBARA, J. G. (1996) Collaborative System. *American Association for Artificial Intelligence,* 86**,** 269-357.
BASS, L., CLEMENTS, P. & KAZMANKAZMAN, R. (2003) *Software Architecture in Practice*, Addison-Wesley Professional
BAUER, B. & ODELL, J. (2005a) UML 2.0 and Agents: How to Build Agent-Based Systems with the new UML Standard. *Engineering Applications of Artificial Intelligence,* 18.
BAUER, B. & ODELL, J. (2005b) UML 2.0 and Agents: How to Build Agent-Based Systems with the new UML Standard. *Engineering Applications of Artificial Intelligence,* 18**,** 141-157.
BBN (2004a) Cougaar Architecture Document. *BBN Technologies Document.*
BBN (2004b) Cougaar Development's Guide. *BNN Technologies Document.*
BELLIFEMINE, F., BERGENTI, F., CAIRE, G. & POGGI, A. (2005) Jade - A Java Agent Development Framework IN BORDINI, R. H., DASTANI, M., DIX, J. & SEGHROUCHNI, A. E. F. (Eds.) *Multi-Agent Programming.* Springer US.
BELLIFEMINE, F., CAIRE, G. & GREENWOOD, D. (2007a) *Developing Multi-agent Systems with JADE,* West Sussex England, Wiley
BELLIFEMINE, F., CAIRE, G. & GREENWOOD, D. (2007b) *Developing Multi-agent Systems with JADE,* West Sussex England, Wiley
BERGENTI, F. & HUHNS, M. N. (2004) On the Use of Agents as Components of Software Systems. *Methodologies and Software Engineering for Agent Systems.* Springer USA 19-31.
BERNON, C., GLIZE, P., PICARD, G. & GLIZE, P. (2002) ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering. IN PETTA, P. (Ed.) *Agent-Oriented Information Systems - ESAW 2002.* Madrid, Spain, Springer Berlin / Heidelberg.
BICHLER, M. (2000) A Roadmap to Auction-Based Negotiation Protocols for Electronic Commerce. *The 33rd Hawaii International Conference on System Sciences.* Hawaii-USA, IEEE Computer Society.
BLAKE, M. B. & GOMAA, H. (2005) Agent-oriented compositional approaches to services-based cross-organizational workflow. *Decision Support Systems,* 40**,** 31-50.
BOLCHINI, D. & PAOLINI, P. (2004) Goal-driven requirements analysis for hypermedia-intensive Web applications. *Requirements Engineering,* 9**,** 85-103.
BOUZOUBA, K., MOULIN, B. & KABBAJ, A. (2001) CG-KQML: an agent communication language and its use in a multi-agent systems. *The 9th Int. Conf. on Conceptual Structures.*
BRAUBACH, L., POKAHR, A. & LAMERSDORF, W. (2008) A Universal Criteria Catalogue for Evaluation of Heterogeneous Agent Development Artifacts. IN JUNG, B., MICHEL, F., RICCI, A. & PETTA, P. (Eds.) *From Agent Theory to Agent Implementation, 6th Int. Workshop, AAMAS 2008.* AT2AI-6 Working Notes ed. Estoril, Portugal, EU, AAI Journal.
BREITMAN, K. K., CASANOVA, M. A. & TRUSZKOWSKI, W. (2007) Methods for Ontology Development. *Semantic Web: Concepts, Technologies and Applications.*
BRESCIANI, P., GIORGINI, P., GIUNCHIGLIA, F., MYLOPOULOS, J. & PERINI, A. (2004) Tropos: An Agent-Oriented Software Development Methodology. *Autonomous Agents and Multi-Agent Systems.* Netherlands, Kluwer Academic

BULKA, B., GASTON, M. & DESJARDINS, M. (2007a) Local strategy learning in networked multi-agent team formation. *Autonomous Agents and Multi-Agent Systems,* 15**,** 29-45.

BULKA, B., GASTON, M., MARIE & DESJARDINS (2007b) Local Strategy Learning in Networked Multi-agent Team Formation. *Kluwer Academic Publishers,* 15**,** 29-45.

BUSSMANN, S. & MÜLLER, J. (1992) A Negotiation Framework for Cooperating Agents. IN S.M.DEEN (Ed.) *CKBS-SIG (CKBS'92).* DAKE Centre, Univ. of Keele.

CABAC, L. & MOLDT, D. (2005) Formal Semantics for AUML Agent Interaction Protocol Diagrams. *Agent-Oriented Software Engineering V.*

CAIRE, G., COULIER, W., GARIJO, F., GOMEZ, J., PAVON, J., LEAL, F., CHAINHO, P., KEARNEY, P., STARK, J., EVANS, R. & MASSONET, P. (2001) Agent-Oriented Analysis using Message/UML. IN M.J. WOOLDRIDGE, G. W., AND P. CIANCARINI (Ed.) *Agent-Oriented Software Engineering.* Springer Berlin / Heidelberg.

CASTELFRANCHI, C. (1995) Commitments: From Individual Intentions to Groups and Organizations. *1st. International Conference on Multi-Agent Systems (ICMA-95).* San Francisco, USA, Springer-Verlag 41-48.

CAVEDON, L. & TIDHAR, G. (1995) A logical framework for Multi-agent systems and joint attitudes. *Distributed Artificial Intelligence Workshop on Eighth Australian Joint Conference on Artificial Intelligence.* Canberra, Australia, Springer-Verlag

CERNUZZI, L., JUAN, T., STERLING, L. & ZAMBONELLI, F. (2004) The Gaia Methodology. *Methodologies and Software Engineering for Agent Systems.*

COHEN, P., LEVESQUE, H. & SMITH, I. (1997a) On Team Formation. IN HINITKKA, J. & TUOMELA, R. (Eds.) *Contemporary Action Theory, Syntheses.*

COHEN, P., LEVESQUE, H. & SMITH, I. (1997b) On Team Formation. IN HINITKKA, J. & TUOMELA, R. (Eds.) *Contemporary Action Theory, Synthesis.* .

COHEN, P. R. & LEVESQUE, H. (1991) Confirmations and Joint Action. *12th International Joint Conference on Artificial Intelligence 1991.* Sydney, Australia, 951-957.

CORCHO, O., FERNÁNDEZ-LÓPEZ, M. & GÓMEZ-PÉREZ, A. (2003) Methodologies, tools and languages for building ontologies. Where is their meeting point? *Data & Knowledge Engineering,* 46**,** 41-64.

D'INVERNO, M. & LUCK, M. (2004) *Understanding Agent System,* Berlin, Heidelberg, Springer-Verlag

DAM, K. H. & WINIKOFF, M. (2003) Comparing Agent-Oriented Methodologies. *Agent-Oriented Information Systems (AOIS-2003).* Melbourne, Springer.

DAM, K. H. & WINIKOFF, M. (2005) Comparing Agent-Oriented Methodologies.

DELOACH, S., WOOD, M. & SPARKMAN, C. (2001) Multiagent System Engineering. *International Journal of Software Engineering and Knowledge Engineering,* Vol. 11, No. 3.

DELOACH, S. A. (1999) Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems. IN WAGNER, G. & YU, E. (Eds.) *Agent Oriented Information Systems.* Seattle (USA).

DELOACH, S. A. & KUMAR, M. (2005) Multi-agent systems engineering: an overview and case study. IN HENDERSON-SELLERS, B. & GIORGINI, P. (Eds.) *Agent-Oriented Methodologies.* IDEA Group Publishing.
DEWAYNE, E. P. & ALEXANDER, L. W. (1992) Foundations for the study of software architecture. *SIGSOFT Software. Eng. Notes,* 17**,** 40-52.
DIGNUM, F., DIGNUM, V., THANGARAJAH, J., PADGHAM, L. & WINIKOFF, M. (2008) Open Agent Systems, *Agent-Oriented Software Engineering VIII.*
DORAN, J. E., FRANKLIN, S., JENNINGS, N. R. & NORMAN, T. J. (1996a) On Cooperation in Multi-Agent System. *Foundation of Multi-Agent Systems.* UK, University of Warwick.
DORAN, J. E., FRANKLIN, S., JENNINGS, N. R. & NORMAN, T. J. (1996b) On Cooperation in Multi-Agent Systems. *Foundations of Multi-Agent Systems.* University of Warwick UK.
DRAGAN, G., X, EVI, X, DRAGAN, D., X, VLADAN, D., X017E, I & X (2007) MDA-based Automatic OWL Ontology Development. *Int. J. Software. Tools Technol. Transf.,* 9**,** 103-117.
DURFEE, E. H., LESSER, V. R. & CORKILL, D. D. (1989) Trends in Cooperative Distributed Problem Solving. *IEEE Transactions on Knowledge and Data Engineering,* 1**,** 63-83.
EASTERBROOK, S., SINGER, J., STOREY, M.-A. & ABSTRACT, D. D. (2007) *Selecting Empirical Methods for Software Engineering Research*, Springer
ERIC, G. L. & GALINA, L. R. (2009) Designing ontologies for higher level fusion. Elsevier Science Publishers B. V.
ERIC, M. D., ANDR, VAN DER, H. & RICHARD, N. T. (2005) A comprehensive approach for the development of modular software architecture description languages. *ACM Trans Software Eng. Methodology,* 14**,** 199-245.
EYAL, A. & PEDRITO, M.-Z. (2004) Logic-based subsumption architecture. *Artif. Intelligence*, 153**,** 167-237.
FAN, X. & YEN, J. (2004) Modeling and Simulating Human Teamwork Behaviors Using Intelligent Agents. *Physic of Life Reviews.* Pennsylvania, USA, Elsevier B.V. 173-201.
FANKAM, C., JEAN, S., BELLATRECHE, L. & AÏT-AMEUR, Y. (2008) Extending the ANSI/SPARC Architecture Database with Explicit Data Semantics: An Ontology-Based Approach IN 2008, E. (Ed.) *ECSA 2008.* Springer Berlin / Heidelberg.
FARATIN, P., SIERRA, C. & JENNINGS, N. R. (2000) Using Similarity Criteria to Make Negotiation Trade-Offs. *International Conference on Multi-Agent Systems (ICMAS-2000).* Los Alamitos, CA, USA, IEEE Computer Society.
FASLI, M. (2007) *Agent Technology for e-Commerce*, John Wiley 453
FERRARI, L. (2004) The Aglets 2.0.2 User's Manual
FIPA (2000) Communicative Act Library Specification. FIPA TC C ed., Foundation for Intelligent Physical Agent, Switzerland, 2000 41.
FISHER, M., GHIDINI, C. & KAKOUDAKIS, A. (2006) Dynamic Team Formation in Executable Agent-Based Systems. *Agent Technology from a Formal Perspective.*
FRIEDMAN-HILL, E. (2003) *Jess in Action*, Manning Publication Co.
GARLAN, D. & SHAW, M. (1994) An Introduction to Software Architecture. Carnegie Mellon University.Garlan94SAIntroTR
GAŠEVIC, D., DJURIC, D. & DEVEDŽIC, V. (2009) The MDA-Based Ontology Infrastructure. *Model Driven Engineering and Ontology Development.*

GENESERETH, M. R. (1994) Software Agents. *Communication of the ACM,* 37**,** 48 - 147

GENESERETH, M. R., BOBROW, D., BRACHMAN, R., GRUBER, T., HAYES, P., LETDINGER, R., LIFSCHITZ, V., MACGREGOR, R., MCCARTHY, J., PATIL, R. & SCHUBERT, L. (1992) Knowledge Interchange Format. IN EFFORT, I. W. G. O. T. D. K. S. (Ed.) Stanford, California, Computer Science Department - Stanford University

GEORGEFF, M. P. (1987) Reactive Reasoning and planning. *7th. National Conference on Artificial Intelligence.* Seattle, Washington, USA, AAAI-Press.

GEORGEFF, M. P. & INGRAND, F. F. (1989) Decision-Making in an Embedded Reasoning System. *11th. International Joint Conference on Artificial Intelligence.* Detroit, Michigan, USA, Morgan Kaufmann.

GERD, W. & KULDAR, T. (2004) Towards Radical Agent-Oriented Software Engineering Processes Based on AOR Modeling. *Proceedings of the Intelligent Agent Technology, IEEE/WIC/ACM International Conference.* IEEE Computer Society.

GILB, T. (1985) Evolutionary Delivery versus the "waterfall model". ACM.

GILB, T. (1988) *Principles of Software Engineering Management, p. 91and 92*, Addison-Wesley

GIORGINI P., A. S. H. (2005) Agent-Oriented Methodologies: An Introduction. IN HENDERSON-SELLERS, B. & GIORGINI, P. (Eds.) *Agent-Oriented Methodologies.* Idea Group Inc. (IGI). 413.

GOMEZ-PEREZ, A., FERNANDEZ-LOPEZ, M. & CORCHO, O. (2004) *Ontological Engineering with examples from the areas of Knowledge Management, e-Commerce and the Semantic Web*, Springer

GÓMEZ-PÉREZ, A. & MANZANO-MACHO, D. (2003) Survey of ontology learning methods and techniques. IST Project IST-2000-29243 OntoWeb

GONZALEZ-PALACIOS, J. & LUCK, M. (2008) Extending Gaia with Agent Design and Iterative Development. *Agent-Oriented Software Engineering VIII.*

GRADY, B., ROBERT, M., MICHAEL, E., BOBBI, Y., JIM, C. & KELLI, H. (2007) *Object-oriented analysis and design with applications, third edition*, Addison-Wesley Professional

GRAU, B., HORROCKS, I., MOTIK, B., PARSIA, B., PATELSCHNEIDER, P. & SATTLER, U. (2008) OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web,* 6**,** 309-322.

GREER, D. & BUSTARD, D. W. (1996) Towards an Evolutionary Software Delivery Strategy based on Soft Systems and Risk Analysis. *IEEE Symposium and Workshop on Engineering of Computer Based Systems (ECBS'96).* IEEE.

GROSZ, B. J. & KRAUS, S. (1996) Collaborative Plans for Complex Group Action. *Artificial Intelligence,* 86**,** 269-357.

GROSZ, B. J. & SIDNER, C. L. (1990) Plans for Discourse. IN COHHEN, P., MORGAN, P. R., L., J. & POLLACK, M. E. (Eds.) *Intentions and Communication.* Cambridge, MA: MIT Press 417-444.

GRUBER, T. (1992) Ontolingua: A mechanism to support portable ontologies. Stanford University, Knowledge Systems Laboratory

GRUBER, T. R. (1993) A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*.

GRUNINGER, M. & FOX, M. S. (1995) Methodology for the Design and Evaluation of Ontologies. *IJCAI'95, Workshop on Basic Ontological Issues in Knowledge Sharing, April 13, 1995.* citeULike-article-id: 1273832.

HARTONAS, C. (2003) Towards Team-Orientation in Agent Design: Proof-Checking Plan Integrity, Social Structure and Privacy of Intra-Team Communication. Greece, Technological Education Institute of Larissa
HEDERSON-SELLERS, B. & GIORGINI, P. (2005) *Agent-Oriented Methodologies*, Idea Group Publishing
HELENA SOFIA, P., JO & O, P. M. (2001) A methodology for ontology integration. *Proceedings of the 1st international conference on Knowledge capture.* Victoria, British Columbia, Canada, ACM.
HESSE, W. (2005) Ontologies in the Software Engineering process. IN AL., R. L. E. (Ed.) *Tagungsband Workshop on Enterprise Application Integration, 2005 and:* *http://sunsite.informatik.rwth-aachen.de/* GITO-Verilog Berlin, CEUR-WS.
HEVNER, A. R., MARCH, S. T., PARK, J. & RAM, S. (2004) Design science in information systems research. *MIS Quarterly,* 1.
HOEK, W. V. D., JAMROGA, W. & WOOLDRIDGE, M. (2007) Toward a Theory of Intention Revision. *Springer Science+Business Media B.V.,* 155.
HOLT, A. (1988) Diplans - a new language for the study of co-ordination. *ACM transactions on Office Information Systems* 109-125.
Aglet Development Toolkit. url: http://aglets.sourceforge.net/. Access date: 23/07/2008
JACK Development Toolkit. url: http://www.agent-software.com/. Access date: 07/07/2008
Agent UML. url: Access date: 29/10/2009
OWL Web Ontology Language. url: http://www.w3.org/TR/owl-features/. Access date: 01/06/2009
IGLESIAS, C. A., GARIJO, M. & GONZALEZ, J. C. (1999) A Survey of Agent-Oriented Methodologies. IN MULLER, J. P. (Ed.) *ATAL'98.* Germany, Springer-Verlag 317-330.
ISIK, M., STULP, F., MAYER, G. & UTZ, H. (2007) Coordination without Negotiation in Teams of Heterogeneous Robots. IN (EDS.), G. L. E. A. (Ed.) *RoboCup.* Berlin-Germany, Springer-Verlag.
IYAD, R., SARVAPALI, D. R., NICHOLAS, R. J., PETER, M., SIMON, P. & LIZ, S. (2003) Argumentation-based negotiation. *Knowl. Eng. Rev.,* 18**,** 343-375.
JACOBSON, I., BOOCH, G. & RUMBAUGH, J. (2000) *The Unified Software Development Process*, Addison Wesley Longman Inc.
JEFFREY SOLOMON, R. (1986) Rational interaction: cooperation among intelligent agents. Stanford University.
JEN-HSIANG, C., RACHID, A., KUO-MING, C. & NICK, G. (2002) Architecture of an Agent-Based Negotiation Mechanism. *Proceedings of the 22nd International Conference on Distributed Computing Systems.* IEEE Computer Society.
JENNINGS, N. F., JENNINGS, N. R., NORMAN, T. J., FARATIN, P. & ODGERS, B. (2000) Autonomous Agents for Business Process Management. *Int. Journal of Applied Artificial Intelligence,* 14**,** 145–189.
JENNINGS, N. R. (1996) Coordination Techniques for Distributed Artificial Intelligence. IN G. M. P. O'HARE, N. R. J. (Ed.) *Foundations of Distributed Artificial Intelligence.* New York, John Wiley & Sons Inc.
JENNINGS, N. R., FARATIN, P., LOMUSCIO, A. R., PARSONS, S., WOOLDRIDGE, M. J. & SIERRA, C. (2004) Automated Negotiation:

Prospects, Methods and Challenges. *Group Decision and Negotiation,* 10**,** 199-215.

JIM, B., TOM, M., MATTHIAS, Z., AWAIS, R. & GÜNTER, K. (2005) Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice,* 17**,** 309-332.

JOHN, D. (2005) *Object-Oriented Analysis and Design*, Addison Wesley

JOHN, E. G., WILLIAM, T. S. & WILLIAM, F. G. (2007) Six Major Phases of Systems Analysis. *How to Do Systems Analysis.*

JONES, T. C. (1986) *Systematic Software Development Using VDM,* London, Prentice-Hall

JONKER, C. M. & TREUR, J. (2001) An Agent Architecture for Multi-Attribute Negotiation. *the 17th International Joint Conference on AI,* Morgan Kaufman.

KATIA, P. S. (1989) Multiagent compromise via negotiation. *Distributed Artificial Intelligence (Vol. 2).* Morgan Kaufmann Publishers Inc.

KELLER, U., LARA, R., POLLERES, A., TOMA, I., KIFER, M. & FENSEL, D. (2004) WSMO Web Service Discovery. IN KELLER, U., LARA, R. & POLLERES, A. (Eds.) *Frameworks for Semantics in Web Services,* Web Service Modeling Ontology

KIM, I.-C. (2006) An Agent-Oriented Approach to Semantic Web Services. *Next Generation Information Technologies and Systems.*

KINNY, D., GEORGEF, M. & RAO, A. (1996) A Methodology and Modelling Technique for System of BDI Agents. IN VELDE, W. V. D. & PERRAM, J. (Eds.) *Modelling Autonomous Agents in a Multi-Agent World MAAMAW'96.* Germany, Springer-Verlag

KINNY, D., LJUNGBERG, M., RAO, A., TIDHAR, G., WERNER, E. & SONENBERG, E. (1992) Planed Team Activity. *4th. Workshop on Modelling Autonomous Agents in a Multi-Agent World MAAMAW '92.* Rome, Italy, Springer-Verlag 225-256.

KLUSCH, C. H. S. N. S. W. I. Z. K. F. M. (2008) Integration of Multiagent Systems and Semantic Web Services on a Platform Independent Level. *International Conference on Web Intelligence and Intelligent Agent Technology.* 2008, IEEE/WIC/ACM

KONE, M. T., SHIMAZU, A. & NAKAJIMA, T. (2000) The State of the Art in Agent Communication Languages. *Knowledge Information Systems,* 2**,** 259-284.

KREIFELTS, T. & VON MARTIAL, F. (1990) A negotiation framework for autonomous agents. *In Proceedings of the Second European Workshop on Modeling Autonomous Agents and Multi-Agent Worlds.* North-Holland.

LABROU, Y. & FININ, T. W. (1998) Semantics and Conversations for an Agent Communication Language. *CoRR,* cs.MA/9809034.

LAMSWEERDE, A. V. (2001) Goal-Oriented Requirements Engineering: A Guided Tour. *5th. IEEE International Symposium on Requirements Engineering.* Toronto Canada, IEEE.

LANGLEY, P., LAIRD, J. E., & ROGERS, S. (2008) Cognitive architectures: Research issues and challenge. *Cognitive Systems Research.* Elsevier

LAWLEY, R., LUCK, M., DECKER, K., PAYNE, T. & MOREAU, L. (2004) Automated Negotiation Between Publishers and Consumers of Grid Notifications. *UK e-Science All Hands Meeting 2004* United Kingdom, World Scientific Publishing Company.

LESZCZYNA, R. (2008) Evaluation of Agent Platforms (ver. 2.0). Luxembourg, European Commission 2008. JRC47224
LEVESQUE, H. J., COHEN, P. R. & NUNES, J. (1990) On Acting Together. *AAAI-90.* Morgan Kaufman Publishers, Inc., 94-99.
LI CHANGHONG, L. M., KOU JISONG (2002) Cooperation Structure of Multi-Agent and Algorithms. *IEEE International Conference on Artificial Intelligence Systems (ICAIS'02).* China, IEEE.
LIN, P. (2005) Tool Support for Agent Development using the Prometheus Methodology. IN JOHN, T. & MICHAEL, W. (Eds.).
url: Loom, URL: http://sevak.isi.edu:4676/loom/shuttle.html. Access date: 29/06/2009
LUCK, M. (1999) From Definition to Deployment: What Next for Agent-Based System. *Knowledge Engineering Review,* 14(2).
LUCK, M., ASHRI, R. & D'INVERNO, M. (2004) *Agent-Based Software Development,* London, Artec House Inc.
LUCK, M. & D'INVERNO, M. (1996) Engagement and Cooperation in Motivated Agent Modelling. IN ZHANG, C. & LUKOSE, D. (Eds.) *Distributed Artificial Intelligence Architecture and Modelling.* Australia, Springer - Verlag 70-84.
MALLEY, S. A. O. & DELOACH, S. A. (2001) Determining When to Use an Agent-Oriented Software Engineering Paradigm. *In Proceedings of the Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001.*
MALONE, T. & CROWSTON, K. (1990) What is coordination theory and how can it help design cooperative work systems? *CSCW '90: Proceedings of the 1990 ACM conference on Computer-supported cooperative work.* ACM Press.
MARC-PHILIPPE, H. (2003) Extending Agent UML Sequence Diagrams. *Agent-Oriented Software Engineering III.*
MARC-PHILIPPE, H. & ODELL, J. (2005) Representing Agent Interaction Protocols with Agent UML. *5th. International Workshop AOSE.*
MARCO, C. & MARIO, V. (2002) An analysis of agent speech acts as institutional actions. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 3.* Bologna, Italy, ACM.
MARK, R. A., ALVAH, B. D., ROBERT, W. & RALPH, W. W. (1989) Conflict resolution strategies for non-hierarchical distributed agents. *Distributed Artificial Intelligence (Vol. 2).* Morgan Kaufmann Publishers Inc.
MATSUBAYASHI, K. & TOKORO, M. (1993) A Collaboration Mechanism on Positive Interactions in Multi-Agent Environments. IN SPRINGER-VERLAG (Ed.) *International Joint Conference on Artificial Intelligence.* Tokyo, Japan, 346-351.
MATTEO, C. & ROBERTA, C. (2005) A Survey on Ontology Creation Methodologies. *International Journal on Semantic Web & Information Systems,* 1 49 - 69.
MATTHEW, E. G. & MARIE, D. (2005) Agent-organized networks for dynamic team formation. *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems.* The Netherlands, ACM.
MATTHEW, E. G. & MARIE, D. (2008) THE EFFECT OF NETWORK STRUCTURE ON DYNAMIC TEAM FORMATION IN MULTI-AGENT SYSTEMS. *Computational Intelligence,* 24**,** 122-157.
MCBREEN, P. (1998) Using Use Cases for Requirements Capturing.

MICHAEL, J. W. (1999) Software Engineering with Agents: Pitfalls and Pratfalls. IN NICHOLAS, R. J. (Ed.).
MORSE, B. J. M. & FIELD, P. A. (1995) *Qualitative research methods for health professionals*, SAGE
MULLAR, J. P., WOOLDRIDGE, M. & JENNINGS, N. R. (1997) *Intelligent Agent Theories, Architectures and Languages*, Springer 401
NARINDER, S. & MICHAEL, G. (1991) Epikit: a library of subroutines supporting declarative representations and reasoning. ACM.
NICHOLAS, R. J., KATIA, S. & MICHAEL, W. (1998) A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems,* 1**,** 7-38.
NOY, N. F. & MCGUINNESS, D. L. (2001) Ontology development 101: A guide to creating your first ontology. Stanford Medical Informatics. Technical Report SMI-2001-0880
NWANA, H. S. (1996) Software Agents: An Overview. *Knowledge Engineering Review,* 11**,** 1-40.
NWANA, H. S., LEE, L. & JENNINGS, N. R. (1996) *Coordination in software agent systems*. *British Telecommunication Technology Journal* 14**,** 79-88.
NYUNT, P. P. & THEIN, N. L. (2005) Software Agent Oriented Information Integration System in Semantic Web. *6th Asia-Pacific Symposium on Information and Telecommunication Technologies.* Yangon.
ODELL, J. (2002) Objects and Agents Compared. *Journal of Object Technology,* 1**,** 41-54.
ODELL, J., PARUNAK, H. V. & BAUER, B. (2003) Extending UML for Agents. *Agent-Oriented Software Engineering* Melbourne, Springer-Verlag
ODELL, J., VAN DYKE PARUNAK, H. & BAUER, B. (2001) Representing Agent Interaction Protocols in UML. *Agent-Oriented Software Engineering.*
OGATA, N. & COLLIER, N. (2004) Ontology express: Statistical and non-monotonic learning of domain ontologies from text. *of the Workshop on Ontology Learning and Population held at the 16th European Conference on Artificial Intelligence (ECAI'2004).* Valencia, Spain.
ONN, S. & ARNON, S. (2001) Evaluation of modeling techniques for agent-based systems. *Proceedings of the fifth international conference on Autonomous agents.* Montreal, Quebec, Canada, ACM.
PADGHAM, L. & MICHAEL, W. (2004) *Developing Intelligent Agent Systems,* England, John Wiley & Sons, Ltd 23 82
PAL'CHUNOV, D. E. (2005) Logical Definition of Object Domain Ontology. *9th. Asian Logic Conference.* Novosibirsk, Russia, World Scientific.
PAVÓN, J., GÓMEZ-SANZ, J. J. & FUENTES-FERNÁNDEZ, R. (2005) The INGENIAS Methodology and Tools. IN HENDERSON-SELLERS, B. & GIORGINI, P. (Eds.) *Agent-Oriented Methodologies.* Idea Group Publishing.
PEI PEI, K., SHANIKA, R. & LEON, S. (2005) Improving Goal and Role Oriented Analysis for Agent Based Systems. *Proceedings of the 2005 Australian conference on Software Engineering.* IEEE Computer Society.
POKAHR, A. & LAMERSDORF, W. (2008) A Universal Criteria catalogue for Evaluation of Heterogeneous Agent Development Artifacts. IN PETTA, B. J. A. F. M. A. A. R. A. P. (Ed.) *6th Int. Workshop, May 13, 2008, AAMAS 2008.* Estoril, Portugal, EU.Lars Braubach, BraubachPokahrLamersdorf2008.
POLLACK, M. (1986) A Model of Plan Inference that Distinguishes Between the Beliefs of Actors and Observers. *New York 24th. Annual Meeting on*

Association for Computational Linguistics New York, Association for Computational Linguistics 207-214 207-214
POLLACK, M. (1990) Plans as Complex Mental Attitude. IN R, C. P. & L, M. J. (Eds.) *Intentions in Communication.* Cambridge MIT Press 77-103.
RABINOVICH, A. Z., POCHTER, A. N. & ROSENSCHEIN, A. J. S. (2008) Coordination and Multi-Tasking Using EMT. IN FOX, D. & GOMES, C. P. (Eds.) *AAAI.* Chicago, Illinois, USA, AAAI Press.
RAO, A. S. (1996) AgentSpeak (L): BDI Agents speak out in a logical computable language. IN VELDE, W. V. D. & PERRAM, J. W. (Eds.) *7th. European Workshop on Modelling Autonomous Agents in a Multi-Agent World.* Heidelberg, Germany Springer-Verlag, Heidelberg, Germany, 42-55.
RAQUEL, T. (2007) Comparison and Performance Evaluation of Mobile Agent Platforms. IN SERGIO, I. & EDUARDO, M. (Eds.) Athens, Greece IEEE.
ROBINSON, D. J. (2000) A Component Based Approach To Agent Specification. *SCHOOL OF ENGINEERING, AIR FORCE INSTITUTE OF TECHNOLOGY (AU),* WRIGHT-PATTERSON AIR FORCE BASE.
RODNEY, A. B. (1986) A Robust layered Control System for Mobile Robot. *IEEE Journal of Robotics and Automation,* 2**,** 14-23.
RODNEY, B. A. (1991) Intelligence without representation. *Artif. Intelligence,* 47**,** 139-159.
RUDOWSKY, I. (2004) Intelligent Agent. *Americans Conference on Information Systems.* New York.
SEARLE, J. R. (1969) *Speech Acts. An Essay in the Philosophy of Language*, Cambridge
SELLERS, B. H. & GORTON, I. (2003) Agent-based Software Development Methodologies. *Workshop on Agent-Oriented Methodologies.* Seattle, USA, Springer-Verlag
SIERRA, C., JENNINGS, N., NORIEGA, P. & PARSONS, S. (1998) A Framework for Argumentation-Based Negotiation. *Lecture Notes in Artificial Intelligence.* Springer-Verlag.
SIMON, H. A. (1996) *The Sciences of the Artificial*, MIT Press, Cambridge, MA
SIRICHAROEN, V. W. (2007) Ontologies and Object models in Object Oriented Software Engineering *International MultiConference of Engineers and Computer Scientists 2007.* Hong Kong, 21-23 March, 2007, IAENG
SOMMERVILLE, I. (1992) *SOFTWARE ENGINEERING*, ADDISON WESLEY
SPIVEY, J. M. (1989) *The Z Notation: A Reference Manual,* London, Prentice-Hall
STANTON, N., SALMON, P. M., BADER, C. & JENKINS, D. P. (2005) *Human Factors Methodology: A Practical Guide for Engineering and Design*, Ashgate Publication Limited
STUM, A. & SHEHORY, O. (2004) A Framework for Evaluating Agent-Oriented Methodologies. Volume 3030/2004.
SUDEIKAT, J., BRAUBACH, L., POKAHR, A. & LAMERSDORF, W. (2005) Evaluation of Agent–Oriented Software Methodologies – Examination of the Gap Between Modeling and Platform. IN ODELL, J. (Ed.) *5th International Workshop, AOSE 2004.* New York, NY, USA, Springer-Verlag Berlin Heidelberg 2005.
SUSAN, E. C., ROBERT, A. M. & VICTOR, R. L. (1988) Multistage negotiation in distributed planning. *Distributed Artificial Intelligence.* Morgan Kaufmann Publishers Inc.

SYCARA, K., GIAMPAPA, J. A., LANGLEY, B. & PAOLUCCI, M. (2003) The RETSINA MAS, a Case Study. IN GARCIA, A., LUCENA, C., ZAMBONELLI, F., OMICI, A. & CASTRO, J. (Eds.) *Software Engineering for LargeScale Multi-Agent Systems.* Springer-Verlag 232-250.

SYCARA, K. P. (1998) Multiagent Systems. *American Association for Artificial Intelligence*.

SYCARA, K. P. & SUKTHANKAR, G. (2006) Literature Review of Teamwork Models, Pittsburgh, Pennsylvania, Robotics Institute, Carnegie Mellon University, 31CMU-RI-TR-06-50

TAMBE, M. (1997) Teamwork in Real-World, Dynamic Environments. *2nd.International Conference on Multi-Agent Systems (ICMAS)*

TELLER, J., LEE, J. & ROUSSEY, C. (Eds.) (2007) *Ontologies for Urban Development,* Geneva, Springer-Verlag Berlin.

THALHEIM, B. (2000) *Entity-Relationship Modeling: Foundations of Database Technology* Springer-Verlag, Berlin, Germany

THOMAS, J., ADRIAN, P. & LEON, S. (2002) ROADMAP: extending the gaia methodology for complex open systems. *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1.* Bologna, Italy, ACM.

TIM, F., RICHARD, F., DON, M. & ROBIN, M. (1994) KQML as an agent communication language. *Proceedings of the third international conference on Information and knowledge management.* Gaithersburg, Maryland, United States, ACM.

TIMOTHY, J. N. & CHRIS, R. (2001) Delegation and Responsibility. *Proceedings of the 7th International Workshop on Intelligent Agents VII. Agent Theories Architectures and Languages.* Springer-Verlag.

TUOMELA, R. (2001) Collective Intentionality and Social Agents. *Artificial Intelligent IMF.* Toulous, France.

TVEIT, A. (2001) A survey of Agent-Oriented Software Engineering. *First NTNU CSGSC.* Springer-Verlag

VAUGHAN, J., CONNELL, R., LUCAS, A. & RONNQUIST, R. (2003) Toward Complex Team Behaviour in Multi-agent Systems Using a Commercial Agent Platform. IN TRUSZKOWSKI, W., ROUFF, C. & HINCHEY, M. (Eds.) *WRAC.* Berlin Heidelberg, Springer-Verlag 175-185.

VIEIRA, R., MOREIRA, A., WOOLDRIDGE, M. & BORDINI, R. H. (2007) On the Formal Semantics of Speech-Act Based Communication in an Agent-Oriented Programming Language. *Artificial Intelligence Research*.

VLADAN, D. (2002) Understanding ontological engineering. *Commun. ACM,* 45**,** 136-144.

VRBA, P. (2004) JAVA-Based Agent Platform Evaluation. *Holonic and Multi-Agent Systems for Manufacturing.* Springer Berlin / Heidelberg

WERKMAN, K. J. (1990) Knowledge-based model of negotiation using shareable perspectives. *The 10th International Workshop on Distributed Artificial Intelligence.* MCC Technical Report.

WEYNS, D., OMICINI, A. & ODELL, J. (2007) Environment as a First Class Abstraction in Multiagent Systems. *Autonomous Agents and Multiagent Systems.* MA, USA, Kluwer Academic Publishers Hingham, 5-30.

WILSKER, B. (1996) A Study of Multi-Agent Collaboration Theories. Pasadena, University of Southern California. ISI/RR-96-449

WOOD, M. F. & DELOACH, S. A. (2001) An Overview of the Multi-agent System Engineering Methodology. IN CANCARINI, P. & WOOLDRIDGE, M. (Eds.) *First International Workshop on Agent-Oriented Software Engineering.* Limerick Ireland, Springer Verlag, Berlin.

WOOLDRIDGE, M. & DUNNE, P. E. (2006) On the computational complexity of coalition resource game. *Artificial Intelligence,* 170.

WOOLDRIDGE, M., JENNINGS, N. & KINNY, D. (2000) The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems.* Netherland, Kluwer Academic Publishers.

WOOLDRIDGE, M. & JENNINGS, N. R. (1995) Intelligent Agents: Theory and Practice. *Knowledge Engineering Review,* 10

XIAOMENG, S. & LARS, I. (2002) A Comparative Study of Ontology Languages and Tools. *Proceedings of the 14th International Conference on Advanced Information Systems Engineering.* Springer-Verlag.

YOGESH, S., ANJANA, G. & MANOJ, K. (2008) Evaluation of Agent Oriented Requirements Engineering Frameworks. *Proceedings of the 2008 International Conference on Computer Science and Software Engineering - Volume 02.* IEEE Computer Society.

ZAMBONELLI, F., JENNINGS, N. & WOLDRIDGE, M. (2003) Developing Multiagent Systems: The Gaia Methodology. ACM