MESSAGE: Methodology for Engineering Systems of Software Agents

Methodology for Agent-Oriented Software Engineering

Authors:
Richard Evans (editor), Broadcom Eireann Research Ltd.
Paul Kearney, Jamie Stark, BT
Giovanni Caire, TI
Francisco J. Garijo, Jorge J. Gomez Sanz, Juan Pavon, TE
Francisco Leal, Paulo Chainho, PT Inovação
Philippe Massonet, CEDITI

Abstract
This document presents the MESSAGE methodology for Agent Oriented Software Engineering. Agent technology is gaining more and more importance in the domain of telecommunication applications (e.g. EURESCOM P712, P815, P704, P810, P845; ACTS projects KIMSAC, AMASE, ABROSE, CLIMATE). The concepts and technology have been brought to a stage where they are useable in real applications, and there is a growing understanding of how to apply them to practical problems. To exploit this investment, the knowledge of where and how to employ agent-oriented ideas must become part of the repertoire of skills of the EURESCOM partners’ software engineering teams.

Software engineering (SE) methodologies have proved to be successful in increasing speed to market of software development projects, lowering the development cost and providing better quality. It is possible for small teams to apply SE principles informally, but for large projects a well-defined methodology is essential. Message is an attempt to combine all of the best features of existing agent oriented software engineering (AOSE) methodologies while grounding agent-oriented concepts in the same underlying semantic framework used by UML, the standard Modelling language for Object Oriented Software Engineering.

EDIN 0223-0907
Project P907
For full publication September 2001
Disclaimer

This document contains material which is the copyright of certain EURESCOM PARTICIPANTS, and may not be reproduced or copied without permission.

All PARTICIPANTS have agreed to full publication of this document.

The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the PARTICIPANTS nor EURESCOM warrant that the information contained in the report is capable of use, or that use of the information is free from risk, and accept no liability for loss or damage suffered by any person using this information.
Preface

(Prepared by the EURESCOM Permanent Staff)

Agents are one of the main topics in the area of modern telecommunication applications. But without a reasonable methodology for the development of agent based systems, professional development of these applications is informal, cumbersome, error prone, and thus, expensive. Tools are required for the support of this methodology, because they simplify and formalise the development process. The more general case of the methodology can be focused in the EURESCOM context on the specific domain of telematic services.

The focus of the project work is the extension of existing methodologies for object-oriented software development to agent-oriented applications as well as the identification of tools, which would support this methodology. This should encompass the whole software life cycle, analysis, design, implementation, testing, installation, and reiteration. The methodology should be centred around the agent oriented realisation of telematics services and telecommunications applications. The work should make use of experiences made in other projects including EURESCOM projects like P712 and P815, and should take into account work going on in FIPA, OMG and TINA-C.

This is a Technical Information (deliverable 3 out of 4 deliverables) on the Methodology for Agent-oriented Software Engineering of the project.

The Project started its activities on June 1999 and finished in June 2001. The Project was led by Win Coulier (RB), and from January 2001 by Philippe Massonet (CEDITI). BT, IT, PT, RB, TI, and FT were participating in the Project.

Heinz Brüggemann

Project Supervisor
Executive Summary

Agent technology is gaining more and more importance in the domain of telecommunication applications (e.g. EURESCOM P712, P815, P704, P810, P845; ACTS projects KIMSAC, AMASE, ABROSE, CLIMATE). Agents are one of the main topics in the area of modern telecommunication applications [Eur99]. But without a reasonable software engineering methodology for the development of agent based systems, professional development of these applications is informal, cumbersome, error prone, and thus, expensive.

Software engineering (SE) methodologies have proved to be successful in increasing speed to market of software development projects, lowering the development cost and providing better quality. These benefits are greater for large projects, than smaller ones.

The agent-oriented approach is well suited to developing complex and distributed systems [Jen01, Gri01]. Complex software has a large number of parts that have many interactions, and solve problems that are in general only nearly decomposable. Agent-Oriented Software Engineering (AOSE) aims to provide methods, techniques and tools to solve these problems.

It is not appropriate to use standard OO methodologies for developing agent-oriented (AO) applications. The reason is that the concept of ‘agent’ is different from that of ‘object’, and the difference is significant at a high level of abstraction. This is not to say that OO methodologies should be ignored, however. Rather, the aim of MESSAGE is to extend existing methodologies to allow them to support AOSE.

To this end, EURESCOM project P907 has defined the MESSAGE methodology (Methodology for Engineering systems of software agents) for agent-oriented analysis and design. The MESSAGE methodology consists of applicability guidelines, a modelling notation, and a process for analysis and design of agent systems. The MESSAGE modelling notation extends the UML notation with agent centric concepts. The RUP [Kru00] has been adopted as a coarse-grained iterative process model, and MESSAGE specific analysis and design processes expressed in terms of the MESSAGE modelling language.

The methodology has been assessed by exercising it on two case studies, i.e. UPA4T (Universal Personal Assistant for Travel) - a task-specific instantiation of a generic personal assistant application, and ACSOSS (Adaptive Customer Service OSS) - a decentralised OSS application performing end-to-end co-ordination within a customer service business process. In addition feedback was obtained from work within the LEAP EU 5th Framework Project [Lea01], where aspects of the MESSAGE methodology were applied to an application supporting a mobile workforce. Feedback from these case studies has been used to improve the methodology.

MESSAGE as it stands is not a complete, mature agent-oriented methodology. It does, however, make some significant practical contributions to the state of the art [Gar01, Cai01] that are likely to influence on-going initiatives in this area, e.g. Agent UML [Bar99, Ode00]. In particular, the graphical notation / diagram set, which extend UML class and activity diagrams, is a practical concrete result that could be taken up widely. Furthermore, even in its current state MESSAGE provides practical value to the developer of Multi-Agent Systems.
Table of contents

Preface .................................................................................................................................................. 3
Executive Summary .............................................................................................................................. 3
Table of contents ................................................................................................................................. 3
List of Figures ...................................................................................................................................... 4
List of Tables ....................................................................................................................................... 7
Abbreviations ..................................................................................................................................... 8

1 Introduction ...................................................................................................................................... 10
1.1 Software Engineering Methodologies .......................................................................................... 10
1.2 Agent Oriented Software Engineering ......................................................................................... 11
1.3 Comparison to Other Approaches .............................................................................................. 12
1.4 Structure of document ................................................................................................................ 12

2 The MESSAGE Modelling Language ............................................................................................ 14
2.1 Main MESSAGE Concepts .......................................................................................................... 14
2.1.1 Foundations .......................................................................................................................... 14
2.1.2 Knowledge-level concepts ..................................................................................................... 14
2.1.2.1 Concrete Entities .............................................................................................................. 15
2.1.2.2 Activities ......................................................................................................................... 15
2.1.2.3 Mental State Entities ....................................................................................................... 16
2.2 Analysis Model Views ................................................................................................................ 16
2.2.1 Definition of Views ............................................................................................................... 16
2.2.2 Consistency Between the Views ............................................................................................ 17

3 Analysis Process ............................................................................................................................ 18
3.1 Introduction .................................................................................................................................. 18
3.2 Inputs to the Modelling Process ................................................................................................ 19
3.3 Overview of Modelling Process ................................................................................................ 19
3.3.1 Structural analysis, behavioural synthesis and dynamic behaviour .................................... 21

4 Analysis Diagrams and Case Study ............................................................................................... 23
4.1 Case Study Description ................................................................................................................ 23
4.2 Level 0 Analysis ........................................................................................................................... 23
4.2.1 Organisation view .................................................................................................................. 23
4.2.2 Goal/Task view ...................................................................................................................... 24
4.3 Level 1 Analysis .......................................................................................................................... 26
4.3.1 Organisation view .................................................................................................................. 26
4.3.2 Agent/Role view .................................................................................................................... 26
4.3.3 Interaction view ...................................................................................................................... 27
4.3.4 Domain view .......................................................................................................................... 28

5 Design process and case study ....................................................................................................... 29
5.1 MAS organisation and architecture driven design ................................................................. 29
5.1.1 Design activities .................................................................................................................. 30
5.1.1.1 Refining the analysis entities ......................................................................................... 31
5.1.1.2 Selecting the Agent Architecture ................................................................................... 31
5.1.1.3 Structuring the results according to the Organisation view ........................................... 33
5.1.1.4 Fulfilling the Architecture ............................................................................................. 34
5.1.1.5 Identifying components for external behaviour ............................................................. 35
5.1.1.6 Fulfilling the domain layer ............................................................................................ 36
5.1.1.7 Fulfilling the Decision and Management Layer ............................................................ 37
5.1.2 Design Guidelines ................................................................................................................ 40
5.2 Platform specific design process issues .................................................................................... 42
5.2.1 Target platform issues .......................................................................................................... 42
5.2.2 Description of Low Level Design Process ........................................................................... 43
5.2.3 Interaction Protocol Design .................................................................................................. 46

6 Evaluation of the methodology .................................................................................................... 47
6.1 Towards a Complete and Mature AOSE Methodology ......................................................... 47
6.1.1 Analysis ............................................................................................................................... 47
6.1.2 Design .................................................................................................................................. 47
6.1.3 Implementation ..................................................................................................................... 48
6.1.4 Current Status of AOSE Methodologies ............................................................................. 48
6.2 MESSAGE Evaluation ............................................................................................................... 48

Conclusions ....................................................................................................................................... 52
# Annex 3: Detailed discussion of the analysis process

<table>
<thead>
<tr>
<th>Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.1 Summary</td>
</tr>
<tr>
<td>7.2 Recommendations on the Methodology</td>
</tr>
<tr>
<td>7.3 Open Issues</td>
</tr>
<tr>
<td>7.4 Proposals for Further Work</td>
</tr>
<tr>
<td>Annex 1: MESSAGE Notations at-a-glance</td>
</tr>
<tr>
<td>Annex 2: A-UML Interaction Diagrams</td>
</tr>
<tr>
<td>Annex 3: Detailed discussion of the analysis process</td>
</tr>
</tbody>
</table>

## A3.1 Building the Level 0 Model

- **A3.1.1 Organisation View**
- **A3.1.2 Start accumulating the Domain Model/View**
- **A3.1.3 Identification of services and workflows**
- **A3.1.4 Identification of purpose**
- **A3.1.5 System details**
  - **A3.1.5.1 Goal analysis**
  - **A3.1.5.2 Task knowledge**
  - **A3.1.5.3 Interaction details**
- **A3.1.6 Check for completeness and consistency**

## A3.2 Building the Level 1 Model

- **A3.2.1 Goals and Tasks**
- **A3.2.2 Identification of Agents, Interactions and Relationships (Organisational View)**
  - **A3.2.2.1 Modified Use Case Analysis**
  - **A3.2.2.2 Reconciliation**
  - **A3.2.2.3 Introduce co-ordination/control mechanisms**
  - **A3.2.2.4 Develop domain model**
- **A3.2.6 Detailed descriptions**
- **A3.2.7 Check consistency**
- **A3.2.8 Synthesis of behaviour**
List of Figures

Figure 1 The RUP lifecycle model........................................................................................................... 11
Figure 2. MESSENGE Concepts................................................................................................................ 15
Figure 3 Analysis model, views, diagrams and schemas. ............................................................................ 17
Figure 4. General Workflow schematic for MESSAGE Analysis activity .................................................. 20
Figure 5. Structural analysis, behavioural synthesis and dynamic behaviour ........................................... 21
Figure 6. Level 0 Organisation Diagram (Structural relationships) ............................................................ 24
Figure 7. Level 0 Organisation Diagram (Acquaintance relationships) ......................................................... 24
Figure 8. Level 0 Goal/Task Implication Diagram ....................................................................................... 25
Figure 9. Level 0 Workflow diagram ........................................................................................................ 25
Figure 10. Level 1 Organisation Diagram (Acquaintance relationships) ....................................................... 26
Figure 11. Level 1 Delegation Structure Diagram ........................................................................................ 27
Figure 12. Interaction Diagram ................................................................................................................ 28
Figure 13. Domain Information Diagram .................................................................................................. 28
Figure 14. Main activities .......................................................................................................................... 30
Figure 15. Layer architecture for the agent .................................................................................................. 32
Figure 16. Travel Assistance Agency Organisation View in the analysis ..................................................... 33
Figure 17. System architecture in the design based on the analysis Organisation View ................................. 34
Figure 18. Resource Layer for Agent UPA Flight Information & Notification Assistant ............................ 36
Figure 19. Flight Manager and Travel Managers ....................................................................................... 37
Figure 20. Relationships among the different types of knowledge .............................................................. 38
Figure 21. Allow User Define Flight Notification Goal decomposition with tasks and evidences ............... 39
Figure 22. Graphical notation to represent evidence gathering and success management for goal ............. 40
    AllowUserDefineFlightNotificationG
Figure 23. Instantiation of a social rule to express rejection of requests in case subordination does not exist.. 40
Figure 24 Overview of design process (lower box) .................................................................................. 43
Figure 25. Example message including content ......................................................................................... 45
Figure 26. Example Agent Class diagram, and Agent Instance acquaintance object diagram .................... 45
Figure 27. Inheritance diagram for Behaviours and Behaviour instance containment object diagram ........... 45
Figure 28. FIPA protocol diagram ........................................................................................................... 46
Figure 29. The RUP: Phases, activities and iterations ................................................................................... 52
Figure 30 MESSAGE concepts symbols ................................................................................................... 57
Figure 31 MESSAGE relations symbols ................................................................................................... 57
Figure 32 Example of a “Goal-Task Implication diagram ........................................................................... 58
Figure 33. Usage of implication to show goal decomposition .................................................................. 58
Figure 34. Assignment relation subtypes .................................................................................................. 59
Figure 35. Examples of different Assignment relations in MESSAGE notation .......................................... 59
Figure 36. Usage of UML dependency to represent “consumption” ............................................................ 59
Figure 37. DataFlow as an association class .............................................................................................. 60
Figure 38. Different outputs ....................................................................................................................... 60
Figure 39 Information flow between non-consecutive tasks .................................................................... 61
Figure 40. Example of Organisation diagram ............................................................................................ 61
Figure 41. Example of Delegation structure diagram ............................................................................... 62
Figure 42. Example of Workflow diagram .................................................................................................. 62
Figure 43. Example of Agent diagram .................................................................................................... 63
Figure 44. Example of Interaction diagram ................................................................................................ 63
Figure 45 FIPA Contract Net Interaction Protocol ....................................................................................... 64
Figure 46 .................................................................................................................................................. 65
Figure 47 .................................................................................................................................................. 70
List of Tables

Table 1. Agent/Role Schema for Travel Arrangement Selector.................................................................27
Table 2. Translation from analysis to computational entities................................................................30
Table 3. Identified sessions .........................................................................................................................36
Table 4. Example of Task execution management rule (reduced version)..................................................40
### Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACL</td>
<td>Agent communication language</td>
</tr>
<tr>
<td>ACSOSS</td>
<td>Adaptive Customer Service Operations and Support System</td>
</tr>
<tr>
<td>AI</td>
<td>Artificial Intelligence</td>
</tr>
<tr>
<td>AO</td>
<td>Agent-oriented</td>
</tr>
<tr>
<td>AOSE</td>
<td>Agent-oriented software engineering</td>
</tr>
<tr>
<td>A-UML</td>
<td>Agent UML</td>
</tr>
<tr>
<td>AV</td>
<td>Agent View</td>
</tr>
<tr>
<td>BDI</td>
<td>Belief, Desire, Intention</td>
</tr>
<tr>
<td>CLIPS</td>
<td>C’ Language Integrated Production System</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>DL</td>
<td>Domain Layer</td>
</tr>
<tr>
<td>DML</td>
<td>Decision and Management Layer</td>
</tr>
<tr>
<td>DV</td>
<td>Domain View</td>
</tr>
<tr>
<td>FIPA</td>
<td>Foundation for Intelligent Physical Agents</td>
</tr>
<tr>
<td>GTV</td>
<td>Goal/Task View (of the MESSAGE Analysis Model)</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IV</td>
<td>Interaction View</td>
</tr>
<tr>
<td>JESS</td>
<td>Java Expert System Shell</td>
</tr>
<tr>
<td>KADS</td>
<td>Knowledge and Analysis Design Support</td>
</tr>
<tr>
<td>KIF</td>
<td>Knowledge Interchange Format</td>
</tr>
<tr>
<td>KM</td>
<td>Knowledge Management</td>
</tr>
<tr>
<td>KQML</td>
<td>Knowledge Query and Manipulation Language</td>
</tr>
<tr>
<td>LTSA</td>
<td>Labelled Transition System Analyser</td>
</tr>
<tr>
<td>MAS</td>
<td>Multi-Agent System</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Methodology for Engineering Systems of Software Agents</td>
</tr>
<tr>
<td>MOF</td>
<td>Meta-Object Facility</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group</td>
</tr>
<tr>
<td>OO</td>
<td>Object-oriented</td>
</tr>
<tr>
<td>OV</td>
<td>Organisation View</td>
</tr>
<tr>
<td>PCL</td>
<td>Perception and Communication layer</td>
</tr>
<tr>
<td>RL</td>
<td>Resource layer</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>SE</td>
<td>Software engineering</td>
</tr>
<tr>
<td>SL</td>
<td>FIPA Semantic Language</td>
</tr>
<tr>
<td>SL0</td>
<td>FIPA Semantic Language Profile 0</td>
</tr>
<tr>
<td>SL1</td>
<td>FIPA Semantic Language Profile 1</td>
</tr>
<tr>
<td>SL2</td>
<td>FIPA Semantic Language Profile 2</td>
</tr>
<tr>
<td>TA</td>
<td>Travel Arrangement</td>
</tr>
<tr>
<td>TINA</td>
<td>Telecommunications Information Networking Architecture</td>
</tr>
<tr>
<td>TSP</td>
<td>Travelling Service Provider</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language – an object-oriented modelling language used in several OO methodologies</td>
</tr>
<tr>
<td>UPA</td>
<td>Universal Personal Assistant</td>
</tr>
<tr>
<td>UPA4T</td>
<td>Universal Personal Assistant for Travelling</td>
</tr>
</tbody>
</table>
1 Introduction

This document is the Deliverable 3 of the MESSAGE project (EURESCOM P907). It presents an Agent Oriented Software Engineering methodology, developed in particular for the needs of the telecommunications industry. The agent-oriented (AO) approach promises the ability to construct flexible systems with complex and sophisticated behaviour by combining highly modular components. The intelligence of these components – the agents – and their capacity for social interaction results in a Multi-Agent System (MAS) with capabilities beyond those of a simple ‘sum’ of the agents. The EURESCOM partners (and other companies and universities) have invested considerable manpower and money in agent Research and Development. The concepts and technology have been brought to a stage where they are useable in real applications, and there is a growing understanding of how to apply them to practical problems. To exploit this investment, the knowledge of where and how to employ agent-oriented ideas must become part of the repertoire of skills of the EURESCOM partners’ software engineering teams. Software engineering (SE) methodologies have proved to be successful in increasing speed to market of software development projects, lowering the development cost and providing better quality. It is possible for small teams to apply SE principles informally, but for large projects a well-defined methodology is essential. Most methodologies currently in use or being introduced are object-oriented (OO).

MAS are distributed software systems consequently it might be supposed that existing SE methodologies, and specifically OO methodologies, might be applied for building MAS. However, it is not appropriate to use standard OO methodologies for developing agent-oriented (AO) applications. The reason is that the concept of ‘agent’ is different from that of ‘object’, and the difference is significant at a high level of abstraction. This is not to say that OO methodologies should be ignored, however. Rather, the aim of MESSAGE is to extend existing methodologies to allow them to support Agent-Oriented Software Engineering (AOSE).

MESSAGE is not alone in working towards an AOSE methodology. Three projects that have particularly influenced MESSAGE are MAS-CommonKADS [Igl97], Gaia [Woo00] and Agent UML [Bau99]. MESSAGE aims to combine the best features of these approaches. It is a genuinely agent-oriented methodology, but also builds upon the achievements of software engineering, and is consistent with current SE best practice. Furthermore, MESSAGE grounds agent-oriented concepts in the same underlying semantic framework used by UML, and uses UML-based notation whenever appropriate.

1.1 Software Engineering Methodologies

A software engineering methodology provides methods, guidelines, descriptions, and tools for each for each phase in the life cycle of a system, so as to ensure the production and maintenance of a well-engineered product that is fit for its purpose. Life cycle models in system engineering aim to describe the main stages between the birth of the system and its death. Stages may be structured into phases, which are periods of time where specific activities are carried out. Different actors such as users, customers, owners, managers and developers may perform activities in each phase. The process model defines a set of activities, the sequence in which activities are carried out, and the relationship between activities. Each activity results in one or more deliverables such as specifications, documents, models, or code. Models are defined in a notation that is appropriate for a given deliverable. Examples of notations are Use Case Models, Class Diagrams, and Interaction Diagrams, State Transition Diagrams. All of these notations form part of the Universal Modelling Language (UML) which supports Object Oriented (OO) development. Tools are typically used to edit models in a given notation.

The approach for defining the MESSAGE agent life cycle model and the supporting methodology was to consider as a baseline well known and widely accepted software approaches. Two important choices have been done concerning notations and process model.

The lifecycle model of the Unified Process for software development proposed by Rational [http://www.rational.com/products/rupt/index.jsp, Kru00], was considered the most suitable for MESSAGE. Reasons for these choices are
• It covers all the principal life cycle stages of software development
• It can be applied to different application areas and different application sizes in different organisations
• It takes into account the incremental and evolutionary nature of software development

The Rational Unified Process (RUP) model provides a generic software engineering project lifecycle framework. A project proceeds through a number of phases (Inception, Construction, Elaboration, and Transition). Each of these phases consists of one or more iterations of a sequence of activities: Requirements, Analysis, Design, Implementation and Testing.

The Unified Modelling Language (UML) [Omg99] was selected for two complementary reasons:
• UML is widely accepted as a de facto standard for object-oriented modelling, many software engineers are trained in its use, and commercial software tools are available to support it (some of which are extendable).
• UML is based on a meta-model (UML uses the MOF meta-modelling language [Omg99b]) which makes it extendable [Omg99].

The MESSAGE modelling language is related to UML as follows:
1. It shares a common meta-modelling language (meta-metamodel) with UML and MOF
2. It extends the UML meta-model with ‘knowledge level’ agent-oriented concepts.

A more complete description of the relationship between the MESSAGE meta-model and the UML meta-model is given in [Mes01].

1.2 Agent Oriented Software Engineering

It has been claimed [Jen99] that "the most powerful abstractions are those that minimise the semantic gap between the units of analysis that are intuitively used to conceptualize the problem and the constructs present in the solution paradigm". The OO paradigm has been so successful because it has allowed analysts, designers, customers and users to think of the software as analogous to the physical objects they are familiar with in the real world, and which also feature in the problem domain. Just as in OOSE the fundamental unit of abstraction is the object, in AOSE the primary unit of abstraction is the agent. Objects in OOSE are based on an analogy with inanimate objects in the real world. Similarly, agents are based on an analogy with animate objects.
(typically human beings). An agent-based approach is used when software is required to have human-like attributes, for example apparent intelligence, social ability, goal-directed decision-making, etc. In such cases, the AO paradigm narrows the semantic gap mentioned by Jennings and Wooldridge. Note also that most software systems are embedded in a larger context, for example a business operation. The concepts used in AO analysis (see Section 2.1 below) are very similar to those used to model human organisations. Thus there is a natural continuity in going from (business) requirements modelling to analysis.

Many agent-oriented design patterns are based on simplified models of the human mind and human social structures (organisations, markets, contract nets, etc.). This is because the attributes that are sought: intelligence, adaptability, learning, etc. are inherently difficult to produce. It makes sense, therefore, to model the design on entities (people, animals, organisations, and societies) that display them, even though the mechanisms involved are poorly understood. These patterns apply mainly to the high-level aspects of design: the ‘social’ architecture of the MAS, and the internal structural and behavioural features of the agents. Detailed design, e.g. of mechanisms implementing particular reasoning algorithms, can employ conventional software engineering techniques. Furthermore, agent toolkits are available that include implementations of standard algorithms. If one of these is used (and if the developer is very lucky) the need for design at this very low level may be limited.

In consequence, the MESSAGE methodology focuses on analysis and design activities. It provides a set of Analysis and Design models for Agent systems. It also describes how to go about building these Analysis and Design models. Other stages of the life cycle such as implementation, testing and deployment have not been considered due to the limitations of time and resources of the project. However design models let the door open for investigating the support needed in these phases. Some of this support will be in the form of guidelines whereas others will be the ability to generate automatically sources code, testing procedures and deployment procedures. MAS also raise some in-service software engineering issues, such as the need for run-time monitoring and detection of behavioural anomalies. These are important, but beyond the scope of MESSAGE.

1.3 Comparison to Other Approaches

Work toward an AOSE methodology can be divided into two broad categories. The first category aims to apply existing software engineering methodologies to AOSE. Agent-UML (A-UML) [Bau99] for example defines extensions to UML with notations suited for agent concepts. A-UML has extended UML’s interaction diagrams to handle agent interaction protocols. Although this notation is useful and has been adopted within MESSAGE, it does not have the concept of agent at its centre, i.e. specifying an object’s behaviour in terms of interaction protocols does not make it an agent. The second category of work aims at developing a methodology from agent theory, mainly covering analysis and design. Typically these methodologies define a number of models for both analysis and design [Igl98], notably Gaia [Woo00] and MAS-CommonKADS [Igl97]. The Gaia methodology has two analysis models and three design models. While the analysis models are based on well-defined concepts, these only represent a subset of the concepts required for agent oriented analysis. The design models are not clearly explained and the authors envisage OO methods being used for detailed design. MAS-Common-KADS has six models for analysis, and three for design. While these models are comprehensive, the method lacks a unifying semantic framework and notation. In addition to this work, goal analysis techniques have been shown to be very useful [Myl01], [Dar93]. The techniques range from informal to formal analysis and cover functional and non-functional goal analysis. MESSAGE combines the best features of the above approaches.

1.4 Structure of document

This document begins with Chapter one, which includes an introduction to Agent Oriented Software engineering, and a comparison to other approaches. Chapter two describes the MESSAGE meta-model, which is the set of underlying concepts, which form the basis of MESSAGE modelling constructs. Chapter three examines the Analysis process and model. The process is illustrated by a simple case study. The models from the case study show the different views within the Analysis model. Chapter four describes the process of designing agent systems as
illustrated by two different examples, one using an platform specific approach and one not making use of any agent platform. Chapter five gives a short evaluation of the methodology. Finally conclusions are presented in chapter six. A number of appendices are also presented including an overview of the MESSAGE modelling notations.
2 The MESSAGE Modelling Language

Section 2.1 introduces the main concepts of the MESSAGE modelling language. It starts by defining the notion of state, which is required to understand the relationships between states. Then the main concepts and relationships of the modelling language are described. Section 2.2 describes five views that have been defined on the analysis model to help the analyst structure the model.

2.1 Main MESSAGE Concepts

2.1.1 Foundations

MESSAGE takes UML as a starting point and adds entity and relationship concepts required for agent-oriented modelling. Agent-oriented modelling borrows from the study of human organisations and societies in describing the way in which agents in a Multi-Agent System work together to achieve a collective purpose, and from artificial intelligence (AI) and cognitive psychology to describe the agents themselves. These additional concepts can be defined in terms of object-oriented ones, but deal with ideas and structures at a higher conceptual level. In AI this higher level is often referred to as 'the knowledge level', contrasting knowledge with data. Essentially, MESSAGE uses standard UML as its 'data level' modelling language, but provides additional 'knowledge level' concepts. These additional concepts are defined in the MESSAGE meta-model [Mes01]. The meta-model also gives a declarative interpretation to some UML concepts used to describe behaviour. The most significant of these is “State” which is described hereafter.

The MESSAGE interpretation of State can be described as follows. A UML model is a collection of objects. A full description of this model at a point in time consists of a description of the value of every attribute of every object. Let us call such description a micro-state. It is rarely practical or useful to work directly with micro-states, however. A State is characterised by a partial description of the model, i.e. a constraint restricting the micro-state of model to being one of a set of possible micro-states. The simplest form of constraint would be to give the value of one attribute of one object in the model. Because States are sets (of micro-states), the language of Boolean algebra can be used to describe their relationships (set union, intersection and containment are equivalent to logical or, and and implication).

Note that this is entirely consistent with the UML State concept. From the UML 1.3 specification [Omg99]:

A state is an abstract metaclass that models a situation during which some (usually implicit) invariant condition holds. The invariant may represent a static situation such as an object waiting for some external event to occur. However, it can also model dynamic conditions such as the process of performing some activity (i.e., the model element under consideration enters the state when the activity commences and leaves it as soon as the activity is completed).

The rest of this section describes the knowledge level concepts that feature most prominently in the MESSAGE methodology as it stands at the moment, particularly those that appear explicitly in diagrams.

2.1.2 Knowledge-level concepts

Most of the MESSAGE knowledge level entity concepts can be grouped into the following main categories: ConcreteEntity, Activity, and MentalStateEntity. Figure 2 gives an informal agent-centric overview of how these concepts are inter-related.

---

1 Note that earlier versions of the meta-model used the term Situation instead of State, with state used as a synonym for micro-state. This earlier terminology may still linger in parts of the MESSAGE documentation.
2.1.2.1 Concrete Entities

Agent: An Agent is an atomic autonomous entity that is capable of performing some (potentially) useful function. The functional capability is captured as the agent's services. A service is the knowledge level analogue of an object’s operation. The quality of autonomy means that an agent’s actions are not solely dictated by external events or interactions, but also by its own motivation. We capture this motivation in an attribute named purpose. The purpose will, for example, influence whether an agent agrees to a request to perform a service and also the way it provides the service. SoftwareAgent and HumanAgent are specialisations of Agent.

Organisation: An Organisation is a group of Agents working together to a common purpose. It is a virtual entity in the sense that the system has no individual computational entity corresponding to an organisation; its services are provided and purpose achieved collectively by its constituent agents. It is not an amorphous collection of Agents, however. As with a human organisation, the agents are connected by organisational relationships (such as superior-subordinate and customer-provider relationships), control / management procedures, and workflows and Interactions.

Role: The Role concept allows the part played by an agent to be separated logically from the identity of the Agent itself. The distinction between Role and Agent is analogous to that between Interface and (object) Class: a Role describes the external characteristics of an Agent in a particular context. An Agent may be capable of playing several roles, and multiple Agents may be able to play the same Role. Roles can also be used as indirect references to Agents. This is useful in defining re-usable patterns.

Resource: Resource is used to represent non-autonomous entities such as databases or external programs used by Agents. Standard object-oriented concepts are adequate for modelling Resources.

2.1.2.2 Activities

Task: A Task is a knowledge-level unit of activity with a single prime performer. A task has pre- and post-conditions (which are constraints defining States). If the Task is performed when its pre-condition is valid, then one can expect the associated post-condition to hold when the Task is completed. A Task is also implicitly associated with a third State: the state in which the Task is in progress. Composite Tasks can be expressed in terms of causally linked sub-tasks (which may have different performers from the parent Task) by means of logical and temporal relationships between associated States (see the section on notation). Similarly, logical relationships can be used to connect Tasks and the Goals they achieve.

Interaction and InteractionProtocol: The MESSAGE concept of Interaction borrows heavily from the Gaia methodology [Woo00]. An Interaction by definition has more than one participant, and a purpose that the participants collectively must aim to achieve. The purpose typically is to reach a consistent view of some aspect of the problem domain, to agree terms of a service or to
exchange to results of one or more services. An InteractionProtocol defines a pattern of Message exchange associated with an Interaction.

2.1.2.3 Mental State Entities

A wide variety of internal architectures are used for agents, and MESSAGE is intended to be applicable to most of these. However, without some basic abstract reference model it is difficult to say anything meaningful. We suppose that the architecture separates an inference mechanism from a knowledge base and a working memory. The knowledge base contains fixed or slowly changing domain or problem-solving knowledge in a declarative form. The working memory contains more transient sensed or derived information. We view this working memory as an abstract database holding instances of MentalStateEntities, and its contents define the Agent’s mental state. For present purposes we focus on one type of MentalStateEntity: Goal.

Goal: A Goal associates an Agent with a State. If a Goal instance is present in the Agent’s working memory, then the Agent intends to bring about the State referenced by the Goal. Some Goals are intrinsic to the agent’s identity, and are derived from its purpose. These persist throughout the life of the Agent. Others are transient tactical Goals. It is often useful to express the purpose in terms of a utility function that associates ‘goodness values’ with States. The target situation of the Goal is then the one that is estimated to maximise utility (determined dynamically). Note that the agent’s knowledge base needs to include ‘rules’ governing assertion and deletion of (tactical) Goals. One fairly standard rule would be to assert a Goal to provide a given service whenever the Agent agrees with another Agent to do so.

Two other simple but important concepts used in MESSAGE are InformationEntity (an object encapsulating a chunk of information) and Message. The agent-oriented concept of Message differs from the object-orient one in a number of respects. In UML, a Message is a causal link in a chain of behaviour, indicating that an Action performed by one object triggers an Action by another object. In MESSAGE, a Message is an object communicated between Agents. Transmission of a Message takes finite time and requires an action to be performed by the Sender and also the receiver. The attributes of a Message specify the sender, receiver, a speech act (categorising the Message in terms of the intent of the sender) and the content (an InformationEntity).

2.2 Analysis Model Views

Agent-oriented analysis employees a rich set of concepts, which makes it difficult to understand all the aspects of the analysis model from a single viewpoint. It is convenient, therefore, to define a number of views or perspectives that emphasise different aspects of the full model. This set is similar to that used in MAS CommonKADS [Igl97]. Each view focuses on a limited but consistent aspect, but together they provide a comprehensive view.

2.2.1 Definition of Views

An analysis model is a complex network of inter-related classes and instances derived from concepts defined in the MESSAGE meta-model. MESSAGE defines a number of views that focus on overlapping sub-sets of entity and relationship concepts.

Organisation view (OV) – This shows ConcreteEntities (Agents, Organisations, Roles, Resources) in the system and its environment and coarse-grained relationships between them (aggregation, power, and acquaintance relationships). An acquaintance relationship indicates the existence of at least one Interaction involving the entities concerned.

Goal/Task view (GTV) – This shows Goals, Tasks, States and the dependencies among them. Goals and Tasks are both associated with States, so that they can be linked by logical dependencies to form graphs that show e.g. that achieving a set of sub-goals implies that a higher level Goal is achieved, and how Tasks can be performed to achieve Goals. Graphs showing temporal dependencies can also be drawn, and we have found UML Activity Diagram notation useful here.

Agent/Role view (AV) – This focuses on the individual Agents and Roles. For each agent/role it uses schemata supported by diagrams to its characteristics such as what Goals it is responsible for,
what events it needs to sense, what resources it controls, what Tasks it knows how to perform, 'behaviour rules', etc.

**Interaction view (IV)** – For each interaction among agents/roles, this view shows the initiator, the collaborators, the motivator (generally a goal the initiator is responsible for), the relevant information supplied/achieved by each participant, the events that trigger the interaction, and other relevant effects of the interaction (e.g. an agent becoming responsible for a new goal). Larger chains of interaction across the system (e.g. corresponding to use cases) can also be considered.

**Domain view (DV)** – Shows the domain specific concepts and relations that are relevant for the system under development (e.g. for a system dealing with making travel arrangements, this view will show concepts like trip, flight, ticket, and hotel).

### 2.2.2 Consistency Between the Views

Structuring a model in terms of views is widely used in methodologies to describe separately the various aspects of a system. The ability to ensure consistency between the different views of a system is an important factor for the success of design methods that support multiple views.

![Diagram](https://via.placeholder.com/150)

Figure 3 Analysis model, views, diagrams and schemas.

Figure 3 shows how artefacts, i.e. diagrams and schemas, are related to views on the analysis model. Each view refers to model elements that are defined in a single analysis model. This ensures minimal consistency across all views: the concepts exist and have the same name in all the views. It is possible to express more precise consistency requirements within and between views, but has not been studied further in the project.
3 Analysis Process

3.1 Introduction

The purpose of Analysis is to produce a specification (or analysis model). The specification is a model (or collection of views) of the system to be developed and its environment, that is agreed between the analyst and the customer (and other stakeholders). It aids communication between the development team and the customer, and provides a basis from which design can proceed with confidence. The analysis model is produced by stepwise refinement.

Refinement Approach: The top level of decomposition is referred to as level 0. This initial level is concerned with defining the system to be developed with respect to its stakeholders and environment. The system is viewed as a set of organisations that interact with resources, actors, or other organisations. Actors may be human users or other existing agents. Subsequent stages of refinement result in the creation of models at level 1, level 2 and so on.

At level 0 the modelling process starts building the Organisation and the Goal/Task views. These views then act as inputs to creating the Agent/Role and the Domain Views. Finally the Interaction view is built using input from the other models. The level 0 model gives an overall view of the system, its environment, and its global functionality. The granularity of level 0 focuses on the identification of entities, and their relationships according to the meta-model. More details about the internal structure and the behaviour of these entities are progressively added in the next levels.

In level 1 the structure and the behaviour of entities such as organisation, agents, tasks, goals and domain entities are defined. Additional levels might be defined for analysing specific aspects of the system dealing with functional requirements and non functional requirements such as performance, distribution, fault tolerance, security. There must be consistency between subsequent levels. In the MESSAGE project only level 0 and level 1 have been considered.

Analysis Refinement strategies: Several strategies are possible for refining level 0 models. Organisation-centred approaches focus on analysing overall properties such as system structure, the services offered, global tasks and goals, main roles, resources. The agents needed for achieving the goals appear naturally during the refinement process. Then co-operation, possible conflicts and conflict resolution may be analysed.

Agent centred approaches focus on the identification of agents needed for providing the system functionality. The most suitable organisation is identified according to system requirements. Interaction oriented approaches suggest progressive refinement of interaction scenarios which characterise the internal and external behaviour of the organisation and agents. These scenarios are the source for characterising task, goal, messages, protocols and domain entities.

Goal/task decomposition approaches are based on functional decomposition. System roles, goals and tasks are systematically analysed in order to determine the resolution conditions, problem-solving methods, decomposition, and failure treatment. Task preconditions, task structures, task output and task post-condition may determine what Domain Entities are needed. Agents playing certain roles must perform goals and tasks. Consequently looking at the overall structure of goal and tasks in the Goal/task view decisions can be made on the most appropriate agents and organisation structure for achieving those goals/tasks.

The experience in MESSAGE shows that the different views of the system leave the analyst free to choose the most appropriate strategy. In practice a combination of refinement strategies with frequent loop-backs among them are used. The analysis process might start with the OV, then switch to the AV and continue with the IV. The results of the analysis of specific interaction scenarios may lead to reconsider part of OV, and thus to further refining and adapting of OV constituents.
3.2 Inputs to the Modelling Process

It is assumed that the starting point for analysis is an informal (natural language text and diagrams) statement of requirements from the customer plus background documentation including:

**Organisation chart:** This provides a description of how the business organisation is structured in terms of reporting responsibilities. This will in general always follow a hierarchical structure.

**Company goals description:** This may take the form of a company mission statement, but additional information may be required from other sources.

**Business Processes:** This is a codified form of the activities carried out by the organisation represented in terms of workflows. Where the workflow is automated there will be a formal description of the business process. If not this information may only be available through interviewing members of staff.

**Company activities description:** This is related to processes but the emphasis here is on the terminology used within the company and the type of information that the company deals with on a day-to-day basis.

3.3 Overview of Modelling Process

The Analysis Model is a collection of instances of the entity and relationship concepts from the MESSAGE meta-model. At given level of resolution there are two main modelling steps:

1. Identifying the entities and specifying how they relate to each other. It is recommended that this be done initially in three parallel streams which are then reconciled:
   - concrete entities and the acquaintance and organisational relationships, etc. that link them. This involves working primarily within the Organisation View.
   - Goals and Tasks and the logical and temporal relationships among them. This involves working primarily within the Goal/Task View.
   - information and other domain entities and relationships. This involves working primarily within the Goal/Task View.

   These are reconciled by making identifications such as between a Goal and an Agent’s purpose, a Task and an Agent’s service, etc. Note that Acquaintance relationships are refined to become Interactions.

2. Describing the internal details of the entities, primarily Agents, Roles Organisations (and other concrete entities), and Interactions. Description of this detail involves working within the Agent/Role and Interaction Views. However, it will also involve elaboration of the Goal/Task View and Domain View. The amount of detailed specified depends on the project concerned.

A model will normally involve at least two levels of resolution. Level 0 is concerned with defining the system to be developed (represented as a single entity) with respect to its stakeholders and environment. Level 1 refines the system to be developed into an Organisation of interacting Agents. If necessary, further levels of refinement can be obtained by recursively applying the process that obtains Level 1 form Level 0.

The overall process is shown schematically in Figure 4 with the steps labelled by the main view involved. The arrows represent major dependencies rather than a strict time ordering. It is to be expected that backtracking to revise earlier steps will be a common occurrence.
Figure 4. General Workflow schematic for MESSAGE Analysis activity
3.4 Structural analysis, behavioural synthesis and dynamic behaviour

The principal defining characteristic of an agent is autonomy, i.e. an agent is able to decide for itself what to do in a given circumstance. Consequently, while a top-down approach is taken in defining the structure of the MAS, a bottom-up approach is taken regarding its behaviour. An agent's behaviour is thus dictated by its purpose and the circumstances in which it finds itself. The system behaviour is a result of the behaviour of the agents and the interactions between them. This system behaviour must be evaluated against the requirement, see Figure 5. If there are undesirable aspects to the system behaviour then the top-down leg of the cycle must be revisited - perhaps the goals of some of the agents need to be revised, or maybe the detail of the interactions or agent behaviour. This is now discussed in a little more detail.

Section 3.3 above described the top-down analysis phase. At the end of this, there exists a complete Analysis Model. Part of the detailed description of the Agents in the model is a description of their behaviour, i.e. how they respond to events. Interactions couple the behaviour of Agents, thus the detailed description of behaviour, implies the behaviour of the multi-agent system. However, even if the behaviour of individual agents is relatively simple, the behaviour of the MAS is likely to be complex. For example, the same input may result in different a different response at different times because some external factor has changed or because the mental state of on or more of the Agents in the system has changed in a significant way. The guidelines presented here are intended to result in a system that responds in a way that is sensible in the circumstances (rather than rigidly pre-determined). Nevertheless it is important to check for pathological behaviour.

We recommend that this checking is done bottom-up. Thus having checked that the specified behaviour of individual agents appears reasonable (with the remainder of the system assumed to be constant), pairs of agents coupled by interactions are considered. These pairs can then be expanded into chains dealing with end-end workflows. Then the behaviour of the system handling multiple concurrent workflows can be considered, and ultimately the behaviour of the system as it evolves over time. This last point is particularly important where the agents learn or adapt during the course of their lifetimes. Note that almost all agents do learn to some degree as their mental state changes with experience, and this in turn influences behaviour.

Predicting the behaviour of complex dynamic systems is an active research topic, and we do not claim to have a method for checking system behaviour. Nevertheless, there are some steps that can be taken to detect basic problems at the analysis stage. These include walk-throughs by an inspection team. The various UML behavioural diagrams and notations such as Use Case Maps (UCM) are useful here. Note, however, that they are used as descriptive rather than prescriptive tools.

Model-based checking tools such as Imperial College's LTSA [Mag99] may be useful in some circumstances. The richness of an Agent's state space compared to that of a finite state machine is likely to cause problems in large systems. However it may be possible to simplify the behaviour model, and still achieve useful results such as checking for deadlocks in complex protocols used in combination. An alternative to finite state machine-based approaches is to use mathematical techniques and simulations based on continuous models, and to look for system attractors, check
for stability, etc. Such techniques require specialist skills, and investment in time and effort, however.

The sections below consider the top-down phase in more detail.
4 Analysis Diagrams and Case Study

This section illustrates the MESSAGE concepts and views on the UPA4T case study. MESSAGE diagrams are also introduced with proposed notations. For a more complete description of the notation and diagrams used in the case study, please refer to annex 1. Section 4.1 describes the case study that will illustrate MESSAGE analysis and design. Section 4.2 describes the level 0 analysis, i.e. where the system is considered as a black-box, by describing the organisation and goal/task views. Section 4.3 shows the level 1 analysis, i.e. where the system is described, by presenting all of the views.

4.1 Case Study Description

The example application that was selected as a case study to illustrate the methodology was known as the UPA4T (Universal Personal Assistant for Travel) - a task-specific instantiation of a generic personal assistant application. UPA4T was taken from Analysis through Design and to implementation of selected features.

The case study involves assisting a traveller with his travelling activities by making travel arrangements that, as far as possible, maximise his satisfaction. The assistance should provide autonomous analysis of the user's travel requirements, identification of an appropriate package of services (flights, hotel rooms, etc.) from various travelling service providers (TSP), (with the user's agreement) making of the bookings, and monitoring that the travel arrangement is carried out as planned.

Context: travelling from one location to another involves following a travel plan with a very tight schedule. It might involve taking a taxi from one’s home to the airport, taking a flight to an intermediate location, taking a connection to the final destination where a rented car has been booked and can be picked up to drive to the hotel where reservations have been made. Unfortunately for the traveller many things can go wrong with a travel plan. For instance, the taxi can be caught up in a traffic jam, and the traveller will miss his flight, or the initial flight can be late, and the connection will thus be missed. Any of these incidents will prevent the initial travel plan to be followed.

Requirements: Given the fact that many travellers will soon be equipped with proper wireless terminals, TSP now want to improve the efficiency of the whole process by developing a system (distributed both on these terminals and on the terrestrial network) that

- gathers travel requirements from the traveller;
- assist in identifying and arranging relevant travel services offered by the travel operators;
- assist with the booking of travel tickets;
- provide planned travel alerts, Travel Status Alerts and notice of Changes to Arranged Travels;

Appropriateness of an Agent Approach to the case-study: Since the travel arrangement relevant to a travel requirement must be proactively provided to the traveller’s personal travel agent, the system to be developed requires its components to show a high degree of autonomy. Moreover it is almost impossible to exactly foresee all possible problems that can happen during travel and therefore goal oriented behaviour will be needed to re-plan the travel arrangement when problems arise. Finally finding a travel arrangement may require some form of negotiation and distributed co-ordination.

4.2 Level 0 Analysis

4.2.1 Organisation view

The analysis starts at level 0 viewing the system to be developed as a black box and focusing on its relationships to the entities in its environment (e.g. users, stakeholders, and resources).
Two diagrams from the level 0 organisation view are reported as examples showing the main (from the system point of view) structural and acquaintance relationship in the TSP.

Figure 6 describes structural relationships in a level 0 organisation diagram. The diagram shows that the Knowledge Management (KM) system is owned by the TSP. A Salesperson is part of a team and there are several teams in the TSP. It should be noticed that this organisation diagram is a UML class diagram where proper icons have been associated to different stereotypes. At level 0 the system under development, i.e. the KM System, is seen itself as an organisation that will be analysed at level 1.

Figure 6. Level 0 Organisation Diagram (Structural relationships)

Figure 7 shows the acquaintance relationships in the level 0 organisation diagram. The KM system interacts with two roles, the System Administrator and the Salesperson and with two external systems (resources), the Travel Database to retrieve travel arrangements and the Booking Database to insert the bookings requested by salesperson on behalf of Travellers. Moreover, it interacts with the Administrative Team to prepare the bills that will be sent to travellers. A Salesperson interacts with Travellers to gather travel requirements and provide travel arrangements. It has to be noticed that the Salesperson does not interact directly with the Travel Database and the Booking Database. All these interactions are carried out through the KM system.

Figure 7. Level 0 Organisation Diagram (Acquaintance relationships)

4.2.2 Goal/Task view

As for the Goal view the main goal of the system (i.e. providing assistance to the Traveller) is and/or decomposed according to the Goal decomposition diagram in Figure 8 below.
The Goal/Task implication diagram in Figure 8 shows that the main goal of the system (Traveller Assisted) is satisfied when a travel arrangement for the travelling requirements has been selected, and that the traveller is notified of problems that will prevent the travel arrangement from being followed. The TravelArrangementProvided goal on its turn is satisfied when the travelling requirements are known, travel arrangements have been provided by different TSP, that the best travelling arrangement has been identified, and that a travel arrangement has been selected by the traveller. The decomposition of the NotifiedOfTravelProblems goal is not shown. Alternative decompositions can be modelled with or-decomposition notation not illustrated here.

Alternatively, or in conjunction with goal decomposition it is useful to analyse how a given service is realised by a partially ordered set of Tasks. The example in Figure 9 shows the workflow of Tasks implementing the Travelling-arrangement-selection service by means of a Workflow Diagram (i.e. a UML Activity Diagram where tasks are shown instead of activities). The diagram shows the classes that are input/output of Tasks using object flows and the roles that perform the Tasks.
4.3 Level 1 Analysis

4.3.1 Organisation view

Moving from level 0 to level 1, analysis focuses on the system itself identifying at-a-glance the main pieces of functionality required (seen as roles and/or types of agents). The approach followed in this simple case study is to consider only roles initially and to define what agents will populate the system and what roles each agent will play at the beginning of the design process. However the developer is free to start identifying agents during analysis.

Figure 10. Level 1 Organisation Diagram (Acquaintance relationships)

Figure 10 shows the level 1 acquaintance relationships in an organisation diagram. The Traveller and the Salesperson can interact directly, but each of them interacts with roles that can act on his behalf. The interaction between TA Gatherer and TA Selector roles, and the TSP Assistant requires a contract-net protocol to identify the best travel arrangement offered by a TSP. The TSP whose travel arrangement proposal has been selected by the TA selector, interacts with an Airline Booking role to carry out the contract, i.e. to make the flight reservation in the case of a flight.

4.3.2 Agent/Role view

Delegation structure diagrams, Workflow diagrams, and textual Agent/Role schemas are useful to describe the Agent/Role view. The term structure is used to emphasise the fact that it is an extended class diagram that describes the structure. UML provides an aggregation relation (diamond-headed arrow) to express a relationship between a composite entity and its constituents. However, it is rarely the case that the parent-constituent relationship is simply one of aggregation. The structural framework into which the constituents must fit is usually important. The Structure concept is introduced in order to describe such frameworks in re-usable way. In the case of a goal/task diagram it expresses how a Goal of an Organisation is decomposed into sub-goals, which are then, assigned to the Organisation's constituents.

A delegation structure diagram shows how the sub-goals obtained decomposing a goal of an organisation are assigned to the agents/roles included in the organisation. Clearly this diagram is strictly related to (and must be consistent with) both the goal decomposition diagram showing the decomposition of the organisation goal and the organisation diagram showing the agents/roles inside the organisation.
Figure 11 shows a Delegation structure diagram. Only the root and the leaves of the decomposition of the parent organisation goal are shown.

Similarly a workflow diagram shows the roles in an organisation that must perform the tasks necessary to implement a given service provided by the organisation. An example of this type of diagram was show in section 4.2.2.

For each agent/role there is one Agent/Role schema that describes its characteristics. At the analysis level this information is typically quite informal and therefore free text is preferred to a graphical notation. The schema below describes the Assistant role.

<table>
<thead>
<tr>
<th>Role Schema</th>
<th>Travel Arrangement Selector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goals</td>
<td>BestTravelArrangementIdentified, TravelArrangementSelected</td>
</tr>
<tr>
<td>Capability</td>
<td>Some learning capability is required to keep the profile of the traveller updated on the basis of his traveling preferences.</td>
</tr>
<tr>
<td>Knowledge, Beliefs</td>
<td>A profile of the traveling preferences of the traveller to be used to evaluate which travel arrangements best matches a travel requirement.</td>
</tr>
<tr>
<td>Agent requirements</td>
<td>This role will be played by the personal travel agent that actually assists the traveller.</td>
</tr>
</tbody>
</table>

Table 1. Agent/Role Schema for Travel Arrangement Selector

4.3.3 Interaction view

This view highlights which, why and when agents/roles need to communicate leaving all the details about how the communication takes place to the design process.

The interaction view is typically refined through several iterations as long as new interactions are discovered. It can be conveniently expressed by means of a number of interaction diagrams. These diagrams are interaction centric (i.e. there is one of such diagram for each interaction) and show the initiator, the responders, the motivator (often a goal of the initiator) of an interaction plus other optional information such as the trigger condition and the information achieved and supplied by each participant.
The following picture shows as an example the interaction diagram describing the Travelling Request interaction between the TA Gatherer and the TSP Assistant roles. Figure 12 shows an Interaction diagram.

![Interaction Diagram](image)

**Figure 12. Interaction Diagram**

The details of the interaction protocol and the messages that are exchanged between roles can be represented using A-UML sequence diagram [Bau99]. The modelling of an interaction protocol is on the border between analysis and design. The interaction can be detailed either in analysis, or in design. Please refer to Annex 2 for a description of an A-UML sequence diagram.

### 4.3.4 Domain view

The domain view can be conveniently represented by means of typical UML class diagrams where classes represent domain specific concepts and named associations represent domain specific relations. It is typically built in parallel to the other views by adding new concepts and relations as long as they are needed in the other views. Figure 13 provides a very simplified example related to the considered case study.

![Domain Information Diagram](image)

**Figure 13. Domain Information Diagram**
5 Design process and case study

There are many different approaches to agent design and there is little experience that indicates which ones are important. This chapter presents two complementary design approaches with which the MESSAGE project has experimented:

- The first design approach is driven by the MAS organisation and an agent architecture. It considers an agent as an entity that is more than a class: an agent is seen as a subsystem, with an internal architecture that defines the relationships of the different agent components. These components are computational entities that are identified and built by transformation and stepwise refinement from the analysis models. The behaviour of some of these components (e.g., agent control) can be complex (e.g. defined using a BDI schema) or simple (e.g. in reactive agents). The design process is driven by the organisation model in order to assign responsibilities, to define agent interactions, and to model social knowledge.

- The second approach is more agent-platform oriented, and considers that each agent can be mapped to a class. This is mainly derived from the application of most agent models supported by agent building tools, such as Jade or FIPA-OS in which there is one Agent class from which to derive the specific agent type. This approach is valid when agents are simple in behaviour and can be modeled using a classical state machine approach, e.g., as in typical OO languages such as Java. The main concern here is how to organise agent interactions.

5.1 MAS organisation and architecture driven design

The purpose of the design is to produce computational entities that represent the MAS that appeared at the analysis level. In general, the artefacts produced in each of the analysis models should be transformed into computational entities, which should be finally implemented. This means that the analysis entities should be translated in terms of subsystems, interfaces, classes, operation signatures, algorithms, objects, object diagrams, and other computational concepts.

The guidelines for helping the designer in the transformation process are based on the following principles:

- The central computational entities to be developed will be agents. Agents will coexist with resources and users. So the system will be finally composed by these three kinds of entities.

- The organisation defines the architectural framework where agents implement the functionality of the system to be developed. However, there is no computational entity corresponding to the organisation meta-class. An Organisation represents different aspects/views of the system composed by agents and other entities such as resources. The standard views provided by the organisations are: the description of the organisation in terms of purpose, rules etc. The structural description in terms of organisational structures, and workflow description in terms of task description. The main components of the organisation are the agents

- All the entities identified in the Organisation view (Tasks, Purposes, Resources, Information Entities, Workflows, Organisation Structure) will be part of the knowledge of the agents that will form the final Multi-Agent System. By knowledge, it must be understood all the problem solving information the agent needs to achieve its goals. More concretely, the knowledge that the agent specifies is behaviour knowledge, domain knowledge, and social knowledge.

- Agent design should provide a computational model of the agent behaviour which should be consistent with the rest of the views defined in analysis –Interaction view, Task/Goal view, Domain view. This means that the entities identified in these views will be refined into computational entities integrated in the agent knowledge.
Agents cannot offer to execute tasks that are not considered in the Organisation view. This means that the tasks that an agent can execute are a subset of the tasks that appear in the Organisation.

It seems natural that the main computational entity in agent based system would be the agent. However, it is less clear what would be the most appropriate architecture for the agents identified in analysis. The approach for obtaining the agent architecture is based on architectural patterns.

Table 1 shows a possible mapping from analysis entities to design entities. This mapping illustrates the design choices made during the design of the prototypes. More details and examples on the selection and use of architectural patterns are explained in the next sections.

<table>
<thead>
<tr>
<th>Analysis entity</th>
<th>Architectural Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>Package of goals/tasks that the agent has to perform. Communication Resources (servers and proxies) to achieve interaction.</td>
</tr>
<tr>
<td>Interaction</td>
<td>Protocols that the agent has to use to communicate with other agents. TINA-like Sessions [Ber00] to control the progress of communication. Rules to launch the execution of tasks following some control model.</td>
</tr>
<tr>
<td>Agent</td>
<td>Cognitive and reactive agent architecture.</td>
</tr>
<tr>
<td>Goal</td>
<td>Objective to hold the lifecycle of control. Objective Space to deal with objective decomposition and to handle the satisfaction dependencies of objectives. Objective taxonomy.</td>
</tr>
<tr>
<td>Task</td>
<td>OMG Task-Session [Omg00] model to define a computational task. Task taxonomy to represent all tasks in the system.</td>
</tr>
<tr>
<td>Organisation</td>
<td>Package structure to represent the architecture of the system. Social knowledge to constraint the behavior of the system.</td>
</tr>
<tr>
<td>Resources</td>
<td>Objects representing system resources.</td>
</tr>
<tr>
<td>Domain entities</td>
<td>Ontology for knowledge definition. Object representing domain entities. Domain entity managers to handle collections of domain entities.</td>
</tr>
</tbody>
</table>

Table 2. Translation from analysis to computational entities

5.1.1 Design activities

The design process consists of a series of transformation and stepwise refinement activities that are organised in a consistent workflow, which is shown in Figure 14.

![Figure 14. Main activities](image)

---

2 For a more detailed specification of this mapping, please read PIR 3.6 from MESSAGE public web site

3 The session defines the context for an interaction or a set of interactions. The session state determines what communicative actions an agent can perform at a given time, and what can expect. This concept has been borrowed from TINA.
The steps needed for obtaining design artefacts are the following:

- Refine analysis entities into computational entities. This involves definition of classes, interfaces, attributes, methods, and UML diagrams using conventional software engineering techniques.

- Select an Agent Architecture. Selection criteria are based upon the requirements of the agent functionality defined in analysis. Cognitive architectures are required when the agents need to achieve complex communication and processing mechanisms using reasoning and learning capabilities. Reactive agent architecture might be used when simple control mechanisms are involved.

- Fulfil the Agent Architecture: Once the appropriate architectural pattern has been selected the designer should produce application agents by customising the generic components that appear in the initial agent architecture. This activity is strongly dependent upon the selected agent architecture, and upon the role of the agent in the application.

- Update/Modify existing views. During the refinement process design decisions would affect existing analysis views. In those cases, designers should modify original views keeping in mind that consistency and coherence between analysis models and design constructs should be assured.

- Structure results according to the Organisation View. The organization view is important in order to obtain an architectural representation of a multi-agent system. The grouping of agents and relationships established among them is a valuable starting point for system architecture definition.

Next sections provide details about how to perform each of the previous activities.

### 5.1.1.1 Refining the analysis entities

The interaction view in analysis should be refined to express interactions between computational components in design. New notations like UML's sequence diagrams and use cases realisations are needed. The detail of analysis entities is achieved through the following steps:

- Generate Use Cases: for each interaction view instance, generate a use case with the same name.

- Refine Use Cases: if necessary, take each use case and refine until granularity of the use-case is the appropriate.

- Generate sequence diagrams: Translate interaction view instance into a sequence diagram. This is almost trivial.

- Refine sequence diagrams: the sequence diagrams should be as detailed as possible. This implies adding relationships with tasks, resources, and fully specified messages. There is only one constraint: roles and agents are kept as black boxes.

The sequence diagrams identified during the refinement process define what is expected from the systems in terms of tasks, resources, and roles. This information is quite useful since almost a 50% of the agent's computational entities (Resource Layer, Perception & Communication Layer, and Decision & Management layer behaviour part) originate from the translation of interaction entities.

### 5.1.1.2 Selecting the Agent Architecture

The choice of an agent architecture is a key issue in the design since it influences other design activities. The selection criteria followed in the case studies are based on functional considerations. The functionality of the application justify the use of a cognitive architecture since reasoning and learning mechanisms are needed. There are many agent architectures in the literature [Nwa96] [Woo95] which should be studied prior to starting the design. Each one has its particularities, like social capabilities, mobility, or reasoning.
The architecture used in the project is made up of generic components which may be adapted for other applications. These generic components might be transformed according to the required functionality/role of the agent in the organisation. The instantiation act requires to determine in an early stage what components the agent will have, how they will relate, and how they should behave (in generic terms).

The needs in UPA case study were to support user interaction, user modelling techniques, information extraction, and parallelism of agent interactions. As a result, cognitive as well reactive capabilities were needed. Cognitive agents would be responsible of management of user interaction and user modelling. Meanwhile, reactive agents would perform information extraction on demand. In both, the agent architecture should provide facilities to manage threads of communication independently.

An architecture that gathers these requirements is the one presented in [Gom00] and [Gar01]. This architecture has been used in Eurescom projects P712 and P815, and is simple enough for a quick understanding of the steps to follow in the methodology.

The agent architecture is presented in Figure 15 and will be the one used to illustrate an example of generic agent architecture instantiation. It structures agent components in four layers, see Figure 15.

**Figure 15. Layer architecture for the agent**

- Perception and Communication layer (PCL): contains perception and communication processors for getting information from the environment, and interacting with other agents.
- Decision and Management layer (DML): control actions of the agent through deliberative decisions. It also considers the management of the agent (creation, destruction, and monitoring of the agent).
- Domain Layer (DL). It groups domain specific entities. It may contain also the components that represent the effectors of the agent (i.e. task execution), since they are domain dependent.
- Resource layer (RL). It includes the internal resources that the agent may need. These should be distinguished from organisation’s resources, since the ones from the organisation belong to the system. Examples of internal resources are GUI resources.

This architecture allows defining a broad spectrum of agents. Components are identified for the different layers depending on application requirements. For instance, it could be possible to restrict the agent to the Perception and Communication layer and the Resource Layer. This is the case of a
purely reactive agent. On the other hand, cognitive agents ought to instantiate the complete architecture.

Next sections will mention which components are identified and in which layer they should be allocated. Layers will be named using the acronyms appearing in Figure 15.

5.1.1.3 Structuring the results according to the Organisation view

The Organisation view is translated into design as a package structure. Specific packages within the organisation contain descriptions of the purpose, the tasks, the organisational structure and the resources. The agents will be allocated in different packages. (see example from UPA case study in section 4.1). Figure 16 shows an example of Organisation view obtained from the analysis. These view presents the different groups into which the organisation is structured as well as the resources involved and the existing workflows.

Figure 16. Travel Assistance Agency Organisation View in the analysis

In the design, the organisation view is used to define the system architecture. Note that almost every entity appearing at the analysis, see Figure 16, appears in the design in form of UML packages, see Figure 17.
Figure 17. System architecture in the design based on the analysis Organisation View

5.1.1.4 Fulfilling the Architecture

After the refinement using UML notation, there is enough information to proceed with the fulfilling of the architecture. Though each layer could be detailed concurrently, it is recommended to follow a specific order.

Previous stages allows to:

- Extract entities related with communication. Sequence diagrams provide the messages interchanged, the protocols, and the goals pursued in the communication.
- Extract entities related with task execution. Sequence diagrams states in what order task have to be executed as well as the activities involved.

So, it seems rational to:

1. Determine interaction fulfilling PCL and RL. Most data, coming from the interaction view, deal with communication issues. Communication resources are needed as well as mechanisms to guide interaction.
2. Specify behavior knowledge. Having tasks to execute and knowing what they are supposed to do as well as in what conditions they should be executed.

This is the core of the agent from analysis to design translation. Around these two issues (interaction and task execution) the rest of the translation process is articulated. The process focuses on a concrete agent architecture (see section 5.1.1.2). However, the aspects mentioned, the procedure followed, and the results are general enough to be applied to other agent architectures. Following, there is an overview of the different activities to fulfil the proposed agent architecture:

- **Identifying components for external behaviour:** It considers which components are required to support interaction. The components that have been selected are servers, proxies, communication protocols, and sessions. These components provide communication management functionality to handle the different interactions in which the agent involves.
- **Fulfilling the domain layer.** It checks which components are required to support the desired functionality. In this case, tasks, domain entities, and resources are used. It is common to
many agent based approaches to use the task concept as the computational representation of actions of the agent.

- **Fulfilling the Decision and Management Layer.** It studies which components are required to define the control of the agent. In this case, an objectives and rule based approach is applied.

It is clear that these elements are not constrained to this agent architecture, and that they could be used again in different agent architectures. The difference between this process and other processes for other agent architectures should be mainly the allocation of these elements.

5.1.1.5 **Identifying components for external behaviour**

For external entities, like other agents, an agent is a computational that has interfaces to perform complex interactions. To determine the computational components which determines the agent external behavior, the following steps are needed:

- Generate interfaces: from the sequence diagrams, identify the set of functions associated with each role.
- Generate proxies: each agent that is sending a message to other agent must have a proxy to that agent. The proxy represents the other agent internally.
- Generate servers: proxies connect with servers. Servers offer an implementation for an interface that is accessed by proxies.
- Generate session entities: for each interaction, identify groups of messages and consider them as a session entity. The grouping can be further refined with the creation of protocols. Usually, a session entity can be obtained directly from every sequence diagram.

As a result of these activities, there are:

- Interfaces: roles in the interaction are manifesting a set of functions, since in many cases a message or sequence of messages ends with a task execution. Then it is possible to define directly an interface that contains the set of functions associated with a role.
- Communication resources: Proxies and servers supporting external interfaces. Proxies allow connection with servers allocated in other agents. Servers are the entities that provide externally the role interface meanwhile proxies offer the same interface but for internal use of the agent.
- Communication protocols: messages are grouped into protocols. Protocols can also contain nested protocols. Nevertheless there is a higher bound for this process: the session entity.
- Session entities: as it was mentioned, session entities constitute the more abstract communication concept in terms of architectural components.

As an example of fulfilling of PCL and RL layers, Figure 18 shows the resource layer of the UPA flight notification and information assistant. It contains servers and proxies used to perform the interactions identified in the analysis.
Figure 18. Resource Layer for Agent UPA Flight Information & Notification Assistant

These resources are used at the perception level to track communication between this agent and others. At the perception level, the communications activities are abstracted into session entities. These session entities have a correspondence one-to-one with the interactions identified in the analysis. The complete list of identified session is large. Following there is an extract of the list. It was produced associating each interaction view instance to a session entity.

<table>
<thead>
<tr>
<th>Name of the session entity</th>
<th>Scenario</th>
<th>Actors</th>
<th>Purpose of the session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer_User_GenerationS</td>
<td>Starting Customer Service</td>
<td>The user and the player of the customer user access role</td>
<td>Creation of a trial user session in the system</td>
</tr>
<tr>
<td>User_Profile_RetrievalS</td>
<td>Starting Customer Service</td>
<td>Anybody and the UPA/UserProfileManagerR</td>
<td>Retrieve a specific user profile</td>
</tr>
<tr>
<td>Flight_Status_Info_ProvisionS</td>
<td>Flight Status Info Provision</td>
<td>The user and the player of the UPA/FlightInformationR</td>
<td>Determine which flight is to be known and give the user the requested information</td>
</tr>
<tr>
<td>Obtain_Flight_StatusS</td>
<td>Processing Notification</td>
<td>Anybody and the UPA/InformationFinder role player</td>
<td>Obtains the updated flight status of a certain flight</td>
</tr>
<tr>
<td>Define_Flight_NotificationS</td>
<td>Flight Notification Definition</td>
<td>The user and the player of the role UPA/FlightInformationR &amp; Notification role player</td>
<td>Define a flight notification</td>
</tr>
<tr>
<td>Add_Flight_NotificationS</td>
<td>Flight Notification Definition</td>
<td>Anybody and the UPA/FlightNotification role player</td>
<td>Add a new notification to consider</td>
</tr>
<tr>
<td>Obtain_Flight_Status_From_AirlineS</td>
<td>Finding Flight Status Info</td>
<td>The player of the role UPA/InformationFinder and the AirlineInformationDesk role player</td>
<td>Obtains an updated information about a flight status from the airline information desk</td>
</tr>
</tbody>
</table>

Table 3. Identified sessions

5.1.1.6 Fulfilling the domain layer

In this section, the domain entities as well as their managers are defined. Each domain entity defines the operations that it supports. Tasks will also be included as part of the domain layer. Tasks deal with operations to be performed over domain entities, resources, or other agents in order to produce certain results.

Tasks might also be used to achieve integration with legacy systems. Tasks may view legacy systems as resources that provide complex functionality. This will allow to simplify interaction with proprietary software.
The activities involved in this section are:

- Creation of domain entity managers. For each domain entity, a domain entity manager should be created. Domain entity managers provide primitives to cope with management of domain entities. Domain entities come directly from the analysis’ domain view in form of objects.

- Creation of tasks. Tasks are defined as computational objects able to perform an algorithm in certain situations. Sequence diagrams from section 5.1.1.1 provides a context for task execution. Nevertheless, this may not be enough, since a sequence of actions to be executed is also needed. In section 5.1.1.1 only actions involving resources can be detailed. However, on dealing with domain entities, it is possible to have to complete process.

As a result, there are:

- Domain entities: a domain entity is directly an instance of the domain view
- Domain entity managers: there is one domain entity manager for each domain entity type. A domain entity manager provides operations to create, obtain, monitor, and destroy domain entity instances.
- Tasks: an object, probably with state, that is able to use domain entities and resources to perform an action over the world

![Figure 19. Flight Manager and Travel Managers](image)

As an example of domain entity manager, Figure 19 shows managers for flight status entities and travel entities. Flight status is the status of a flight of some user. The travel includes the flight and the booking information. Through these managers, it is easy to handle big numbers of flights or travels.

5.1.1.7 Fulfilling the Decision and Management Layer

The DML is responsible of the control of the agent. The control is oriented towards an implementation of the expected behavior of the agent as it was specified in the interaction view. So DML will have to:

- Decide what to do next based on the current mental state. This decision is expressed in form of goal satisfaction policies.
- Define and maintain a mental state. All the knowledge needed to guide agent’s actions should be present in the DML at the moment of the decision.
In the analysis the behaviour does not appear as such directly. Nevertheless it is defined collecting ideas from:

- **Goals.** A goal has a set of tasks that can satisfy it. Then, it can be said that they have an associated behaviour.
- **Roles functionality.** A role has an associated set of actions and appears in a determined number of interactions.
- **Interaction view.** Interactions are the high level specification of what roles (and, consequently, agents) should do.
- **Agent view.** Although it does not appear in the Agent view, it is mentioned in the schema where agents are specified informally.

There are relationships among the knowledge definition activities. These relationships originate in the knowledge relationships (see Figure 20). Despite these relationships, each part of the knowledge can be developed in an isolated way.

![Figure 20. Relationships among the different types of knowledge](image)

**Domain knowledge** provides an ontology upon which reasoning can be performed. Social knowledge provides also an ontology and a set of constraints to specify external expectations. Behaviour knowledge can be defined without domain knowledge and social knowledge, though, this way, it would be incomplete.

At high level, the activities are:

- **Generate management functions:** these are functions responsible for creating, destroying, and monitoring the agent. These functions are identified from start-up and shutdown scenarios.

- **Define social knowledge:** analysing the organisation view instances, social laws in form of constraints are identified. All the relationships that appear in the organisation view are potential candidates to become restrictions for the behavior of the agent. As a first task, designers have to identify which are the more useful in order to incorporate them into the behavior of the agent.

- **Determine meaningful social relationships.** The relationships of the Organisation View are inspected in order to determine their utility. This decision can be particular for each agent. So an agent can find useful relationships among agents defining constraints with respect to requests or inform speech acts. On the other hand, the same agent can consider useless associations among tasks and resources.

- **Create representations for social relationships.** UML notation can be enough for design, but it should be considered an alternative representation closer to the knowledge processor to be used.

- **Define domain knowledge:**
• Determine meaningful domain entities. As with social relationships, the same reasoning can be made.

• Create representations for domain entities. As with social relationships, the same reasoning can be made.

• Define behavior knowledge: with rules, objectives, and tasks the behavior of the agent will be outlined.

• Create objectives from goals. For each goal, create an objective. An objective is a computational representation of a goal.

• Refine objectives. If necessary, objectives can be refined. Explosion of use cases may lead to new objectives that were not considered before. It is no sense to keep on refining analysis goals. The rational is to continue the process with the objectives.

• Create objective management rules. Each objective has a well-defined lifecycle. Management rules are created to represent that lifecycle in the design.

• Refine rules. Refinement of objectives leading to refinement of rules.

• Verify the resulting set rules. Though a formal process of verification is not the scope of this methodology, it should be noted that some manual test has to be performed over the resulting rule set. An easy proof is to check that all rules can be fired. The number of rules can grow very quickly. It is possible that at the end, not all the rules are fired. This is a high-risk source of errors, since there is code that has not been tested.

Figure 21. Allow User Define Flight Notification Goal decomposition with tasks and evidences

As an example of behaviour and social knowledge definition, following there is an extract of UPA case study. Domain knowledge is present in the examples in form of knowledge level representation of domain entities.

The example starts from the goal decomposition together with relationships with tasks and evidences shown in Figure 21. The decomposition deals with the satisfaction of the goal: provide the user with everything he needs to define a flight notification. The needs are basically some window where the data is written and functions to validate the data.

Within this decomposition, designers have to provide rules for management of goal Allow User Define Flight Notification. In any case, for every objective the management rules refine, start associated tasks, gather evidences produced by those tasks, and decide if the goal is solved or failed, managing also its success and failure. As a first draft for the rules, the proposed graphical notation can be used (see Figure 22).
Figure 22. Graphical notation to represent evidence gathering and success management for goal AllowUserDefineFlightNotificationG

With more detail, one the management rules for Allow User Define Flight Notification goal is described in Table 4. The complete specification covers all types of management rule and is included in the same table template.

| Objective: | CreateVisualResourcesNotificationDefinitionG |
| Purpose: | In order to type the notification data a GUI should be shown. This goal is satisfied when such GUI is shown |
| Rule type: | Task execution |
| Conditions: | Objective is in REFINED state |
| Action Plan: | • Start CreateNotificationDefinitionWindowT to show a window with the form data  
• State is SOLVING |

Table 4. Example of Task execution management rule (reduced version)

With respect to social knowledge, it is extracted from the social relationships identified in the Organisation view instantiation. The Organisation view provides an ontology upon which the social rules can be defined. As an example, Figure 23 shows an instantiation of a social rule to represent rejection of requests from non-subordinated agents. Rejection conditions are expressed in terms of representations of the relationships appearing at the organisation view, in this case the predicate isSubordinatedTo. In the Organisation view, this relationship is represented as an edge with direction from the subordinated entity to the subordinating entity.

Figure 23. Instantiation of a social rule to express rejection of requests in case subordination does not exist.

5.1.2 Design Guidelines

All along the design process, no specific technology has been considered. Issues like agent platforms should not affect an existing design. Also, the kind of reasoning of the agent has not been
constrained, since the rule mechanism is present, or could be implemented, in most declarative control facilities (PROLOG, CLIPS, JESS, ILOG Rules). Moreover, the control itself could be a simple state machine that behaves as if it were goal driven. In such a state machine, rules would represent valid state transitions. In fact, this design process has been assessed with the development of a MAS system for a prototype (as reported in PIR 4.3) with four types of agents. All agent types were deployed on the Jade platform, and agent behaviour was implemented for two of them using ILOG JRules, with the others were of a simple reactive nature. The design was done independently of the platform, and the reference agent architecture was mapped on the simpler Jade agent model.

Though the proposed high-level design follows a specific reference agent architecture, the process itself is reusable. On using other agent architectures, the same reasoning should be performed as the one shown with the proposed architecture.

Although this approach is based on a client-server communication paradigm, the combination of two complementary client-server communication flows leads to a peer-to-peer schema, like the one used in FIPA for communication between agents. On the other hand, implementations based on ACL message communication usually have a translation process of the message that leads to a simple method invocation. In this sense the proposal presented in the high-level platform design is perfectly compatible with FIPA platforms. To adapt the schema only a slight modification of the communication resources is needed. The proxy represents the initiator of the FIPA-protocol meanwhile the server is the collaborator. In the extremes, normal method invocation can be used meanwhile in the middle messages are passed.
5.2 Platform specific design process issues

This section describes a design process for a low level design, i.e. one that is specific to a target platform, in contrast to the high level design which is platform independent, and was described in section 5. This section starts by describing in which way the choice of an agent platform will have an impact on the low-level design. The next sub-section describes a process for taking the analysis model and creating a design model targeted for implementation on FIPA agent platforms. The last section identifies some guidelines for the design of interaction protocols.

5.2.1 Target platform issues

The choice of target agent platform will impact on the lower level design of an agent application. This section examines the issues that typically arise when producing a design for a particular agent platform. A platform oriented design process is described and illustrated by agent application design examples.

Agent platforms provide basic runtime services to agents thus freeing Agent application developers to concentrate on implementing application logic. Agent Platforms typically provide an execution environment for agents, which includes mechanisms supporting agent naming and directory services, inter agent message transport, co-ordination protocols and ontology support. Agent platforms may also provide a development toolkit to speed the development of Agent applications.

A large number of agent systems have been developed but most have followed proprietary approaches with the result that very few agent systems are capable of inter-operating. The one significant area of interoperability between early agent systems was the adoption of KQML (Knowledge Query and Manipulation Language) for speech act communication and KIF (Knowledge Interchange Format) for expressing the content of speech acts.

Since 1996, more extensive standardisation work has been carried out within FIPA (Foundation for Intelligent Physical Agents) resulting in FIPA interoperability specifications for Agent Management, Agent Communication Agent Message Transport, Agent Abstract Architecture and Agent Applications. A number of FIPA compliant agent platforms have now become available, some under the open source license [Bel99], [Fip99b], [Lea01]. This represents an important grouping to which an agent-based methodology should cater.

The agent services provided by the target platform will impact the form that the design takes. FIPA specifications impact the agent communication model and the content language. The FIPA inter-agent communication model is speech act in nature. All platforms will of necessity provide support for communication such as FIPA compliant naming services and inter agent communication primitives. FIPA specifies SL as the content language in inter-agent communication. Agent platforms will typically provide support for parsing messages composed in FIPA content language.

In addition, the form of the final design will be impacted by the choice of the specific agent platform. A number of different FIPA compliant Agent platforms are currently available. We will focus on those based on an OO programming model, specifically Jade, an open source FIPA compliant platform supplied by Tilab [Bel99].

1. This type of FIPA compliant Agent platforms is based on an OO programming model. This requires that the low-level design should be framed in terms of OO artefacts.

2. The specific execution model of the Agent Platform which may provide a higher level abstraction mechanism for composing behavioural dependencies that support concurrent execution. As an example, the Jade platform utilises an execution model based around the concept of the behaviour. This is not dissimilar to a Java thread except that it is significantly more lightweight and in addition supports the construction of computational [Bel00]. This has an impact on the design in that it requires assumptions which govern how multiple Behaviours co-operatively share a single agent message queue.
5.2.2 Description of Low Level Design Process

This section describes a process for taking the Analysis model and creating a design model targeted for implementation on an agent platform.

The process is overviewed in Figure 24 above. The inputs to the modelling process are the products of the Analysis phase, specifically the Analysis model expressed as the Organisation, Agent, Domain, Goal/Task and Interaction views. The activity diagram in Figure 24 shows the analysis modelling activities in the upper section (green box) and how these activities precede design. The process consists of the following stages and the outputs.

**Model Interactions** - During Analysis, an interaction view will have been created see section 4.3.3. Experience indicates that the choice of protocols (e.g. FIPA interaction protocols) has fundamental implications for the way in which interactions are structured. The issue of casting protocols thus needs to be addressed early, ideally in analysis (this is assumed in the process diagram above). If this is not done during analysis, then during design there will have to be a significant effort made to recast interactions.

**Generate example ACL Messages** - produce an example ACL message for each message in the interaction view. Generating example messages (such as in Figure 25) tests the completeness of the Ontology and may throw up additional entities that are required in the Ontology. Before messages are created, a decision has to be taken regarding which content language to use. The options that enjoy widest support are either use of KQML/KIF or FIPA-ACL /SL. FIPA defines a set of languages SL0, SL1 and SL2 that can be used for defining the content of ACL messages [Fip99]. SL0 is currently the only content language for which parser support exists in Jade and is the most convenient choice. SL0 has certain limitations [Fip99] and for some applications SL1 or SL2 will be a requirement.

**Model Ontology** - The previous activity (Generate example ACL Messages) may uncover refinements that need to be made to the ontology as defined during Analysis. This may be either
changes to the entities described in the ontology or the actions that one agent may request of another. The Ontology will typically be modelled as a UML class diagram see Figure 13.

The following stages in the process apply specifically to an OO based platform.

**Derive Agent Classes** - produce a UML class diagram, see Figure 26, defining each type of agent class (typically one for each role in the organisation view). This is a classic approach in an OO type platform such as, Jade [Bel99], FIPA-OS [Fip99b], and LEAP [Lea01].

As an example, Jade provides two Agent classes that are available for subclassing application-specific agent classes. The Agent class is the most basic choice for subclassing while the GUIAgent class better supports Agents that are required to interface with event driven GUIs.

**Derive Agent Object Acquaintances** - identify instances of agents of a given type, as well as the acquaintances of each agent i.e. the other agents this agent will have to talk to and an indication about how to retrieve their names. In a FIPA compliant platform this can be achieved by searching the DF. The Agent instances and acquaintance information can be diagrammed using a UML Object diagram as in Figure 26. This is similar to the Organisation view in Analysis but the entities being related are classes rather than agents.

**Define Support Classes** - produce a set of class diagrams for support classes such as GUI components.

The following stages are required to design the way that messages are handled by the agent application. The concurrency mechanisms employed in message handling will vary from platform to platform. As an example the techniques used by the Jade [Bel99] and Leap platforms [Lea01] are assumed.

**Derive Message Handler classes** - devise an inheritance structure of application specific behaviour classes for handling the communication (class diagram) see Figure 27. The necessary information can be abstracted from the A-UML diagrams in the Interaction View, as well as the breakdown of tasks in the Task View and the Agent definitions in the Agent View.

Jade provides an execution model that supports concurrency in a way that is natural for dealing with FIPA interaction protocols. The central concept is the Behaviour. Behaviours embed all the operations related to the handling of the correct sequence of messages (including timeouts) in an interaction protocol so that the developer only has to deal with the application specific handling of the ACL messages exchanged during the protocol. Behaviours can be added and removed dynamically within an Agent and can be composed in sequence or in parallel thus forming complex behaviours.

Jade also provides a set of basic Behaviour classes, from which the developer can subclass their own application specific Behaviours. Figure 27 shows the classes provided by Jade and some example application specific Behaviours. Composition of Behaviours is supported by the classes NonDeterministicBehaviour and SequentialBehaviour. Instances of a SequentialBehaviour can contain a number of instances of behaviours that are executed sequentially. An instance of a NonDeterministicBehaviour can contain a number of Behaviour instances that are executed in a non-deterministic order. Jade also provides a set of predefined Behaviours supporting the implementation of all the interaction protocols defined by FIPA [Bel99].

**Derive Message Handler Object Structure** - specify a containment hierarchy of behaviour instances to handle the protocols defined in the Analysis Interaction View. This can be illustrated using a UML Object diagram as in Figure 27. The diagram shows the sequence in which behaviours within a particular agent are activated.

**Define Message Routing** - produce a decision tree for specifying how incoming messages are routed to individual Message Handlers (in Jade to an individual Behaviour instance). A template is a precondition associated with each behaviour instance, which governs to which ready Behaviour a given incoming message should be passed. The precondition can contain variables corresponding to slots in the ACL message envelope, such as the sender agent name, the ontology, the conversation-id and the in-reply-to field.
Figure 25. Example message including content

```prolog
{QUERY-REF
:sender <an Informer4T agent>
:receiver <an Informer4T provider agent>
:ontology TRAVEL-ONTLOGY
:language SL
:content
(all

?x
(matches

?x
(TransferRequirement
 :from TURIN
 :to PARIS
 :when (TimeRange
 :begin 20-Feb-2001/7:00
 :end 20-Feb-2001/11:00
 )
 )
)
)
}
```

Figure 26. Example Agent Class diagram, and Agent Instance acquaintance object diagram

Figure 27. Inheritance diagram for Behaviours and Behaviour instance containment object diagram
5.2.3 Interaction Protocol Design

According to what was previously mentioned in the Analysis Interaction View, the modelling of an interaction protocol is on border of analysis and design and the interaction can be detailed either in analysis, or in design.

This section introduces some discussion about this issue based on previous experience when developing for a FIPA agent platform in general and Jade in particular.

Using FIPA protocols places a non-trivial overhead. In order to utilise FIPA protocols the logic of an application may need to be recast somewhat. This needs to be done early on within the development cycle, ideally during Analysis but early in Design at the latest.

A-UML (see annex 2) provides a valuable technique for specifying Agent interactions. The disadvantage of using A-UML diagrams is their complexity and thus a diagramming tool is essential if they are to be used. If such a diagramming tool is not available then the next best approach is using FIPA protocol diagrams as in Figure 28.

![Figure 28. FIPA protocol diagram](image)

There will in general about the same amount of effort in developing the interactions between agents as in developing the internal agent logic. Where development is distributed among a number of teams, each team will often be given responsibility for a different agent. Under these conditions it is vital to have a detailed and solid design for the agent interactions before internal agent development begins. Otherwise there is a danger of agent implementations having to undergo substantial reengineering at integration time.

One of the more difficult areas in a design task for a Jade Agent that interacts with a User is the design of the interface between the GUI and the behaviours. Event driven GUIs use a concurrency mechanism that doesn’t fit naturally with that used in Jade. This can lead to flows of control that are difficult to follow, with GUI methods being executed in the Agent thread and vice versa. An effective design technique for integrating GUIs with Jade agent applications is an area for further study.
6 Evaluation of the methodology

This section briefly describes the features that an AOSE methodology needs to possess to be called complete and mature, i.e. that could be applied by mainstream software engineering departments to deliver agent-oriented applications. It then provides an evaluation of MESSAGE based on two case studies.

6.1 Towards a Complete and Mature AOSE Methodology

Such a methodology should at least cover the analysis, design and implementation phases to ensure that a consistent set of concepts has been identified to allow a smooth transition between phases.

6.1.1 Analysis

From a software engineering perspective, the main activities in analysis are elicitation, specification and validation of required properties. Concepts and languages for analysis are needed to deal with a multi-agent system as a whole with organisational and co-ordination properties, individual agents with their properties. Dedicated basic concepts and languages are needed to describe each of these levels.

Methods to support analysis and verification are needed. Analysis methods need to support elicitation of requirements and use cases from stakeholders, validation of requirements and use cases with stakeholders, specification of informal, semi-formal and formal specification of (behavioural) requirements and use cases. Furthermore, traceability between the different artefacts is needed. Verification methods are needed to prove global properties of the overall multi-agent system and properties of the agents.

These concepts and methods need to be based on well-defined foundations. This entails at least semantical foundations for concepts representing dynamic aspects such as time and action, and locality, as well as foundations for concepts representing mental states such as belief, desire, and intentions.

Reusable ontologies can play an important role in specification of requirements and properties used during verification.

6.1.2 Design

Design involves the specification of a multi-agent system software architecture and the agents within the system. A design should be an implementation-independent description of the system.

Concepts and languages for design are needed for the architecture of the multi-agent system as a whole (organisational structure), the individual agents, specific architectures for components or other structures within an agent. A framework of reusable patterns and ontologies is needed.

Design Methods are needed to produce specifications that can be analysed. It should be possible to create a conceptual design first and transform it into a detailed design. The complexity of a design has to be kept manageable by some form of decomposition of the overall design, and the possibility to design by configuration of reusable components. To obtain solid foundations, a formal semantics of the design modelling language is important.

Libraries of generic models and patterns are needed, such as generic organisation models, generic task models, ontology patterns, etc.

Guidelines and heuristics are needed for supporting the designer during the design process in the following aspects:

- Transforming analysis entities into computational entities. These guidelines need to cover top-down and bottom-up aspects within a design process and should include the exploitation of reusable structures.

- Selecting and adapting the architectural patterns to obtain specific organisations, agents, tasks, goals, and ontologies needed in the application
• Detecting inconsistencies and ambiguities between analysis concepts and the corresponding design constructs. Consistency checking and verification methods are needed to help the designer in the validation of global aspects such as how agents perform the functionality of the system, and specific aspects such as how individual agents achieve specific goals and tasks.

### 6.1.3 Implementation

Current implementation concepts and toolkits are very diverse. They can be closely related to the agent concepts that also play a role in analysis and design, but an implementation can also be based on the concepts available in the chosen implementation environment.

An important question is how an implementation is related to a detailed design specification. If no precise relation is defined, then it is hard to use results from analysis to make claims about the (implemented) system. Implementation can be derived from a detailed design by automated code generation, or performed by hand. In both cases, implementation and testing of (partial) prototypes can be done interleaved with the design process. Implementation should reuse components from libraries of reusable agents or organisation structures for example. Moreover, during implementation testing methods are important, which should relate to other concepts used in the analysis.

The development process needs well-defined traceability link between conceptual modelling languages and the implementation environments.

### 6.1.4 Current Status of AOSE Methodologies

Two broad classes of approaches to defining AOSE methodologies have been taken. The first class adapts existing object-oriented methodologies for analysis and design, whereas the second is based on agent theory. The main problem with the first approach is that object-oriented decomposition help identify objects, not agents which are more coarse grained than objects, and that many agent features are difficult to model in purely object-oriented terms.

There are already many agent-oriented analysis and design methodologies. However, most of them are incomplete and there is little consensus among them. Furthermore, there is a lot of divergence in implementation toolkits that have their own different sets of concepts. This makes the transition from an implementation independent detailed design specification to an implementation complex.

### 6.2 MESSAGE Evaluation

Evaluation of MESSAGE methodology was performed as part of the Case studies task of the project. Following a selection process, the case studies were conducted in two phases, Analysis) and Development, i.e. Design and Implementation. An assessment of the methodology based on this experience was then performed. Two case study scenarios were selected for Analysis:

- **UPA4T (Universal Personal Assistant for Travel)** - a task-specific instantiation of a generic personal assistant application
- **ACSOSS (Adaptive Customer Service OSS)** - a decentralised OSS application performing end-to-end co-ordination within a customer service business process.

For practical reasons, only UPA4T was taken forward to Design, and only selected features of this were implemented. In addition feedback was obtained from work within the LEAP EU 5th Framework Project [Lea01]. LEAP applied aspects of the MESSAGE methodology to an application supporting a mobile workforce.

Due to delays in the MESSAGE project, only a partial methodology (documented in Deliverable 1 and primarily covering Analysis) was available for direct evaluation via the case studies. A preliminary evaluation concluded that this showed promise, but the following areas in particular needed attention:

- A clearer and more consistent semantics was needed for the basic entity and relationship concepts in the modelling language, and particularly those used in the diagrams
• A graphical notation distinguishing visually the fundamental MESSAGE entities and relationships from the default UML ones was needed, plus distinct diagram types focussing on particular aspects of the analysis model.

• A clearer process model and more specific guidelines were required to give practical assistance to the analyst.

All of these were addressed in producing the final version of the methodology. Some of the enhancements were available to the LEAP project, and the feedback was positive on the basic set of concepts, the graphical notation and diagrams allowing for the fact that the notation was still fluid at that stage. The basic concepts and notation have also been discussed with James Odell[4] who is a leading figure in object- and agent-oriented methodologies. Odell viewed them as a useful contribution to the field. Limited evaluation exercises on the final analysis method confirm that it is significantly improved with respect to the above-mentioned points.

The design method was defined in parallel with the Development case study, so that the case study provided source material for the methodology rather than being a test of it. Consequently, no explicit evaluation of the design method has been performed. A subjective assessment based on reading the final methodology is that the design method is at a similar stage of maturity as the initial version of the analysis method. There are many useful ideas present derived from the experience of the project partners, but they need to be marshalled so as to tell a consistent and coherent story. Much of the design method represents a refinement of the corresponding elements of the analysis method. No additional notation is introduced: a mixture of UML diagrams, MESSAGE analysis notation and ad hoc notations are used.

A problem faced by all general agent-oriented methodologies is the wide variety of co-ordination/control models and internal architectures used for agents. It is difficult to give explicit guidance or provide a useful modelling language and notation without the context provided by one or other of these models. This is true to some extent during analysis, but the problem is more severe at design. Probably the best solution would be to define a family of abstract design patterns covering the more popular styles of agent. The design method would then have a generic part culminating in selection of an appropriate combination of patterns, followed by alternative notations and process pathways specific to the selected patterns. The MESSAGE design method goes part of the way towards this by describing separate guidelines for each of two design philosophies. They are illustrated by worked examples from the UPA4T case study. These contain some good advice from designers with considerable experience of agent-based systems, and are very instructive provided the designer is able to abstract the advice and apply it intelligently to his/her own problem. At a general level, the approaches typify two main families of agent-oriented design. However, much variation exists within these families at a detailed level.

The participants in the design case studies regarded the following as important features of MESSAGE:

• MESSAGE provides a set of views of the system and a taxonomy of concepts that help the developer define the Multi-Agent System. The concepts proved to be a sufficiently complete set of construction primitives for the case study problems. Using the views of the systems as building patterns helps developers obtain a complete specification of the MAS.

• MESSAGE provides architectural patterns based on the agent concept. These patterns are useful in both analysis and design. In analysis, for example, using the pattern that defines an agent, an analyst knows that he/she has to identify goals, tasks, and roles. In design, agents identified in analysis are transformed into computational entities through a set of design activities organised in a workflow. Examples of design activities are agent interface definition, communication refinements, agent knowledge definition, design of resources, etc. Guidelines helping developers with design choices are also provided. These include the choice between cognitive architecture and reactive architecture for agents, and the choice of communication paradigms among agents and other computational entities such as agents to agents, agents to resources and agent to users.

4 http://www.jamesodell.com/
MESSAGE is a model-driven approach. The system itself is described as an organisation generated by instantiating the Organisation concept from the meta-model. The main computational entities of the system are the agents, which are described using agent architectures. In making the transition from analysis to design, the abstract domain-oriented qualities of the organisation are realised through e.g. the knowledge, beliefs and behaviour of the computational agents. For example, the workflow structure of an organisation might be realised through rules in each agent that send notification messages when a task is completed, and other rules that cause the agent to begin executing a task when a certain type of message is received provided other conditions are satisfied. This approach was felt useful in the case studies.

The experience gained developing the prototypes shows that MESSAGE concepts and notations are easily assimilated into current engineering practice. It was possible to extend UML and to use its extension mechanisms to provide partial tool support [Cai01b] for MESSAGE.

The high level approach followed in the design enhances reusability since it does not consider domain specific situations. When possible, the proposed design applies design patterns and abstractions of existing solutions for domain specific problems, like using TINA-like session entities to handle communications between agents.

The following specific points were highlighted for consideration in future enhancements of the Design methodology:

- Clearer guidelines are needed regarding InteractionProtocols, and whether they are best considered during Analysis, Design, or both.
- The dynamic assignment of roles, services and tasks to agents during run-time is not currently covered explicitly.
- It would be useful to have some graphical diagram support for the definition of messages. When supported by a case tool these diagrams, combined with the ontology, would facilitate the generation of message examples in an effective way.
- In one exercise a technique for Protocol definition was used whereby the atomic exchanges could be realised either by messages (when they are performed by different roles) or by transactions (when they are performed by the same role). This technique brought several benefits, and should be further explored in future enhancements:
- Although the methodology does provide advice on design within the framework of a cognitive architecture, it does not provide assistance with the design of agent cognitive features.
- Consistency checking and verification methods are needed to help the designer in detecting inconsistencies and ambiguities between analysis concepts and the corresponding design constructs.

In conclusion, MESSAGE as it stands is not a complete, mature agent-oriented methodology. The MESSAGE modelling language has focused on describing the structure of the model, i.e. by providing extended class diagrams, and has not worked much on behaviour modelling, where the project has reused the interaction protocol diagrams from A-UML. Furthermore concurrency was taken into account, but was not fully addressed in the project, e.g.; conflict resolution such as several agents trying to access a common resource simultaneously and deadlocks. Emergent behaviour, which is an important issue for agents, was not investigated in the project. This is still a research issue however. It does, however, make some significant practical contributions to the state of the art that are likely to influence on-going initiatives in this area (e.g. Agent UML [Bau99]). In particular, the graphical notation / diagram set is a practical concrete result that could be taken up

3 http://www.jamesodell.com/
widely. Furthermore, even in its current state MESSAGE provides practical value to the developer of Multi-Agent Systems.
7 Conclusions

7.1 Summary

The agent-oriented approach is well suited to developing complex and distributed systems [Jen01, Gri01]. Complex software has a large number of parts that have many interactions, and solve problems that are in general only nearly decomposable. AOSE aims to provide methods, techniques and tools to solve these problems. This report has presented the MESSAGE AOSE methodology and illustrated it on an analysis and design case study.

![Figure 29 The RUP: Phases, activities and iterations](image)

The Rational Unified Process (RUP) model [Kru00] provides a generic software engineering project lifecycle framework. A project proceeds through a number of phases (Inception, Construction, Elaboration, and Transition). Each of these phases consists of one or more iterations of a sequence of activities: Requirements, Analysis, Design, Implementation and Testing. MESSAGE focuses on the Analysis and Design activities. Both of these are modelling activities: the main output of each is a model of the system at an appropriate level of abstraction. The two main elements of the MESSAGE methodology are:

- Modelling primitives and notations for agent-oriented analysis and design
- A process model and guidelines that aid the developer in producing the analysis model from the requirements, and the design from the analysis model. Some consideration is also given to implementation of the design.

The methodology is intended for use in mainstream software engineering departments. An extended UML notation was adopted for MESSAGE because most agent-based systems can be implemented in object oriented terms, and because most software engineering departments already know and use UML. The MESSAGE modelling language extends UML by contributing agent knowledge level concepts, and diagrams with notations for viewing them. The diagrams extend UML class and activity diagrams.

The principal agent-oriented concepts used in the analysis model are: Agent, Role, Organisation, Goal, Task and Interaction plus relationships between them (see Section 2). The following overlapping views have been defined focussing on different aspects of the analysis model: Organisation, Goal/Task, Agent/Role, Interaction and Domain. Section 3 described the MESSAGE analysis process and an analysis case study using the MESSAGE/UML notation. The following
diagrams were illustrated: organisation, goal decomposition, task, delegation, workflow, interaction and domain. The use of schemas to textually describe the concepts was also illustrated. A more complete analysis model would complete the MESSAGE diagrams with existing UML notation and A-UML sequence diagrams to describe role/agent interactions.

Section 5 presented the MESSAGE/UML design process and illustrated it on a design case study. Whereas it proved possible to keep the analysis part of the methodology reasonably generic, the design process and representation depends very much which one of many possible styles of agent internal architecture and/or communication is adopted. This choice depends partly on the nature of the application as revealed by analysis, but also on the personal preference of the developer and possibly on constraints restricting the choice of implementation platform/toolkit.

There are two main threads to the MESSAGE approach to design:

- MAS organisation and Architecture driven design; for each agent: selection of the agent architecture(s) followed by design of the data and knowledge structures and means of specifying behaviour appropriate to this architecture
- A Communication-oriented thread, e.g. if a speech-act message language is used, this involves design of interaction protocols, message content language and ontology, etc.

These threads are illustrated separately via two worked examples. The first example concerns an application that requires at least some agents to perform complex reasoning, but in which there are only a few distinct inter-agent messages. The case study illustrates the use of both reactive and cognitive agent architectures. This example is driven by the first thread: choice of architecture, design of knowledge representation, reasoning behaviour, etc. The second example assumes relatively simply agents required to communicate in terms of a sophisticated speech act language, plus the availability of an agent toolkit/platform specialised to FIPA ACL messages. This example is driven by the second thread.

Finally section 6 provided an evaluation of the methodology based on experience with case studies performed as part of the MESSAGE project.

### 7.2 Recommendations on the Methodology

The methodology has been validated on two partial case studies and partially on two field trials in the LEAP project [LEA01]. Both the LEAP field trials aimed to enable greater team working amongst mobile workforces, while still addressing management concerns and operational constraints.

Based on this experience, MESSAGE/UML provides a broad enough set of abstract agent related concepts to create an analysis model using the MESSAGE/UML notation. The entity concepts that were mostly used were Role, Agent, Organisation, Service, Task, Event, Action, Resource, and Interaction. It should be noticed that using UML also allows any of the UML notation to be used during analysis.

The design part of the methodology was defined late in the project and there was much less validation work carried out, and thus less feedback provided. The design part is less mature than analysis and requires further work. Nevertheless, the design methodology, mostly based on experience within Telefonica, provides a good example of an agent design that is general enough to be applicable on a wide range of problems.

### 7.3 Open Issues

The case studies should be considered as simple agents in the sense that they did not rely much on the agent’s reasoning abilities. The case studies did not validate the capability of MESSAGE/UML to model problems and solutions where reasoning of agents is required. One important issue is the level at which these concepts should best be introduced. For example if reasoning in the form of planning is required, should the concepts of plan and rule be made available at the analysis level, the design level, or simply considered as a matter of implementation of tasks that are defined in analysis and design?
Although the concurrent aspects of agent design were not dealt with explicitly in MESSAGE, it is assumed that the UML mechanisms will be sufficient. However, these aspects need to be investigated further.

Issues concerning the dynamic behaviour of multi-agent systems need to be studied. Agents’ behaviours depend strongly on internal state parameters (beliefs, goals, etc) that change over time as a result of interactions with other agents. Thus the behaviour of a non-trivial system of interacting agents is difficult to predict. Better means of specifying and checking dynamic behaviour of multi-agent systems are required.

### 7.4 Proposals for Further Work

As mentioned in section 6 MESSAGE as it stands is not a complete, mature agent-oriented methodology. More work on MESSAGE itself is required to define the semantics of the modelling language more formally and in integrating it with UML. Issues such as consistency of analysis and design models, have not been addressed, and need more work. Most of the MESSAGE project effort was spent on analysis, and design was only addressed late in the project. The transition from analysis to design and architectural modelling with UML were only partially addressed. Broader experience with agent design is needed to have a mature design methodology, and there was not enough time on this project to acquire it. One of the MESSAGE partners (BT) has been working on techniques for specifying agents’ motivations in terms of maximising utility functions rather than achieving goal states. These techniques should be integrated into the MESSAGE methodology.

Also, MESSAGE has proposed a brand new conception of Multi-Agent System organisation [Gar01], enabling engineers to work with MAS from a high level point of view. The organisation view is being explored and enhanced to reflex better issues like social constraints or group formation.

Future developments of MESSAGE should be co-ordinated with other similar projects such as A-UML. A first step would be to integrate results from MESSAGE into on-going agent modelling work such as A-UML [Ode00]. A next step would be for EURESCOM partners to participate in these standardisation efforts in order to make sure that telecom specific requirements are taken into account. This would guarantee that resulting commercial tools will be useful to EURESCOM partners interested in developing agent-based solutions.

There are many open research issues that have that have to be taken into account by modelling languages, and thus were not taken into account in this project. For example, more modelling experience is needed with the emergent behaviour of agents, and on representing reasoning related concepts with MESSAGE. Modelling systems with emergent behaviour will require the study of techniques and guidelines for verification, validation and testing. For example, Better means of specifying and checking dynamic behaviour of multi-agent systems are required.

MESSAGE has focused on the analysis and design phases of software engineering. The relation with other phases of the software lifecycle needs to be addressed. The testing and implementation phases are of important pragmatic consideration.
8 References


Annex 1. MESSAGE Notations at-a-glance

As mentioned, in order to provide a further degree of expressiveness, MESSAGE extends the UML meta-model by adding a number of elements (meta-concepts and meta-relations) to it. This section describes the notation that MESSAGE defines to graphically represent the instances of these new meta-elements in the diagrams that the developer is going to draw.

Two main criteria were followed in the definition of this notation.

- Simplicity. All symbols must be simple so that they can be drawn by hand.
- Intuitiveness. A symbol should remind the reader the concept it represents.

The reader is suggested to get familiar with the MESSAGE meta-model before reading this annex.

Symbols summary

Figure 30 and Figure 31 provide a summary of the symbols representing the MESSAGE concepts and relations respectively. It should be noticed that symbols associated to elements natively included in the UML meta-model are not mentioned here.

![MESSAGE concepts symbols](image)

![MESSAGE relations symbols](image)
Usage of relations

Implication

This relation links one or more elements that have an attribute of type State (such as a Goal whose target attribute is of type State and Task whose precondition and postcondition attributes are of type State) to a single element that has an attribute of type State.

The Implication relation is therefore directional and its semantics is that the State that is an attribute of the element pointed to by the relation is implied by the logical AND of the states that are attributes of the elements where the relation starts from.

It is therefore possible to draw “Task-Goal Implication diagrams” as in Figure 32 whose semantics are:

G1 achieved => T1 feasible
T1 successfully completed => T2 feasible
T2 successfully completed => G2 achieved

![Figure 32 Example of a "Goal-Task Implication diagram"

In particular the implication relation can conveniently be used to graphically represent goal decomposition as exemplified in Figure 33 where Goal G1 is achieved if ((G2 is achieved) AND (G3 is achieved) AND (G4 is achieved)) OR (G5 is achieved)

Assignment

This relation links an element of type AutonomousEntity to an element that has an attribute of type AutonomousEntity (e.g. a Goal whose performer attribute is of type AutonomousEntity). The semantics are such that the assignment from one AutonomousEntity to another follows the direction of the arrow.

It is recognised that there are different types of Assignment each one of which can be asserted for specific subtypes of AutonomousEntity. At the metalevel therefore we have the following Assignment structure.

---

6 A State (or Macro-state) is defined as a set of Micro-states of the world where at each point in time the world is in a single Micro-state that defines it completely. So that “John is on the chair” is a State as it defines all possible Micro-states in which John is on the chair.
At the model level the different types of Assignment are indicated by the same Assignment symbol and are distinguished indicating the particular subtype as a predefined stereotype as in Figure 35.

It has to be notice that a Service has an attribute of type AutonomousEntity that is the provider, but has no attribute “consumer” or similar. Therefore the fact that an AutonomousEntity requires or consumes a service is not expressed as an Assignment relation. UML dependency can be used for this purpose as in Figure 36 showing that Agent A1 consumes the Service S1 provided by Organisation O1.
Acquaintance

This relation links two AutonomousEntity and its semantics are such that it at least one interaction is needed between these entities.

DataFlow

This relation links a DataProsumer to an InformationEntity that is produced or consumed. This is the same relation as the ObjectFlow relation defined in UML and therefore the same symbol is used (where the arrow indicates the flow of the data).

With respect to UML, however, we modify the graphical representation of that relation slightly and in particular we introduce a condensed notation in which an InformationEntity produced by a task and required as an input to another is represented as an association class of a State-Transition relation as shown in Figure 37.

![Figure 37. DataFlow as an association class](image.png)

The condensed notation in particular allows the modeller to clearly represent situations where the output of a task depends on the transition that is followed as shown in Figure 38.

![Figure 38. Different outputs](image.png)

The extended notation on the other hand allows the modeller to represent situations where no transition is present or where an InformationEntity serves as input to more than one task, or even non consecutive tasks as shown the figure below.
Remarks

It should be noted that MESSAGE does not introduce a “precedence” relation to indicate that a Task T1 must be performed after another Task T2 has completed as in workflow diagrams. UML state-transitions are sufficient to indicate task ordering.

Similarly MESSAGE does not introduce “power relations” to indicate that e.g. an agent is the “master” of another agent as UML dependency can be used for this purpose.

Sample diagrams

This section provides a collection of sample diagrams illustrating each type of MESSAGE diagram.

Organisation diagram

Figure 39 Information flow between non-consecutive tasks

Figure 40. Example of Organisation diagram
Goal-Task Implication diagram

Figure 32 and Figure 33 already provide example of a Goal-Task Implication diagrams.

Delegation structure diagram

![Delegation structure diagram](image)

Figure 41. Example of Delegation structure diagram

Workflow diagram

![Workflow diagram](image)

Figure 42. Example of Workflow diagram
Agent diagram

Figure 43. Example of Agent diagram

Interaction diagram

Figure 44. Example of Interaction diagram

**FIPA Contract Net Interaction Protocol**

In the FIPA contract net interaction protocol, one agent takes the role of manager which wishes to have some task performed by one or more agents and further wishes to optimise a function that characterises the task in terms such as price or time to complete. The manager solicits proposals from other agents by issuing a call for proposals, which specifies the task and any conditions the manager is placing. Agents receiving the call for proposals are viewed as potential contractors and are able to generate proposals to perform the task (propose acts). The contractor’s proposal includes the preconditions that the contractor is setting out for the task such as price or duration. Alternatively the contractor may refuse to propose. Once the deadline passes the manager evaluates any received proposals, and selects agents to perform the task. The agent(s) of the selected proposal will be sent an accept-proposal act. The proposals are binding on the contractor, so that once the manager accepts the proposal, the contractor acquires a commitment to perform the task. Once the contractor has completed the task, it sends a completion message to the manager. To guard against the case where a contractor does not answer a call for proposals, the manager may potentially be left waiting indefinitely. To guard against this, the call for proposal includes a deadline by which replies should be received by the manager. Proposals received after this deadline are automatically rejected with the given reason that the proposal was late. The FIPA Contract Net interaction protocol is shown in Figure 45.

**Figure 45 FIPA Contract Net Interaction Protocol**

![Diagram of FIPA Contract Net Interaction Protocol]
Annex 3: Detailed discussion of the analysis process

This annex follows on from Section 3, which gives an overview of the analysis process.

A3.1 Building the Level 0 Model

Level 0 is concerned with defining the system to be developed with respect to its stakeholders and environment.

![Diagram of Level 0 Model]

**Figure 46.**

A3.1.1 Organisation View

This is adapted from the Rich Picture Diagram technique used in SSM (Soft Systems Methodology). The first step is to prepare a provisional list of external entities that are significant to the requirements or operation of the system, including:

- (categories of) user
• other stakeholders
• agents with which the system may need to interact
• resources (e.g. databases, computational resources, etc.) that the system uses, controls, or receives input from

These can be obtained by analysing the requirements document or through discussion with the customer.

The next step is to produce a high-level diagram showing the relationships among the external entities and the system. This could be prepared jointly with the customer, or prepared initially by the analyst, then revised through discussion with the customer. Begin by placing an Organisation symbol in the centre of the diagram and placing the other entities around it. Use the standard MESSAGE symbol set to distinguish between major categories of entity (Human/Software Agents, Organisations, Resources, etc.), but do not agonise over this, you can change your mind later. Add textual, colour, graphical decoration to the symbols as appropriate to aid intuitive understanding.

Sketch major relationships between entities (re-arranging symbols as required for clarity). At this stage, just indicate the basic categories of relationship through use of the appropriate type of arrow according to MESSAGE notation (see appendix). Where there is no special MESSAGE notation, use a UML association or dependency symbol as appropriate, with a label added for clarity of meaning:

• Acquaintanceship (i.e. the entities interact) - bi-directional.
• Power' - directional. Used to indicate one entity 'is the boss of', controls, influences, is served by another. A textual label appropriate to the application can be used to indicate the strength and variant of meaning. In the case of a power relationship between two AutonomousEntities, such a relationship means that the sub-ordinate tries to satisfy the wishes/desires of the superior. The strength of the relationship influences the extent to which the sub-ordinate is willing to sacrifice its self-interest in order to please its superior. For now, don't worry about such subtleties, just label the arrow using the customer's terminology.
• Peer to peer organisational relationships: These connect entities that are dependent on each other in some way, without implying a power relationship. The most common example is some form of customer-provider relationship indicating that the customer entity (potentially) uses a service provided by the provider. For clarity, the UML association class notation can be used to connect the service in question explicitly to the instance of the relationship. Customer-provider is directional, but not all peer-to-peer relationships are.
• UML 'contains', 'specialises' and dependency arrows
• Any other categories of relationships can be indicated by UML associations with suitable labels.

Remember that the purpose of this diagram is to give a clear, intuitive overview to capture a shared understanding between the analyst and customer.

A3.1.2 Start accumulating the Domain Model/View

This part of the specification captures domain concepts that are important to the system and customer. The process is essentially similar to 'classical' knowledge engineering. The knowledge captured will be used:

• to help the analyst understand the problem and to promote effective communication between analyst and customer;

And later:

• as a basis for the inter-agent communication ontology
• to derive internal knowledge representations and knowledge base content for knowledge-based agents.
The Domain View is accumulated throughout the analysis process as and when required. No special notation is introduced. The analyst is encouraged to use the appropriate UML and MESSAGE notation to express the concepts being captured.

A3.1.3 Identification of services and workflows

The starting point for this is the set of acquaintanceship (and also customer-provider) relations identified earlier. The acquaintance arrows simply indicated that the connected entities interact in some way. Here, we start to look more closely at the interaction of the entities. There are three main types of reason for interaction between:

1. co-ordination - basically this is to bring into alignment the mental models of the entities (e.g. to make beliefs or shared plans consistent)
2. supply of information - one entity requires information from the other
3. one entity requires the other to perform a task or achieve a goal.

In this step, the acquaintanceship relations identified earlier are subdivided to obtain Interactions that can each be classified in one of the categories with a simple text label identifying the shared object of co-ordination, the information supplied or goal achieved. For simplicity, let us refer to this labelled item as a service. In the case of 2 and 3 the direction of flow should also be indicated. The analyst will often be faced with a choice of how to classify the interactions. This is a judgement call, but should take into account clarity of understanding in the context of the problem domain.

Dependencies between the services should be identified to obtain workflows through the system (i.e. chains of tasks performed by different entities, flows of information, requirements for multi-way co-ordination).

A3.1.4 Identification of purpose

(This could be done in parallel with identification of services and workflows).

In the same way that services are derived from acquaintanceship relations, the purpose (primary guiding goal) of an AutonomousEntity is derived from the power relationships. The services define an entity in terms of what it does (capability); the purpose defines it in terms of why it acts (motivation). Typically the motivation of an agent or MAS is defined relative to a 'real' stakeholder, e.g. a personal agent's motivation is to satisfy the needs its owner in respect of the personal agent's functional capability. This step involves:

- Analysing/defining the motivations (purpose) of the stakeholders (in so far as they are relevant to the system being developed)
- Relating the purpose of the system being developed to the purposes of the stakeholders to which it is sub-ordinate
- Modifying the purpose of the system to give a rationale for resolving conflicts of interest among stakeholders.

For example, consider the UPA4T scenario. MyFriend4T has two stakeholders: the traveller role and the travel operator role. The traveller's purpose is to obtain cheap, convenient travel arrangements that satisfy his/her travel requirements. The travel operator's purpose is to make as much profit as possible by supplying travel services. MyFriend4T's purpose can be defined by composing these, then resolving the conflict of interest over economy.

The consistency of services and purpose should be checked and the model revised if appropriate.

In performing this step and the previous one, it is highly likely that domain specific concepts and information object classes will have been identified and added to the domain view.

---

7 In fact these are all inter-related, and it is debatable whether there should be 1, 2 or 3 categories.
A3.1.5 System details

At this point the main external feature of the system to be developed have been identified:

• its motivation (purpose)
• Its capability (services)
• Its requirements for services from others
• The resources available to it and the acquaintances it provides services to and receives services from.

The next step is to ensure that these are tied together in coherent way. In particular, the system:

• Must be capable of providing the required services. Its knowledge must include one or more tasks encapsulating the procedural knowledge required to provide the services. The system should be able to perform the task entirely by itself, or else to perform some sub-tasks itself and have others available as services provided by other entities. It must have access to resources necessary for performing the tasks.

and the system's purpose must provide a rationale for:

• decisions on when, and on what terms, to agree to requests for services;
• decisions on proactive initiatives
• runtime decisions regarding how a service is provided

The system must have access to information enabling it to make these decisions.

At Level 0, high level descriptions of this information are documented in a short form of an Organisation Schema and optionally a 'focus' diagram.

A3.1.5.1 Goal analysis

Although the purpose of the system was captured earlier, it is possible that some further analysis is required here to express the purpose in a more concrete form. 'Goal decomposition' provides techniques for doing this. The result is a number of trees, the roots of which are high-level goals, and the leaf nodes are goals that can be expressed clearly in terms of e.g. constraints on the states of entities in the model.

A3.1.5.2 Task knowledge

For each service, there must be at least one task that achieves the goal associated with the service. Tasks define means of transforming one state into another, and are clearly domain-dependent. It is possible that some tasks that are possible in the domain have already been identified as part of the Domain View. It is also likely that some preliminary identification of relevant tasks has been done in defining the workflows, above.

Suppose we start with a service that the system must perform as identified above. Has a Task already been identified that enables that service to be provided? If so, let us hypothesise that the system will use it to provide the service. If not, the analyst must consult with the customer/user/domain expert to identify an appropriate task. Associated with the task will be preconditions, and resources required.

At this stage it is not necessary to describe how the Task is to be performed. The important thing is to identify the high level Tasks and to consider the feasibility of the system to perform the task using the resources, information, acquaintances, etc. identified.

Tasks should also be identified that can be used to achieve the various leaf goals identified above. The analyst needs to exercise judgement (on a goal-by-goal basis) over whether to introduce tasks explicitly here or delay until later. In some cases it will be more natural to work primarily with a goal, the associated task being implicitly derived from it (i.e. 'a task that achieves Goal A'). In other
cases the reverse may be true; it may be more natural to work primarily with the task (i.e. the goal is to complete Task A successfully). At other times working with both goal and task may be useful.

A3.1.5.3 Interaction details

The basic interactions have already been identified above. Here the details are decided and documented.

- If necessary, sub-divide the interactions already identified and add others to rectify omissions
- Identify the initiators of the interactions
- Clarify and document the goal of the interaction. Note this is a goal that shared by all participants in the interaction
- Clarify and document any information objects, etc., associated with the interaction (e.g. terms that must be agreed before the service takes place).

This information is captured in an Interaction Schema (one for each Interaction).

A3.1.6 Check for completeness and consistency

This is an important step, but it is difficult to give concrete guidance at a generic level. Consistency is always important, but what constitutes completeness depends on the project and also the stage of the project (i.e. elaboration of some of the model elements may be deferred to later iterations). Furthermore, in some cases it is fine for a model element to remain implicit.

As a minimum, the analyst should set aside some time to review the model as objectively as possible. It would be preferable to involve another person or a review team in the process. An example of good practice would be for the analyst to 'walk through' the model with the review team, followed by discussions to agree any actions required.
A3.2 Building the Level 1 Model

The Level 1 model takes as a starting point, the system definition obtained at Level 0. At Level 0 the system being developed was treated as a unitary entity. At Level 1 it is decomposed into an organisation of interacting agents. If a further level of detailed definition is required, one or more Agents can be redefined as Organisations. The process can then be applied to these organisations to obtain a Level 2 model, and so on. For many MAS, Level 1 will be sufficient.

Many of the guidelines described above regarding building the Level 0 model apply also to Level 1 (and subsequent levels if applicable). The main differences are that:

Figure 47.
• at Level 0, the entities and their attributes are largely determined from the requirements and 
  the environment in which the system will operate. At Level 1 (and below), they are derived 
  from the level above through the skill and judgement of the analyst.

• at the lowest analysis level, the detailed descriptions of the various elements need to be 
  provided, whereas at Level 0 (and intermediate levels) this can be deferred. The precision 
  required of the detailed description various from project to project, however.

The paragraphs below concentrate mainly on these points of difference.

A3.2.1 Goals and Tasks

The aim of this activity is to develop trees of Tasks and Goals connected by logical (and temporal) 
relationships describing the repertoire of behavioural knowledge of the system. The roots of the 
trees are Goals based on the purpose and services of the system. The leaf nodes are Tasks whose 
preconditions correspond to the space of States in which the system is expected to achieve those 
Goals. This section describes a top-down procedure for doing this.

We take as starting points a) the purpose, and b) the services of the multi-agents system as 
identified at level 0, and identify 'root Goals' from them. Clearly the system must be able to achieve 
a Goal corresponding to providing each of the services it offers. One or more persistent Goals 
should also be identifiable from the purpose. In general these two types of Goal will be related. For 
example, the purpose will provide criteria for the system to decide when to offer / agree to provide 
a service.

We then proceed to decompose these root Goals. Three basic operations are used:
• decomposing a Goal into a set of sub-Goals
• identifying one or more Tasks that can be used to achieve the Goals
• substitution of a Task by a set of more detailed Tasks and Goals able to achieve the same 
  Goal.

This process continues recursively until the granularity is judged to be sufficiently fine, and the 
trees provide pathways to root Goals from all required States.

A3.2.2 Identification of Agents, Interactions and Relationships (Organisational View)

The objective here is to identify a set of interacting and related agents (or roles) collectively able to 
fulfill the system's purpose and provide its Services. The following strategies can be used 
individually or in combination:

1. identify Agents based on coherent sets of Goals and Tasks. Clearly this is dependent on having 
   performed Goal/Task decomposition first. Interactions

2. identify Agents based on correspondance to entities outside the system. For example Agents 
   serving particular users / classes of user / user roles, Agents Managing particular resources, etc.

3. identify agents based on providing particular services.

In the case of 2 and 3, in addition to the Agents identified directly, it is likely that Agents providing 
core functionality will be required.

For each agent/role, basic level information should be filled in on its schema (identity, purpose, 
services).

Note that at this stage it may not be clear whether Agent or Role should be employed. This is not a 
significant problem, however, since the basic level information is the same for each. Re-
classification will be possible later, anyway.
A3.2.2.1 Modified Use Case Analysis

This is presented here as a useful technique at this stage. Its purpose is to check and refine identification of agents, and help establish the interactions and dependencies among them. In some situations, when the initial choice of Agents is difficult, the analyst may prefer to use it for initial identification also.

Use Case Analysis involves examining a representative selection of usage scenarios to obtain system characteristics appropriate to those specific circumstances, then generalising to obtain characteristics of a full system. This general approach is useful for agent-oriented analysis, but its application must be adapted. The following approach is recommended:

Pick an instance of Interaction identified at Level 0. Identify the most appropriate Agent/Role to handle this Interaction. Is this Agent/Role capable of carrying out the Interaction and providing any services associated with it unaided? If not, ensure interactions are in place with Agents/Roles that provide the additional support. Repeat the question of these agents until a set of interacting agents capable of completing the Interaction and providing the services is achieved. This may require introduction of additional Agents/Roles.

Repeat the process described in the last paragraph on another Interaction instance. If this results results in additional Agents/Roles or Interactions, then the set of agents should be reviewed to see if rationalisation by merging Agents/Roles, or splitting 'bottleneck' Agents/Roles would be an improvement.

This continues until a stable set of Agents/Roles is achieved.

A3.2.3 Reconciliation

This step involves making identifications and associations between entities derived in the organisation view and ones derived in the Goal/Task view. In particular:

- identification of the purpose attributes of Agents and Roles with Goals (Role assignment);
- identification of Services provided by specific Agents/Roles with specific Tasks;
- identification of acquaintance relationships with Interactions. Identification of Interactions with requirements for co-ordination between Tasks;
- association of Goals and Tasks with the Agents/Roles that need to know how to achieve/perform them. Note that (clearly) an Agent must know how to achieve the Goal that is its purpose, and to perform at least one Task corresponding to each of its Services, but in general it will need to know about other (tactical) Goals and Tasks. Note also that several agents may need to know how to achieve a given Goal or perform a given Task;
- association of domain knowledge with Agents/Roles that need to know about it (either to perform their own tasks or to communicate with others).

Depending on the route taken, some of the identifications may have already been made provisionally.

A3.2.4 Introduce co-ordination/control mechanisms

At this stage, end-end workflows should be constructed to check that the MAS is able to deliver the services identified at Level 0. If the modified Use Case analysis was performed earlier as part of Agent/Role identification, then the same scenarios can be used here. If it was not performed earlier, then the exercise can be performed here. This is an opportunity to identify missing elements in the workflows and to rationalise and simplify, e.g. by combining or splitting some Roles.

At the same time, requirements for (on the fly) task allocation, co-ordination, control, conflict resolution, etc. should be considered. Additional Agents/Roles can be introduced specifically to handle any such requirements. Alternatively, the existing Agent/Role specifications can be modified so that the requirements are handled collectively. The choice depends on whether analyst judges a centralised/hierarchical or decentralised organisation to be more appropriate.
A3.2.5 Develop domain model

Following on from Level 0, the domain view is elaborated as required in parallel with other activities.

A3.2.6 Detailed descriptions

At this stage the detailed component of the individual Agent/Role and Interaction specifications is completed. In the case of the Agent/Role specifications this involves filling in the schemata. The level and formality of the description will depend on the system being developed. Partly this description involves collecting together references to elements of the model that are relevant to an individual Agent/Role. However it also involves adding sufficient description of the internal structure (including mental state entities), knowledge, and behaviour for the schema to be used as a specification for an Agent or Agent class.

In the case of Interactions it involves completion of each interaction specification. It could also include description of the corresponding InteractionProtocols, though depending on the project, this could be deferred the design stage.

A3.2.7 Check consistency

At this stage the internal consistency of the Level 1 model should be checked, followed by consistency of Level 1 with Level 0.

A3.2.8 Synthesis of behaviour

The bottom-up synthesis of behaviour begins.