

Project P907

**MESSAGE: Methodology for Engineering Systems of  
Software Agents**

Deliverable 1

**Initial Methodology**

**Suggested readers:**

Software engineering personnel at the EURESCOM shareholder companies.

**July 2000**



EURESCOM PARTICIPANTS in Project P907 are:

- BT
- FT
- IT
- PT
- RB
- TE
- TI

#### **In case of Confidential Deliverables**

This document contains material which is copyright of certain EURESCOM PARTICIPANTS and may not be reproduced or copied without permission. The information contained in this document is the proprietary confidential information of certain EURESCOM PARTICIPANTS and may not be disclosed except in accordance with Section 5 of the EURESCOM's general conditions of the contract.

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

Neither the PARTICIPANTS nor EURESCOM warrant that the information contained in this document 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 the information.

#### **In case of Public Deliverables**

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.

This document has been approved by EURESCOM Board of Governors for distribution to all EURESCOM Shareholders.

## 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 that would support this methodology. This should encompass the whole software life cycle, analysis, design, implementation, testing, installation, and reiteration. The methodology should be centered 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 the first Deliverable of four. It describes the Initial Methodology for Agent-Oriented Developments. The remaining deliverables will be:

- D2 Agent Technology Applicability Guidelines
- D3 Methodology for Agent-Oriented Development (final)
- D4 Recommendations on Supporting Tools

The Project started its activities on June 1999 and will be finished in May 2001. The Project is led by Wim Coulier (Belgacom). BT, FT, IT, PT, RB, TE and TI are participating in the Project.

Heinz Brüggemann  
Project Supervisor

## Executive Summary

This document represents the first deliverable (D1) from the MESSAGE project P907. The document contains an initial version of a methodology for developing Agent Oriented Software, in particular Multi-Agent Systems for Telecommunications applications. The methodology is intended to be of benefit to software engineering personnel in EURESCOM client organisations. This deliverable has concentrated on the Analysis activity due to its central importance within the development process. A later deliverable will make recommendations on how the methodology handles later activities such as Design, Implementation and Testing.

The document presents a set of five analysis models, which can be used by analysts to capture different aspects of an agent system. The models are described in terms of sets of interrelated concepts. The five models are:

**Organisation Model:** This captures the overall structure of the system. It specifies the number and types of agents within the system and how they are related in terms of "power relationships".

**Goal/Task Model:** This captures what the Agent System and constituent agents *do* in terms of the Goals that they work to attain and the tasks they must accomplish in order to do so. The model also captures the way that Goals and Tasks of the system as a whole are related to Goals and Tasks assigned to specific agents within the Multi-Agent System.

**Agent Model:** This model contains a detailed and comprehensive description of each individual Agent and Role within the MAS. This description provides an internal view including the Agent's Goals and the services they provide. This contrasts with the external perspective provided by the Organisation Model.

**The Domain (Information) Model:** This model acts as a repository of information (both entities and relations) concerning the problem domain.

**The Interaction Model:** This model is concerned with capturing the manner in which agents communicate with one another. It specifies interactions from both a high-level and low-level perspective.

The document also recommends notations for representing the model components in graphical or tabular form. Simple example models are also built in order to illustrate practical use of the notations.

Finally, a modelling process is recommended that describes the process by which the set of analysis models can be built. It describes the order in which the models are created and how information in one model acts as input to building of other models. It also describes how the models are progressively decomposed, starting with a Level 0 model in which the MAS is viewed as a single agent and progressing to Level 1 and so on, by successively decomposing each agent into component agents.

## **List of Authors**

Collated by: Richard Evans, TI from contributions from all project partners.

## Table of Contents

<i>Preface</i> .....	2
<i>Executive Summary</i> .....	3
<i>List of Authors</i> .....	4
<i>Table of Contents</i> .....	5
<i>Abbreviations</i> .....	9
<b>1 Overview</b> .....	10
1.1 <i>Introduction</i> .....	10
1.1.1 <i>The MESSAGE project (P907)</i> .....	10
1.1.2 <i>What is this document?</i> .....	10
1.1.3 <i>Who should read this document?</i> .....	10
1.2 <i>Motivation</i> .....	11
1.3 <i>Guidelines for use of the Agent Oriented approach</i> .....	12
1.3.1 <i>The approach</i> .....	12
1.3.2 <i>The agent definition</i> .....	13
1.3.3 <i>The Guidelines</i> .....	13
1.4 <i>Overview of the MESSAGE methodology</i> .....	14
1.4.1 <i>The Software Life Cycle and Methodologies</i> .....	14
1.4.2 <i>The Rational Unified Process model for Software Development (RUP)</i> 15	
1.4.3 <i>Requirements</i> .....	18
1.4.4 <i>Multi-Agent System Analysis</i> .....	18
1.4.5 <i>Multi-Agent System Design</i> .....	18
1.4.6 <i>MESSAGE and other Stages of the Software Life Cycle</i> .....	19
1.4.6.1 <i>Implementation</i> .....	19
1.4.6.2 <i>Testing</i> .....	20
1.4.6.3 <i>Deployment</i> .....	20
1.4.7 <i>Conclusion</i> .....	21
<b>2 Overview of Analysis Models</b> .....	22
2.1 <i>The MESSAGE modelling language and UML</i> .....	22
2.2 <i>Example scenario</i> .....	22
2.3 <i>Levels of resolution</i> .....	23

2.4	<i>The analysis models</i>	23
3	<i>Organisation Model</i>	26
3.1	<i>Purpose of Model</i>	26
3.2	<i>Structure of Model</i>	26
3.3	<i>Concepts</i>	27
3.4	<i>Notations</i>	28
3.4.1	<i>Structural Description of the Organization</i>	28
3.4.2	<i>Behavioural description of the organisation</i>	29
3.5	<i>Example</i>	29
3.6	<i>Guidelines</i>	32
4	<i>Goal/Task Model</i>	34
4.1	<i>Purpose of Model</i>	34
4.2	<i>Structure of Model</i>	34
4.3	<i>Concepts</i>	35
4.3.1	<i>Situation Concept</i>	35
4.3.2	<i>Goal Concept</i>	36
4.3.3	<i>Action Concept</i>	36
4.3.4	<i>Task Concept</i>	36
4.3.5	<i>DecomposeGoal Relationship</i>	37
4.3.6	<i>DecomposeTask Relationship</i>	37
4.3.7	<i>Commitment Relationship</i>	37
4.3.8	<i>Satisfaction Relationship</i>	37
4.4	<i>Notations and Example</i>	37
4.4.1	<i>Goal Graph Definition</i>	38
	<i>Goal</i>	38
	<i>Commitment Assignment</i>	39
	<i>Goal</i>	39
4.4.3	<i>Task Decomposition</i>	40
	<i>Action</i>	41
4.5	<i>Guidelines</i>	41
4.5.1	<i>When to Switch from Goal to Task Decomposition</i>	41
4.5.2	<i>When to Stop Goal Decomposition</i>	41
4.5.3	<i>When to stop Task Decomposition</i>	41
5	<i>Agent Model</i>	42

<b>5.1 Purpose of Model</b> .....	42
<b>5.2 Structure of Model</b> .....	43
<b>5.2.1 Agent</b> .....	43
<b>5.2.1.1 Identity</b> .....	43
<b>5.2.1.2 Roles</b> .....	43
<b>5.2.1.3 Interactions with environment</b> .....	43
<b>5.2.1.4 Mental state and behaviour</b> .....	44
<b>5.2.2 Role</b> .....	44
<b>5.2.2.1 Identity</b> .....	44
<b>5.2.2.2 Agent requirements</b> .....	44
<b>5.2.2.3 Interactions with environment</b> .....	44
<b>5.2.2.4 Mental state and behaviour</b> .....	45
<b>5.3 Concepts</b> .....	45
<b>5.4 Notations</b> .....	47
<b>5.5 Example</b> .....	48
Scenario outline: .....	48
<b>5.6 Guidelines</b> .....	52
<b>6 Domain (Information) Model</b> .....	54
<b>6.1 Purpose of Model</b> .....	54
<b>6.2 Structure of Model</b> .....	54
<b>6.3 Concepts</b> .....	54
<b>6.4 Notations</b> .....	55
<b>6.5 Example</b> .....	55
<b>6.6 Guidelines</b> .....	56
<b>7 Interaction Model</b> .....	58
<b>7.1 Purpose of Model</b> .....	58
<b>7.2 Structure of Model</b> .....	58
<b>7.3 Concepts</b> .....	58
<b>7.3.1 Interaction</b> .....	59
<b>7.3.1.1 Associations</b> .....	59
<b>7.3.2 Interaction Protocol</b> .....	59
<b>7.3.2.1 Associations</b> .....	59
<b>7.3.3 Interaction Role</b> .....	60
<b>7.4 Notations</b> .....	60

<b>7.5 Example.....</b>	<b>60</b>
<b>7.6 Guidelines.....</b>	<b>62</b>
<b>8 Workflows.....</b>	<b>63</b>
<b>8.1 Overview of Modelling Process .....</b>	<b>64</b>
<b>8.2 Inputs to the Modelling Process .....</b>	<b>64</b>
<b>8.3 Building the Level 0 models .....</b>	<b>64</b>
<b>8.4 Building the Level 1 Models .....</b>	<b>66</b>
<b>9 Guidelines.....</b>	<b>67</b>
<b>10 Conclusions .....</b>	<b>69</b>
<b>11 References .....</b>	<b>70</b>
<b>12 Glossary.....</b>	<b>71</b>

## Abbreviations

ACL	Agent communication language
ADL	Architectural Description Language
AI	Artificial Intelligence
AM	Agent Model
AO	Agent-oriented
AOSE	Agent-oriented software engineering
CORBA	Common Object Request Broker Architecture
DM	Domain (Information) Model
DSDM	Dynamic Systems Development Method, a RAD methodology
FIPA	Foundation for Intelligent Physical Agents
GM	Goal/Task Model
IM	Interaction Model
ISDN	Integrated Serviced Digital Network
KADS	Knowledge and Analysis Design Support
KBS	Knowledge-based system
LAN	Local Area Network
MAS	Multi-agent system
MESSAGE	Methodology for Engineering Systems of Software Agents
MOF	Meta-Object Facility
OCL	Object Constraint Language (part of UML)
OM	Organisation Model
OO	Object-oriented
PS	Project Supervisor
RAD	Rapid Application Development
RMI	Remote method invocation
RUP	Rational Unified Process
SE	Software engineering
SoR	Statement of Requirements
UML	Unified Modelling Language – an object-oriented modelling language used in several OO methodologies

# 1 Overview

## 1.1 Introduction

### 1.1.1 The MESSAGE project (P907)

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 applications realised using agents are becoming more complex over time. But the design process of agent-based applications differs greatly from the design of object-oriented applications. Most agent-based systems use object-oriented languages like Java, but common methodologies for the development of object-oriented are inadequate for the design of agent based applications.

The main objectives of P907 are:

- To define a suitable methodology for the development of agent based applications in the telecommunications domain
- To identify and recommend tools which support this methodology
- To define guidelines for the identification of application areas where agent technology is better suited than other technologies, e.g. OOT.

P907 will produce the following set of deliverables.

D1 - Methodology for Agent-Oriented Development – Initial version (this document)

D2 - Agent Technology Applicability Guidelines

D3 - Methodology for Agent-Oriented Development – Final version

D4 - Recommendations on Supporting Tools

### 1.1.2 What is this document?

This document is the first deliverable from the MESSAGE project P907. It presents an initial version of an Agent Oriented Software Engineering methodology, developed in particular for the needs of the telecommunications industry. A later deliverable (D3) will present a more complete and detailed methodological description. The methodology is presented in stages so as to provide an opportunity for feedback from shareholders during the lifetime of the project.

The document provides a set of Analysis models for Agent systems. It also describes how to go about building these Analysis models. The document does not contain a detailed description of how to carry out later activities in the development process, i.e. the design, implementation and testing activities. These issues will be addressed in the deliverable D3.

### 1.1.3 Who should read this document?

The document should be read by software engineering personnel at the EURESCOM shareholder companies. Readers are assumed to be familiar with basic UML notation.

## 1.2 Motivation

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. For some examples, see the projects surveyed in PIR3.1. 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). While there are many variants of OO methodology, a consensus is emerging on notation – UML (Unified Modelling Language [1]). It provides a uniform and consistent visual formalism in which to express the results of different object oriented methodologies proposed during the 1990s. UML can be considered a *de facto* standard for defining the most significant deliverables in software development. Complementing UML is the Unified Process (RUP) for software development proposed by Rational [11]. This is a broad-spectrum iterative software lifecycle model. It identifies two organisational dimensions to the software development process: time (lifecycle phases), and content (models and other artifacts). This is in contrast to the traditional linear 'waterfall' model in which the phases are tied to models. The waterfall model can be seen as one extreme end of the spectrum covered by RUP, and rapid application development (RAD) models such as Dynamic Systems Development Method (DSDM) at the other.

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. If an OO methodology is employed for analysis and design, then the result will be an OO system, even if an agent-oriented toolkit is used to implement the system. 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). See PIR3.3 for an analysis of the extent to which UML and other existing SE methods are useful for the purposes of MESSAGE.

MESSAGE is not alone in working towards an AOSE methodology. AOSE projects can be divided into two main types:

1. Those that aim to apply conventional SE methods to AO systems, modifying them when necessary. Notable amongst these is the work of Odell *et al* on Agent UML (AUML) [10]. The main achievement of this work to date is a means of representing agent interaction protocols. This forms the basis of a submission to OMG [4]. This additional notation is useful, and indeed it is adopted within the MESSAGE methodology. However, specifying an object's behaviour in terms of interaction protocols does not make it an agent. An agent-oriented methodology by definition must have the concept of *agent* at its centre.
2. Those that start from an agent theory, AI or KBS, perspective with the aim of developing a new AO method covering analysis and/or design. Much of this

work has an academic flavour, is insufficiently complete or mature for practical use, or requires skills beyond those to be expected of the average software engineer. Nevertheless, several of these projects contain useful ideas. See PIR3.1 for an analysis of several such projects.

The AOSE methodology projects surveyed in PIR3.1 exhibit a number of similarities. The analysis and design models are divided into a number of sub-models emphasising particular aspects, though the number and scope of the sub-models varies considerably. MAS-CommonKADS [9] and Gaia [3] (referred to as WJK in PIR3.1 after the authors' initials) represent the two ends of this spectrum. MAS-CommonKADS is derived from the KADS knowledge engineering methodology [7]. It has six analysis models, which are refined and a further 3 model are added in design. This set of models is quite comprehensive, but there is no unifying semantic framework or notation. Gaia, on the other hand, has two analysis models and three design models. The Gaia analysis models (roles model and interactions model) are based on well-defined concepts, but represent only a sub-set of the concepts required for AO analysis. The Gaia design models are quite sketchy, but Gaia is primarily an analysis method and the authors envisage object-oriented methods being used for detailed design. A third project influencing MESSAGE is KAOS, a agent-oriented requirements engineering method in which requirements analysis is driven by a goal-oriented strategy [6].

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.

### 1.3 Guidelines for use of the Agent Oriented approach

This section provides initial indications and guidelines to decide, on the basis of the characteristics of a given problem, whether an agent-oriented approach is appropriate.

#### 1.3.1 The approach

In the MESSAGE approach, **an entity (both inside and outside the system to be developed) is not intrinsically an object or an agent**. It is the developer who, during development, may have decided to consider it so. Depending on this choice the developer will describe that entity in terms of attributes and methods or in more sophisticated terms such as goals, knowledge, and actions.

Even an extremely simple entity such as a switch can be considered as an agent. However it is clear that describing a switch in terms of goals that it needs to achieve, actions it is able to perform, is not helpful as it increases the complexity of the problem without adding any important information about the switch. A traditional object-oriented description (e.g. an object with two states, ON and OFF, and a method PUSH() that changes the state) is definitely more appropriate in this case.

In case of complex entities, however, an agent-oriented description can be beneficial as it provides a richer set of base concepts and a higher degree of flexibility.

The guidelines presented in this section are intended to help the developer in deciding when the describing of all or some of the entities in a system as agents (namely adopting an agent-oriented approach) is beneficial for the development process.

### 1.3.2 The agent definition

The purpose of this definition (in line with the previously presented approach) is **NOT** to have a criterion on the basis of which it is possible to state whether a given entity (a software module, a living being, a robot....) is or is not an “agent”. On the other hand it has to be considered as *a preliminary list of the features that will be ascribed to the entities that the developer will choose to consider as agents.*

The main features in the definitions are

- An agent has certain *knowledge* of the world it lives in.
- An agent is responsible for achieving/maintaining certain *goals* that drive its behaviour.
- An agent is able to *observe* the status of certain objects in the environment and to sense certain *events*.
- Mutual interactions between agents are described in terms of *communicative actions* that
  - Have an intrinsic well-defined *semantics* and
  - Embed a content expressed in a given *content language* and containing terms whose meaning is defined in a given *ontology*
- Besides communicative actions, an agent can perform *direct actions* that affect properties of objects in the environment.

It should be noted that Mobile Agents are considered outside the scope of MESSAGE.

### 1.3.3 The Guidelines

The guidelines are summarised below.

***I) An agent-oriented approach is beneficial in situations where complex/diverse types of communication are required.*** Human oriented communication or interaction between heterogeneous entities are examples.

Communicative-act based communication in fact limits the number of message types thus avoiding the proliferation of object methods. This is paid for in terms of more complex arguments (the message content), but a lot of code for handling them can be shared among all agents that “speak” the same content language and “understand” the same ontology. Moreover, sharing a limited set of message types with well-defined semantics allows a widely applicable set of interaction protocols to be defined.

***II) An agent-oriented approach is beneficial when the system must perform well in situations where it is not practical/possible to specify its behaviour on a case-by-case basis.*** E.g. where the system will be deployed in an unknown environment, a graceful degradation of the performance is required, and many alternative solutions are possible.

An agent in fact has a goal-oriented behaviour that is not specified in terms of a mapping from input to output, i.e. in terms of *what* it should do. Instead, it is specified in terms of *how to decide* what to do. This approach leads to more flexible systems that can adapt their behaviour under changing circumstances to satisfy the goals for which they are responsible.

**III) An agent-oriented approach is beneficial in situations involving negotiation, co-operation and competition among different entities.** E.g. where different tasks with conflicting goals must be carried out concurrently, the system is required to serve multiple stakeholders.

In fact agreeing or refusing to accomplish a task that is requested by another agent is made explicit and is not embedded in the body of an object method.

**IV) Taking into account the goal oriented behaviour and the suitability in negotiations, an agent oriented approach is beneficial when the system must act autonomously** e.g. on behalf of a user.

**V) Multi agent systems are intrinsically highly modular.** This is beneficial when the system is expected to be expanded or modified or when the purpose of the system is expected to change.

## 1.4 Overview of the MESSAGE methodology

A methodology is a codified set of procedures for some phase of software engineering, such as analysis and design. The MESSAGE methodology should provide a language, a method and guidelines on how to apply the methodology, that cover at least the analysis and design phases. The methodology should also explain the relationship to implementation, testing and deployment phases.

The project has focused on producing a deliverable for the analysis activity of the MESSAGE methodology. Later stages in the lifecycle will be addressed in deliverable D3. This section provides a global overview of the different activities in the software life cycle that MESSAGE aims to cover by the end of the project. The software life cycle and the Rational Unified Process are briefly presented as the background for discussing what a methodology should provide. Then the different activities of MESSAGE, i.e. analysis and design, are reviewed and the artifacts of each phase and the relationship with other activities are discussed.

### 1.4.1 The Software Life Cycle and Methodologies

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 characterising particular aspects of the life of the system. Phases are periods of time where specific activities are carried out and results produced which may change the status of the system. Activities in each phase may be performed by different actors such as users, customers, owners, managers and developers, according to their particular role within the development process.

A system engineering methodology groups the methods and principles used in a particular discipline. A method is a systematic way of doing something, or alternatively the techniques or arrangements of work for a particular subject.

Life cycle models and methodology are complementary. While life cycle models provide descriptions of both the behaviour of the system, and the rules of interaction between the system, the environment and the actors, methodology focuses on providing methods, guidelines, descriptions, and tools in each life cycle phase, in order to produce systematically the planned results.

From an engineering perspective agent based systems are software products to which existing software life cycle models and supporting methodologies might be applicable. These methodologies provide a body of process models, representation formalisms and artifacts to guide software development.

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 specification documents, analysis models, designs, code, testing specifications, testing reports, performance evaluation reports, etc. Deliverables act as input to other activities.

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. Specialist tools are often used to enable users 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.

**Notations.** The Unified Modelling Language (UML) [1] was selected for two complementary reasons: It provides a uniform and consistent visual formalism in which to express the results of different object oriented methodologies proposed during the decade of 1990; UML can be considered a de facto standard for defining the most significant deliverables in software development.

**Process Model.** The Unified Process for software development proposed by Rational [11] was considered the most suitable for MESSAGE. Reasons for these choices are

- It covers 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
- It uses UML as supporting notation.

More details about the Rational Unified Process model for software development (RUP) are described in the next section

#### **1.4.2 The Rational Unified Process model for Software Development (RUP)**

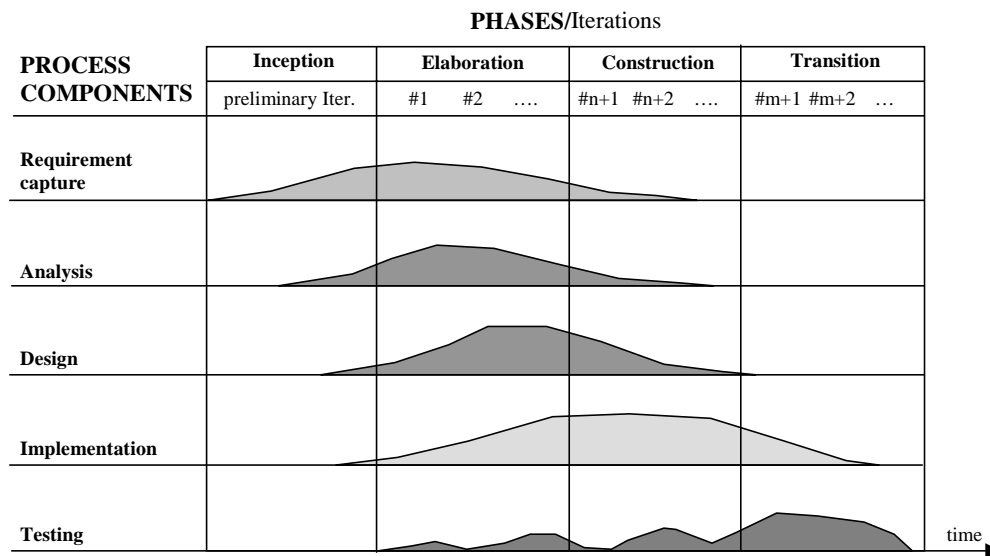
The RUP considers the life cycle to be made up of a series of cycles called iterations where progressive product releases are developed. A software release is more than just the system code. The concept of software system includes all the artifacts that make it possible for the machines and the humans (workers and stakeholders) to interpret the description of the system. The term artefact refers to any kind of information created, produced changed, or used by workers in developing the system. Examples of artifacts are UML diagrams, user interface sketches, code libraries, and prototypes.

Each development cycle consists of four phases:

- Inception phase – Specifying the project vision

- Elaboration phase – Planning the necessary activities and required resources; specifying the feature and designing the architecture
- Construction phase – Building the product
- Transition phase – Supplying the product to the user community

Within each phase a number of iterations may occur. An iteration represents a complete development cycle where different activities such as requirement capture, analysis, design implementation and testing are carried out. The extent to which a particular activity is carried out is dependent upon the phase of development see Figure 1. For example the main activity during the Inception phase is Requirement capture, but it is also possible to do a proof of concept prototype and thus analysis, design, implementation and testing activities are also needed.



**Figure 1 Phases, activities and iterations**

Activities and results/artifacts defined within each phase are summarised in the table below.

<b>Inception phase</b>	
Goal: brought up the first idea about the system, evaluate their feasibility taken into account the views of all the actors e.g. users, owners, developer, etc.	
Activities	Results
<b>Requirement definition</b> to describe the conditions or capabilities to which the system must conform, and what the system should do and not do <b>Analysis</b> to build the Domain Model, to define significant use cases, and risk detection <b>Design</b> to draw the initial architecture	Textual descriptions of system requirements Domain Models Business model and system context. Architecture descriptions drafts with outline views of use cases, principal components, and generic behaviour Possibly a proof of concept prototype demonstrating a reduced set of use cases Risk analysis Possibly a planning of the elaboration phase and

	construction phase
<b>Elaboration phase</b>	
Goal: to obtain a complete and detailed vision of the system.	
<b>Activities</b>	<b>Results</b>
<p><b>Requirement definition:</b> to produce functional requirements, user interface requirements, infrastructure requirements, and non-functional requirements e.g. performance, robustness, fault tolerance, etc.</p> <p><b>Analysis</b> to produce an analysis model, including Use cases, Domain Model and risks</p> <p><b>Design</b> to produce the design model</p>	<p>Descriptions of use cases, analysis models, design, deployment, and implementation</p> <p>Detailed risk analysis</p> <p>Detailed business model</p> <p>Detailed plan for the construction phase</p> <p>Possibly prototypes showing user interface, assessment of techniques, stress, etc.</p> <p>User manual and system testing plan</p>
<b>Construction phase</b>	
Goal: build the system/product	
<b>Activities</b>	<b>Results</b>
<p>Analysis. Adaptation of analysis model</p> <p>Design Adaptation of design model</p> <p>Implementation</p> <p>Testing</p>	<p>The executable software including configuration and installation</p> <p>Analysis model, design model, implementation model, testing model including testing results.</p> <p>Completed and updated architecture description</p> <p>Preliminary user manuals.</p> <p>Legal documents such as contracts license documents, etc.</p>
<b>Transition phase</b>	
Goal: delivery of a system ready for exhaustive testing i.e. a beta release.	
<b>Activities</b>	<b>Results</b>
<p>Analysis. Adaptation of analysis model</p> <p>Design Adaptation of design model</p> <p>Implementation</p> <p>Testing</p>	<p>The executable software including configuration and installation</p> <p>Completed and corrected product release including all models of the system</p> <p>Completed and updated architecture</p> <p>Manuals for all the users</p> <p>Legal documents such as contracts license documents, etc.</p>

**Table 1 Activities and results**

### 1.4.3 Requirements

A statement of requirements (SoR) is a description of the need that the customer expects the system to fulfill. According to Sommerville [12] it is a statement in a natural language of what user services the system is expected to provide, and should be understandable by non-specialist staff, client and contractor management, and potential system users. It is important that it should not build in assumptions about *how* the need will be fulfilled.

The issues for multi-agent systems (MAS) are basically the same as for other types of software, and MESSAGE recommends using existing requirements approaches and standards for establishing the requirements for a MAS.

### 1.4.4 Multi-Agent System Analysis

The purpose of analysis is to produce a system/specification (or analysis model). This is an interpretation of the problem to be solved (i.e. the requirements) represented as an abstract model in order to a) understand the problem better, b) confirm that this is the right problem to solve (validation) and c) facilitate design of the solution. It must therefore be related both to the statement of requirements and to the design model (which is an abstract description of the solution).

MESSAGE provides an **agent analysis model** that introduces the agent (as opposed to an object) abstraction as one of its fundamental building blocks, describing both state and behaviour. Other important agent-oriented concepts, such as tasks and actions, goals (with associated relationships indicating dependency, conflict, constraints, etc), dialogues and messages (as objects), roles (of agents) and acquaintance relationships can be modelled. The MESSAGE agent-oriented modelling language provides the means of modelling the dynamics of interactions such as messages and protocols.

### 1.4.5 Multi-Agent System Design

The overall goal of design is the transformation of the analysis entities into computational entities, which may be implemented into the computational environment using existing languages and tools. Software design consists in defining the overall architecture for a software system. Design consists in identifying the major components of the system, specifying what they are to accomplish, and establishing the interfaces among the components. Design is the first step of transforming the requirements (a statement of what is to be done) into the functioning software (how the requirements are accomplished). Design also includes lower level work such as detailed specification of data structures and algorithms within the identified components. Requiring structural aspects to be specified in a configuration language or architectural description language (ADL) enforces this clear separation between program structure and algorithmic behavior.

The MESSAGE **design model** will provide agent interaction constructs to describe inter-agent communication and information exchange between agents and their environment. The architecture model will also provide means (such as a facilitator or directory agent) whereby the agent can identify other agents with whom he must communicate. The design model will also model the **agent organisation**. This covers the capability of agents to co-operate with other agents for problem solving. The Agent Model will also describe the **agent's internal structure** and behavior. The

internal design of the individual agents will be described in terms of reusable components. Constructs for describing basic agent concepts will be defined: observation, action, communication, goals, plans, etc. These design concepts will be related to corresponding concepts in the analysis model, but address issues about how these concepts should be implemented. Multi-Agent System design should take into account platform choices and adherence to standards such as FIPA. Although in principle design should be independent of a specific platform, design abstractions constrain designs to a specific family of platforms.

As in analysis design can be done by stepwise **refinement** of analysis entities such as the agents, their interactions, their tasks, and the domain knowledge. The design method could consist of the following design steps:

1. Decide/select the computational architecture for the agents in the organisation and define their computational interfaces;
2. Refine interaction diagrams showing the realisation of organisation tasks into computational conversation/messages using agent's interfaces. UML Interaction diagrams, and activity diagrams may also be used;
3. Analyse and define the behaviour of each agent for performing its tasks, and for interpreting/reacting to external messages. UML Interaction diagrams, and activity diagrams may be used;
4. Modify/refine the internal structure of the agents when needed to perform the tasks. Add new knowledge/components if needed.
5. Update/tune the structure and the computational behaviour of each agent to satisfy functional requirements.

Prototyping might be used for validating design choices concerning the Agent internal structure and behaviour, or the organisational aspects such as conflict resolution, dependencies between agents, information flow, and control.

## 1.4.6 MESSAGE and other Stages of the Software Life Cycle

### 1.4.6.1 Implementation

Implementation produces the system itself in terms of software source code, scripts, binaries, executables, configuration data, etc. The implementation activity is tightly related to the design of the system and it should be consistent with it. Given a good design model, the system implementation should be a translation from the modules, interfaces, algorithms and other constructs defined in the design model into data structures, procedure calls and sequences of instructions directly supported by the selected programming language;

MESSAGE should provide **guidelines** for the selection of an agent **platform** supporting the main features required by the system to be developed. Guidelines should be provided for the selection of a suitable agent-oriented programming **language**. Guidelines should be provided to guide the developer in implementing the system so that it is possible to retrieve the system architecture from the implementation.

### 1.4.6.2 Testing

The aim of testing in software systems is to determine whether or not the behaviour of the implemented system satisfies its specification. However, testing can only show the presence of errors, but cannot guarantee the absence of them. Testing is usually associated with verification and validation activities that occur in all of the phases in the life cycle. It is often said that verification asks the question: “*Is this program right?*”, while validation asks: “*Is this the right program?*”. Validation refers to the utility of the end product e.g. running code in the computational environment, with respect to real business needs. Testing may well reveal that assumptions the customer made in the requirements are wrong. Verification deals with checking global properties of the artifacts produced in each phase such as consistency, completeness, and correctness. Verification activities are also performed to determine consistency, correctness and completeness between different artifacts produced in each phase such requirements, architecture and component design. Testing either manual or automated examines system behaviour by executing sample data sets.

MESSAGE design models should allow high-level representation formalisms to be derived for testing interaction scenarios for Multi-Agent Systems (MAS). These scenarios might include elements such as initial stimulus, preconditions, goals of each agent, the expected conversation, relationships and hierarchical links with other scenarios, etc. Automatic generation and execution of MAS interaction testing should be provided.

### 1.4.6.3 Deployment

Deployment models for distributed systems aims to describe the relationships between logical entities defining the system functionality and the physical entities –processors and other computational resources- where logical components will be executed. In summary the deployment model describes the mapping between the software architecture and the hardware.

UML provides an object model notation to define deployment artifacts. The physical topology of the hardware on which the system executes is represented as a collection of nodes. A node is a physical entity with local memory and processing capability. Nodes might be described as class-like entities using attributes, operations and relationships such as dependency, generalisation and association. Dependency relationships can be used to show the correlation between a node and the components it deploys. Association relationships can be used to show the means of communication between nodes such as Internet, LAN, ISDN, etc. This allows for each association to define roles, multiplicity, and constrains such as bandwidth, speed, etc. Associations might also be stereotyped to model new kind of connections.

The deployment model can describe several different network configurations, including test and simulation configurations. It plays an important role in design and implementation since the distribution model might determine the realisation of critical components of the system dealing with communication and parallel processing. These components have a direct impact on key issues such as performance, robustness, stability and security.

The MESSAGE design model should allow high-level description formalisms to be derived for describing the physical and computational environment in terms of nodes, connections, resources and constraints. A high-level description formalism for

describing the deployment process should be used: agent's location, roles, preconditions and constraints, etc.

#### **1.4.7 Conclusion**

Although the MESSAGE methodology focuses on notations and methods for agent analysis and design, it must but also support the relationship to other stages of the life cycle such as requirements, implementation, testing and deployment. Some of this support will be in the form of guidelines whereas others will be the ability to derive high-level representations of the specification for these other stages.

## 2 Overview of Analysis Models

The analysis activity primarily involves developing and maintaining a model of the Multi-Agent System and its environment at a high level of abstraction. Its purpose is to: understand the problem in multi-agent terms; confirm that this is the right problem to solve (validation); form a basis for design of the solution.

### 2.1 The MESSAGE modelling language and UML

MESSAGE has adopted a metamodelling approach. Metamodelling concerns the definition of the concepts used in a modelling language. Many software engineering modelling languages, including UML, are based on metamodels. The 'standard' metamodelling architecture has the following four layers (lowest layer first), illustrated by database-oriented examples:

1. Information (or object or data) layer: e.g. describing the actual records in the database
2. Model: e.g. describes the types of record used in the database
3. Metamodel: e.g. describes the concept of a database record
4. Meta-metamodel: defines the language used to define concepts such a database record.

Thus many modelling languages can share a common meta-metamodel. The four main concepts in the metamodelling language (i.e. the meta-metamodel) used by UML and MOF (OMG Meta-Object Facility [2]) are (adapted from [2]):

- Classes: which model the meta-objects
- Associations: which model binary relationships between meta-objects. These are in addition to *generalisation* and *aggregation*.
- Data-types
- Packages: which modularise the models.

These will be familiar to users of UML, as the basic concepts are used in all the layers.

The MESSAGE modelling language is related to UML as follows:

1. It shares a common metamodelling language (meta-metamodel) with UML and MOF (layer 4)
2. It extends the UML metamodel with agent-oriented concepts.

The MESSAGE metamodel is introduced gradually below as the concepts are used. The full metamodel is described in PIR3.4.

### 2.2 Example scenario

The explanations of the model are illustrated with examples based on a simple problem scenario. We suppose that EURESCOM has a travel department (or a contract with a travel agent) to make travel arrangements on behalf of its project supervisors, and that there is a requirement for a software system to assist with this

process. The purpose of the travel department (supported by the software system) is to:

- Maximise the effectiveness of the project supervisors by relieving them of the burden of making travel arrangements, helping make most effective use of their time, and providing an adequate standard of comfort so that they arrive fresh for meetings, etc.
- Make most effective use of the travel budget by controlling costs.

There are two classes of customer for the system: Project Supervisors (PS) who make requests for travel arrangements in order to attend project meetings, and the Finance Director (FD) who monitors spend against the travel budget and alters policy and procedures accordingly. The PS' motivation is to get convenient and comfortable travel and accommodation so that they can perform their work effectively. The FD's motivation is to save money on individual trips so that the budget will cover all necessary travel and any excess budget can be used elsewhere. There is some 'behind the scenes' negotiation between the PS collectively and the FD regarding what constitutes an adequate standard of comfort, etc., and the FD takes this into account in setting the policy.

Making the travel arrangements is a shared task between human travel agents and the software system. The requirement is flexible about how this task is shared. There is also a certain amount of flexibility as regards the interface between the customers and the system.

### **2.3 Levels of resolution**

It is quite usual for analysis methods to begin by examining the relationship of the planned software system to its users, organisational context, etc., and then to recursively analyse the software system at finer and finer resolution until the components of the system can be specified sufficiently precisely for design to take place. We term the highest level view, in which the planned software is seen as a single element of the system, as Level 0, the first level of decomposition, Level 1, etc.

Agent-oriented analysis models software systems by analogy to human organisations and societies, so that very similar concepts, notations and analysis techniques can be used at Level 0 and at subsequent levels. This contrasts to some extent with UML, which uses the concept of 'actor' to model a human user or other interested party at Level 0, but uses class / object to model the software system and its components. In agent-oriented analysis human actors and software agents are conceptually similar and can be viewed as subclasses of Agent. Thus, at Level 0, we see the software system as an Agent on a par with human and organisational agents, and we can define its role in the parent organisation in a similar way. At Level 1, we can use analogous concepts to relate the agents within the software system to the software system as a whole. In other words, what appears as an agent at one level of resolution will appear as a community of agents if decomposed at the next level of resolution.

### **2.4 The analysis models**

Agent-oriented analysis employs 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 sub-models that emphasise different

aspects of the full model. These are not disjoint models, but are rather different perspectives on a single complex model. The set of models is as follows:

- Domain (Information) Model (DM)
- Agent Model (AM)
- Organisation Model (OM)
- Goal/Task Model (GM)
- Interaction Model (IM)

This set is broadly similar to that used in MAS CommonKADS [9].

The **Agent Model** (AM) consists of descriptions of the purpose, relationships, behaviour, and other attributes of individual agents and of roles. Each description gathers together information on an agent or role from other models and adds internal detail. A role is essentially a position that may be filled by a qualified agent.

The **Organisation Model** (OM) focuses on the structure of organisations and the relationships among the agents that they contain<sup>1</sup>. It also describes conflict resolution mechanisms, norms, etc, that enable the group of agents to function as a unit serving a common purpose.

Whereas the OM describes how organisations are decomposed into constituent agents, the **Goal/Task Model** (GM) describes how high level goals (for example defining purposes of an organisation) are decomposed into lower level goals (e.g. ones that can be assigned to constituent agents). Similarly, it shows how high level tasks (e.g. services provided by an organisation) can be decomposed into sets of sub-tasks (e.g. services that are provided by the constituent agents). Note that a goal describes motivation (e.g. a need to be in a particular location at a particular time), while a task describes activity (e.g. catching a flight to that destination).

Information about the problem domain is captured in the **Domain (Information) Model** (DM). In the example scenario the DM would capture entities and relationships such as *ticket, airline, airport, flies to*.

The **Interaction Model** (IM) describes interactions between agents and the protocols that implement them. Essentially, an Interaction defines at high level a means of agents influencing each other, e.g. negotiating the terms of a service. An example of Interaction is the dialog between a PS and the travel office that captures the PS' requirements regarding a particular business trip. The Interaction identifies the common goal of the participants (to define the travel requirements), the initiator (the PS), etc. There will be an exchange of information, questions asked, etc. as the details are gathered, but this is not important at the level of abstraction of the Interaction. Where it is necessary to prescribe an order to messages, this can be done via Protocols.

In deriving a model of Multi-Agent System, there is a natural ordering to development of the models. First OM and GM are developed and linked so that there is a correspondence between sub-goals and the purposes of agents and between sub-tasks and services the agents can provide. This process identifies the agents and their main features, the details of which are elaborated in the AM. However, to achieve the

---

<sup>1</sup> An organisation is not an agent: An organisation is a collection of agents with a common purpose. More than just gathering the agents, it gives a structure to the collection of the agents, putting restraints on their behavior (through rules).

higher-level goal the agents must cooperate, and this drives the development of the IM. In parallel with all of this, the DM is elaborated, as extra domain concepts are required.

The following sections describe and illustrate the analysis models in more detail.

## 3 Organisation Model

### 3.1 Purpose of Model

The Organisation Model aims at defining the structure and the behaviour of a group of agents (in the Multi-Agent System and in the external organisation) working together to reach common goals. The Organisation Model represents the responsibilities and authorities with respect to entities such as processes, information, and resources. It represents the structure of the organisation in terms of sub-organisation such as departments, divisions, sections, etc.

Organisation might be considered from two complementary views. The social view, which characterises the overall behaviour of the group, and the individual view, which characterises the behaviour of the agents to achieve common/social goals. The Organisation Model deals with the social view and the Agent Model covers the individual view.

From an engineering perspective the OM provide a useful abstraction for understanding the overall structure of the Multi-Agent System, what the agents are, what resources are involved, what is the role of each agent, what are their responsibilities, which tasks are achieved individually and which achieved through co-operation.

### 3.2 Structure of Model

The structure of the model is represented in

Figure 2. The OM consists of one or more inter-related Organisations, which might have organisational relationships representing control dependencies, and/or flow of information dependencies such as reporting dependencies. Each organisation consists of four entities describing the purpose of the organisation, its control mechanism, its organisational structure and the workflow for performing organisation tasks. These entities are defined more in detail below

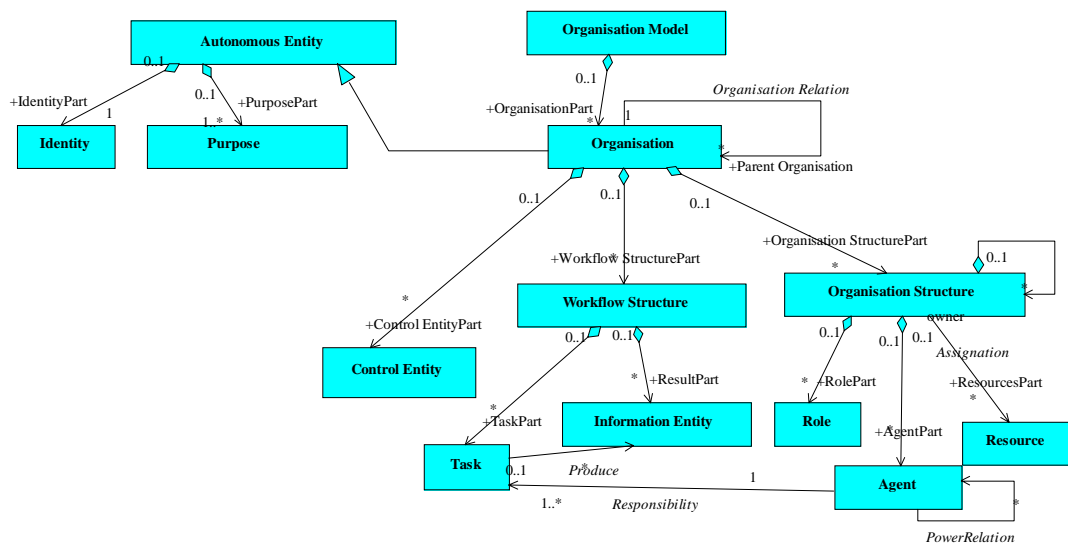


Figure 2 Structure of The Organisation Model

### 3.3 Concepts

The main concepts used in the Organisation Model are described in this section including Organisation, Organisation Structure, Control Entity, Workflow, Resource, Power Relationship and Organisation Relations. The remaining concepts used in the Organisation Model, including Agent, Role, Task and Information Entity, are described in the next Analysis Models.

**Organisation:** this concept characterises a group of agents working together to reach a common purpose. Organisations are defined recursively. An organisation may consist of a single agent, a group of cooperating agents, and/or an aggregation of organisations with relationships between them. The organisation might be described by its purpose, its structure, the control mechanisms, and the workflow for performing organisation tasks. Each of these aspects is described by concepts, which are described below.

**Organisation Structure:** This concept defines the composition of the organisation and internal relationships. Organisation Structures are defined as graphs where the nodes are made up of agents, and/or roles, and/or other organisation structures. The Organisation Structure may have assignment links to resource entities. Agents in an Organisation Structure may play different roles or collaborate with other agents to play organisation roles. Agents may have power relationships between them. This relationship illustrates control dependencies and information reporting.

**Control entity:** This concept characterises the way in which conflicts and control mechanisms are defined in the organisation. Taxonomies of control classes such as conflict resolution entities, norms and laws may be used at the domain level for illustrating the control policy of the organisation.

**Workflow Structure:** This characterises the arrangement of tasks and goals in the organisation. Responsibility relationships illustrate how tasks are assigned to

agents/what tasks an agent is required to do, and what results should be produced by each task.

**Resource:** This characterises the physical and logical entities needed for the organisation to perform its tasks. This might be databases, legacy systems, input/output processors, etc. The concept of Resource is also described in the Agent Model in section 5.3.

**Autonomous Entity:** is an entity with an identity and a purpose.

**Identity:** a property or characteristic that uniquely distinguishes an individual entity.

**Purpose:** a goal or utility function that is central to identity of entity and which guides its decisions and actions.

**Power Relationship:** This concept denotes the degree of influence agents (playing roles) have over each other. They allow us to construct management hierarchies (i.e. who is the boss of whom). *isInfluencedBy* is the weaker form indicating an indirect line of authority.

- **isInfluencedBy:** A type of power relationship between agents indicating the power to change the internal state of an agent by another agents or one persuade others agents through negotiation to perform certain task and/or to send information.
- **isSubordinateTo:** (reverse: manages). This indicates a 'line management' manager-subordinate relationship. CommonKADS [7] refers to this as a 'formal power relationship'.

**Organisation relations:** An organisation relation denotes dependencies between organisations such as subsidiary of divisions, departments, etc

## 3.4 Notations

UML may be used for the definition of the structural (static) aspects of the organisation and for the definition of behaviour (dynamic) of the organization.

### 3.4.1 Structural Description of the Organization

For describing the structural aspects of organisations the UML metamodel described in

Figure 2 is used as a template. During analysis the analyst first identifies the concrete organisation, and decides which optional entities are relevant to characterise the organisation under analysis. The recursive components of the Organisation Structure allows for the definition of complex organisation structures with specific roles, agents and resources.

Instantiation of *Organisation* relationships allow the definition of inter-organisational relationships such as business-to-business relationships, government to company relationships, dependencies, ownership, etc.

Instantiation of *power* relationships between agents allow the definition of authority and reporting dependencies.

The description of the organisation is through description entities. The notation for these entities may be structured text. For example the notation for describing a purpose may be the following table:

#### **Purpose description**

<i>Organisation identification</i>
<i>Last update time</i>
<i>Mission of this organisation: Textual description of the mission</i>
<i>Goals: Links to goal description objects</i>
<i>Responsibilities: Links to responsibility description objects</i>

### **3.4.2 Behavioural description of the organisation**

UML notations may also be used for illustrating the following aspects:

- How the organisation behaves with respect to external entities such as other organisations and clients.
- What is the internal behaviour of the organisation in achieving specific goals or performing specific tasks?

Use cases, interaction diagrams, and activity diagrams may be adequate for both aspects. Activity diagrams are widely accepted for representing workflow as illustrated in Figure 4.

### **3.5 Example**

The example in Figure 3 shows the structure of an Organisation that provides flight tickets to potential users. The organisation model is called TravelAssistanceAgency. The structure of the TravelAssistanceAgency class is derived from the definition of Organisation meta-class.



TravelAssistanceAgency
SuperClass: Organisation
Purpose: Goals Links to goal description
Organisation structure Description: The Organisation Structure is made up of a team of specialised agents. The user agents assist the user in performing user-related tasks. The information agents perform tasks related to information searching. One or more information agent are subordinated to the User agent Roles: Flight information and ticket provisioning Agents: InformationAgent, UserAgent
Workflow structure Description. The tasks performed by the Organisation are defined by the TATaskPart relation. The results produced by each task are defined by the TAResultPart relation. Relations between tasks and results are shown in the class diagram. Tasks: SuggestOtherFlight, FindCheaperTicket, BuyTicket, SendTicket Results: FlightSuggestion, TicketOffer, FlightTicket

An example of the use of a UML activity diagram for illustrating the behaviour of the Organisation is in Figure 4 below. The interaction diagram shows the tasks performed by the Travel agency Organisation, Eurescom Project supervisor and Eurescom financial department for flight ticket provision.

The process starts when the project supervisor defines the requirements for his or her travel, and ends when the financial department pays the bill. The travel agency should find the cheapest ticket satisfying the travel requirements, make the reservations and send the tickets. Tasks are interrupted until results from other tasks are available. This is represented by the long bar.

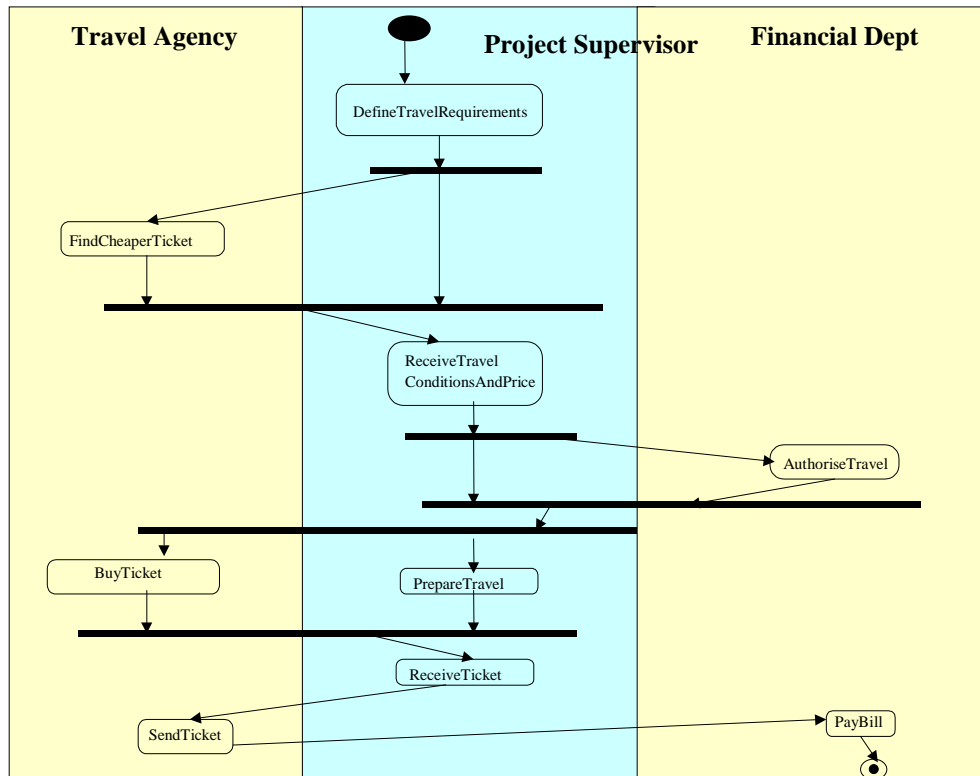


Figure 4 Example of an Activity diagram

### 3.6 Guidelines

The Organisation Model should be progressively built by stepwise refinement.

**Step 1:** Start analysis modeling the system and its environment by identifying involved stakeholders<sup>2</sup> and their relationships. In this model stakeholders and systems are modeled as organisations and their relationships are modeled as organisation relationships.

**Step 2:** Identify examples of interactions among stakeholders and the system for illustrating functional requirements.

- From these examples identify the tasks/goals. Doing this the parts of the system organisation corresponding to purpose and workflow structure should be defined. This may also lead to an initial identification of agents and their power relationships.
- Further elaborate task description and task realisation in the Goal/Task Model.

**Step 3:** Identifies the roles. Role description may be filled

**Step 4:** Identify the agents and the power relationships between them

**Step 5:** Identify the resources needed for the agents to perform the tasks

**Step 6:** Allocate tasks to agents and start the analysis of individual agent behaviour in the Agent Model

<sup>2</sup> Examples of stakeholders are clients, users, developers, owners, managers, providers, and operators.

**Step 7:** Analyse the interactions needed to achieve organisation tasks. Define the Interaction Model

**Step 8:** Refine/update the Agent Model, the Interaction Model and the Organisation Model.

**Results from analysis:** The principal results from analysis are:

The Organisation Model provides information about the structure and the global behaviour of the System. This model may be considered equivalent to the System architecture in Software engineering, but using agents and agent organisations as key concepts. The way in which this System provides the requested functionality to different stakeholders is expressed at a high-level of abstraction through the Interaction Model. More detail about agent's structure and behaviour is in the Agent Model. Descriptions of task and domain are in the corresponding Task model and Domain Model.

## 4 Goal/Task Model

### 4.1 Purpose of Model

The Goal/Task model answers *why*, *who* and *how* questions on the analysis model. The goals of the composite system, i.e. the Multi-Agent System and its environment, and their decomposition into sub-goals, answer *why* questions. The responsibility of agents for their commitments answers *who* questions. The performance of tasks and actions by agents, the goals the tasks satisfy and the decomposition of tasks into sub-tasks answers *how* questions. This model may be used to further describe tasks involved in an Organisational Workflow.

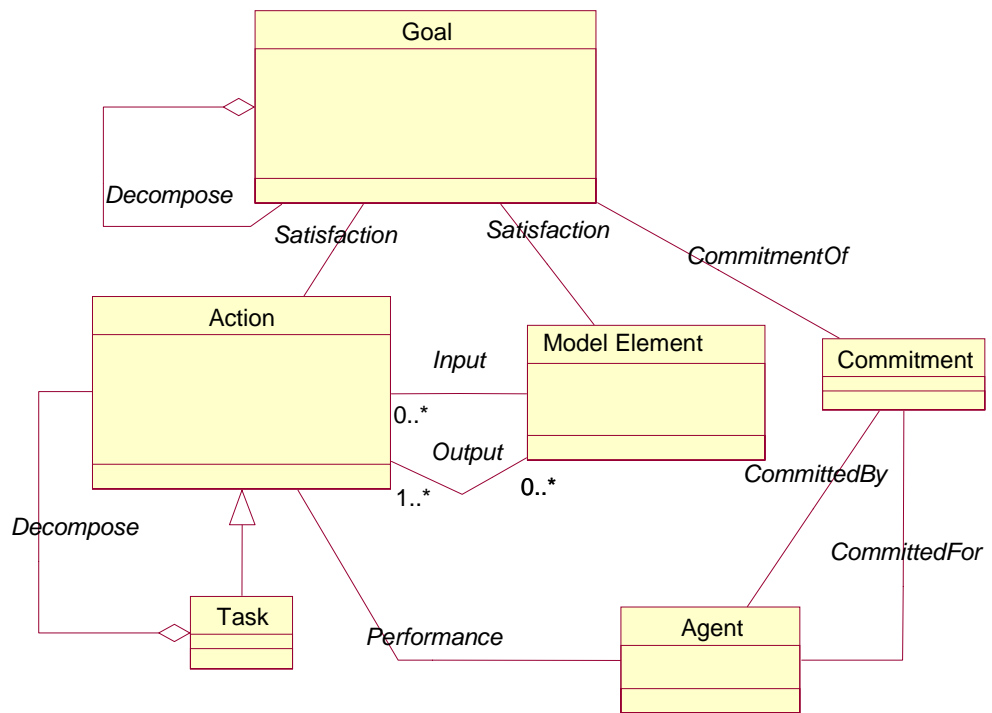
Section 4.2 introduces the structure of the Goal/Task model and introduces the main concepts and relationships. Section 4.3 defines the MESSAGE concepts and relationships that are needed to build Goal/Task models that have not yet been defined. Section 4.4 illustrates the Goal/Task model on the Travel agent example and introduces graphical and textual notations for the model.

### 4.2 Structure of Model

The Goal/Task Model structure is defined in Figure 5. Goals describe the *desired states* of the system and its environment. Tasks describe *state transitions* that are needed to satisfy agent goal commitments. The state transition is specified as a pre- and post-condition attribute pair. Actions are atomic tasks that can be performed by the agents to satisfy their goal commitments. Task inputs are Model Elements<sup>3</sup> that are processed in task. Task outputs are updates of the input Model Elements plus any new Model Element produced by the task. The *desired states* of a Model Element are specified by attributes called Invariants, which are conditions that should always be true.

---

<sup>3</sup> Model Element is an UML concept adopted by MESSAGE, defining elements composing Models.



**Figure 5 Goal/Task Model**

The concepts used in the Goal/Task Model, including Task, Goal, Agent, Task decomposition, Goal Decomposition, commitment, Perform, Input, Output and Satisfaction, are defined in the next section.

### 4.3 Concepts

The Agent concept and the Perform, Input and Output relationships are defined in the Agent Model. This section will define the following:

- Concepts: Situation, Goal, Task, Action and
- Relationships: Goal/Task decomposition, satisfaction and commitment.

#### 4.3.1 Situation Concept

The **Situation** is a set of states of the world that might be expressed using OCL constraint language<sup>4</sup>.

<sup>4</sup> The Object Constraint Language (OCL) is part of the UML and can be used to specify all kinds of constraints, pre- and post-conditions, guards, etc.

### 4.3.2 Goal Concept

A *goal* is a situation, i.e. a set of states of the world, that an agent is committed to achieve/maintain. A set of states of the world is not in general a goal unless there is an agent committed to achieve/maintain this set of states. A goal is defined in terms of :

- a mandatory informal definition
- an optional formal definition (formalDef.).
- a goal structure defined in terms of sub-goals composing the goal

A goal describes an objective for an agent<sup>5</sup>, e.g. as a consequence of playing a particular role. Informally, a goal is a state of affairs to be achieved, maintained, or avoided; or a utility function to be maximised<sup>6</sup>. A utility function is a function that acts as a measure of the desirability of world states from the perspective of a single agent. In general, it could be binary, multi-valued, or continuous valued.

### 4.3.3 Action Concept

An action is an atomic unit of functionality to modify the state of the system and the environment. An action is defined in terms of :

- a mandatory name;
- a mandatory *informal definition*;
- an optional *pre-condition* defining the state that must be true before the task can be performed;
- an optional *post-condition* defining the state that will be true after the task will be completed.

Action inputs are Model Elements that are processed by the action. Action outputs are updates of the input Model Elements plus any new Model Element produced by the action.

### 4.3.4 Task Concept

A *task* is an action that can be decomposed into sub-tasks and actions; it inherits all relationships and attributes. It has a *task structure* defined in terms of sub-tasks composing the task.

However, for a collection of sub-tasks to constitute a task, it is normally necessary for them to satisfy some ordering constraints (e.g. performing task A is equivalent to performing sub-task A1 and then sub-task A2). In order to formalise such ordering relationships between tasks, the following relationships between a set of events and an event are introduced.

- enables:  $\{A\}$  enables  $B$  means that once one event in the set  $\{A\}$  has occurred it is possible for  $B$  to occur.

---

<sup>5</sup> Note that a multi-agent system is itself formally an agent.

<sup>6</sup> It could be expressed as a function to be minimised. In this case a term such as 'energy function' would be more appropriate. There is no loss of generality in just using 'utility function' in the definition of a 'goal', since an energy function is simply a negative utility function.

- precedes:  $\{A\}$  precedes  $B$  means that  $B$  cannot occur unless all the events in the set  $\{A\}$  have occurred.

Next, the idea is introduced that (labelled) events are ‘emitted’ are various stages in the execution cycle of a task. It would be useful to define a standard set of such labels with well-defined significance (start, end, abort, suspend, etc.). Furthermore, (sub-) tasks may not be able to proceed until a specified event has been ‘observed’. Suppose  $\{e_i\}$  is the set of end events of a collection of tasks  $A$ , and  $s$  is the start event of another task,  $B$ . Specifying  $\{e_i\}$  enables  $s$  means that  $B$  can start once one of  $A$  has finished. Specifying  $\{e_i\}$  precedes  $s$  means that  $B$  can start once all of  $A$  have finished. A network of tasks with start and end events linked by enables and precedes relationships can readily be displayed using a UML activity diagram, with the precedes concept represented by a synchronisation bar.

### 4.3.5 DecomposeGoal Relationship

The DecomposeGoal meta-relationship is defined between a goal and a set of goals. A Goal may be decomposed into sub-goals such that the satisfaction of the sub-goals entails the satisfaction of the parent goal. The decompose meta-relationship can only be formally shown to hold if each of the argument goals has a well-formed formalDef. Attribute value.

### 4.3.6 DecomposeTask Relationship

Tasks may be decomposed into sub-tasks. The DecomposeTask relationship is defined between a single Task and a sequence of sub-tasks or actions. The sequence of sub-tasks defines the composition of the sub-tasks that needs to be respected for the sub-tasks to decompose the parent task. The pre-condition of the first sub-task should be implied by the parent tasks pre-condition, and the post-condition of the last sub-task should imply the parent’s post-condition.

### 4.3.7 Commitment Relationship

The commitment ternary relationship is defined between two agents and a goal. An agent has the obligation to establish the goal that is expected by another agent.. An agent may participate in one or several commitments. An agent that has a commitment must be capable of fulfilling it. This means that he must have the knowledge, or the ability to acquire it, and have access to Model Elements that are needed to fulfil the commitment.

### 4.3.8 Satisfaction Relationship

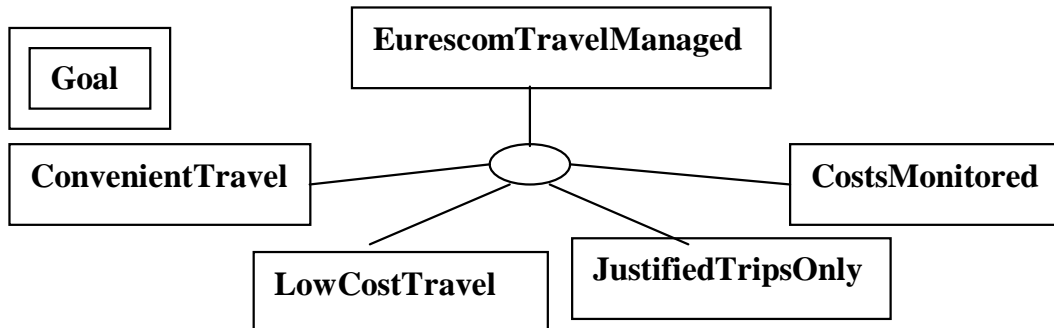
The satisfaction relationship is defined between a Task or a Model Element and a Goal. The relationship means that the Task pre- and post-conditions of the Model Element invariant make the Commitment true.

## 4.4 Notations and Example

The example shows how a goal is decomposed into sub-goals and Commitments, and how the Commitments are assigned to the responsibility of an agent. The Tasks that

satisfy the commitments are then decomposed into actions that are performed by an agent. The notation employed here is adopted from the KAOS methodology [13].

#### 4.4.1 Goal Graph Definition

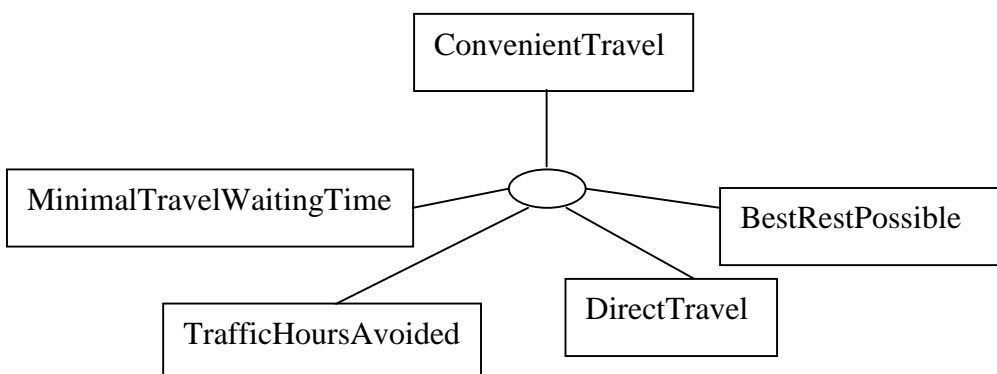


**Figure 6 Travel MAS Goal Graph**

Figure 6 shows that the root goal of the MAS is “EurescomTravelManaged” and that it is decomposed into four sub-goals. The decomposition means that if “ConvenientTravel”, “LowCostTravel”, “JustifiedTripsOnly ” and “CostsMonitored” are satisfied, then the “EurescomTravelManaged” goal is also satisfied. The ordering of the sub-goals from left to right has no meaning in this decomposition. The following schema defines the “EurescomTravelManaged” goal:

<b>Goal</b>	
Name	EurescomTravelManaged
Informal Definition	The Eurescom travel service should be managed to provide a quality travel service to all Eurescom personnel.
Decomposed Into	ConvenientTravel, LowCostTravel, JustifiedTripsOnly, CostsMonitored
Priority	High

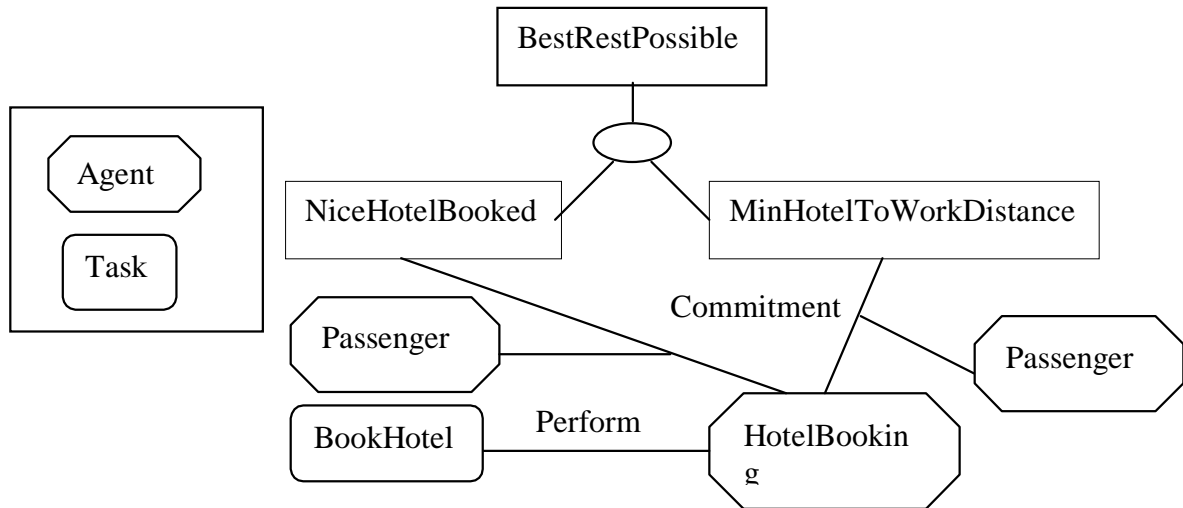
**Table 2 EurescomTravelManaged Goal Schema**



**Figure 7 ConvenientTravel Goal Decomposition**

The Goal “ConvenientTravel” is itself decomposed into four sub-goals “MinimalTravelWaitingTime”, “TrafficHoursAvoided”, “DirectTravel” and “BestRestPossible”.

**4.4.2 Commitment Assignment**



**Figure 8 HotelBookingAgent Commitment**

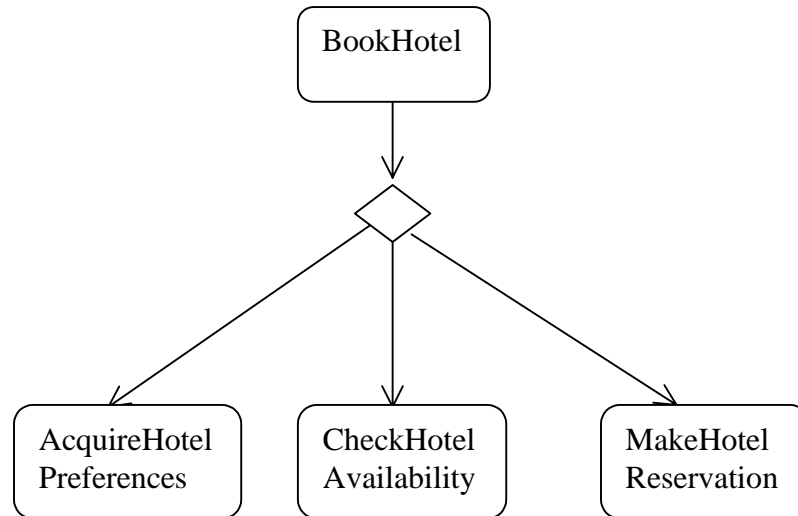
Figure 8 shows how the goal “BestRestPossible” is decomposed into two sub-goals “NiceHotelBooked” and “MinHotelToWorkDistance”. The “HotelBookingManager” is committed to establishing the goals for the passenger agent. This agent performs the “BookHotel” Task to fulfil his two commitments.

The schema for the “MinHotelToWorkDistance” Goal is given below:

<b>Goal</b>	
Name	MinHotelToWorkDistance
Informal Definition	The distance from the hotel to the various working locations should be minimal.
ExpectedBy	Passenger
ObligationOf	HotelBookingManager
Decomposed From	BestRestPossible
Category	Optimisation
Priority	High

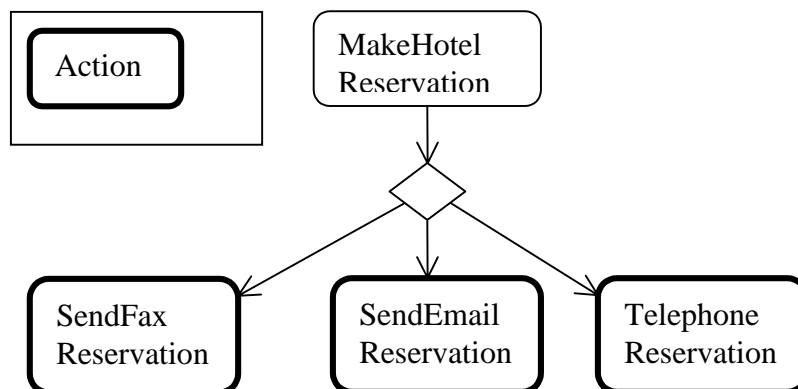
**Table 3 MinHotelToWorkDistance Goal Schema**

#### 4.4.3 Task Decomposition



**Figure 9 BookHotel Task Decomposition**

Figure 9 shows how the “BookHotel” task is sequentially decomposed into three sub-tasks: “AcquireHotelPreferences”, “CheckHotelAvailability” and “MakeHotelReservation”.



**Figure 10 MakeHotelReservation task decomposition**

Figure 10 shows how the task “ MakeHotelReservation” can be alternatively decomposed into “SendFaxReservation”, “SendEmailReservation” or “TelephoneReservation”. The schema for describing the “SendEmailReservation” action is described below:

<b>Action</b>	
Name	SendEmailReservation
Informal Definition	The reservation is made or the agent is informed of failure to reserve
PerformedBy	HotelBookingManager, or HumanSecretary
Input	ReservationRequest, HotelEmailAddress
Output	ReservationConfirmation or ReservationFailure
Pre-condition	The hotelEmailAddress is correct, and is accessible
Post-condition	If there is a reservation, then the reservation is made according to the ReservationRequest

**Table 4 SendEmailReservation Action Schema**

## 4.5 Guidelines

The goal decomposition process shows us how to derive the goals from that define an agent or role's identity/purpose from the system requirements. The agent commitments show how the system requirements are ensured. The process of goal decomposition involves taking the goal defining the purpose of a Multi-Agent System, decomposing it into sub-goals that can be assigned as the 'purposes' of the sub-agents. The MAS may additionally need mechanisms for a) conflict resolution b) co-ordination.

### 4.5.1 When to Switch from Goal to Task Decomposition

Goals allow the analyst to specify *what* should be done, whereas tasks specify *how* it should be done. Task decomposition should start when:

- The desired property can be expressed as a single state transition with a pre- and post-condition expression;
- The task can be performed by an agent of the system.

### 4.5.2 When to Stop Goal Decomposition

Goal decomposition should stop when a task performed by an agent can satisfy the desired property.

### 4.5.3 When to stop Task Decomposition

Task decomposition should stop when tasks can be performed by a single agent.

## 5 Agent Model

### 5.1 Purpose of Model

The Agent Model consists of a set of individual agents and roles. An element of the Agent Model gathers together information specific to an individual agent or role, including its relationships to other entities. It also adds further descriptive detail specific to that agent or role.

PIR2.1 (see also 1.3.2 above) identifies the main characteristics of an agent as being its identity, its behaviour, and its mental state. It also states that an agent is situated in an environment with which it interacts. The agent's mental state is a representation of itself and its environment in terms of knowledge and beliefs, goals, intentions, etc. Its behaviour can be thought of as a set of rules that determine how it responds to events such as the arrival of new information. Both state and behaviour are more complex than is the case for objects. Behaviour includes reasoning and decision-making processes that affect mental state, and mental state includes behaviour-related elements allowing it to perform complex, goal-directed, situation-dependent activities extended over time. There are many alternative theoretical frameworks for mental state and behaviour. MESSAGE is not committed to any one of these.

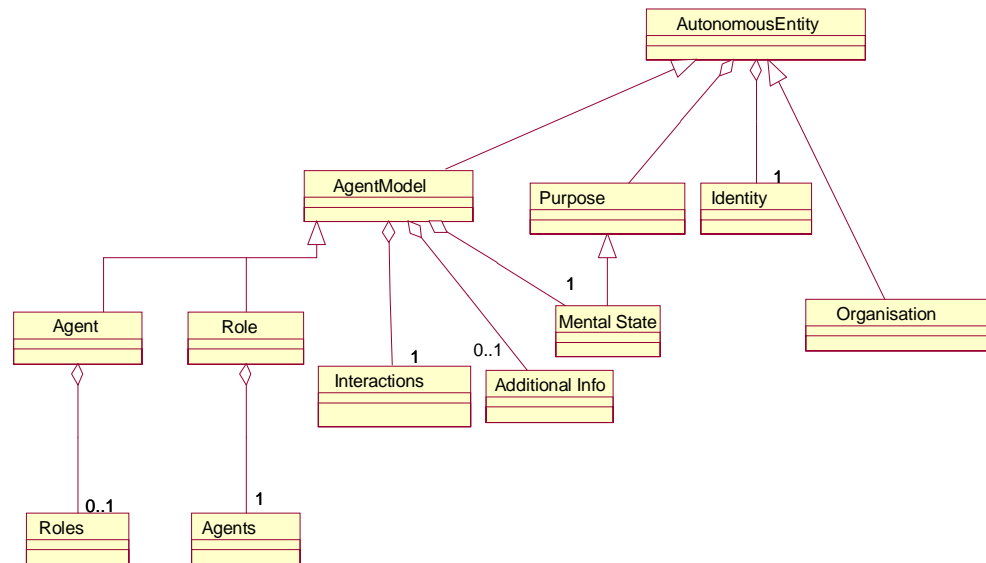
The internal behaviour of an agent can be described as a repeating three-stage cycle: perception, decision and action:

- Agent perception: an agent perceives its environment through Information Entities that describe Events or information on the state of the world. The Information Entities could result e.g. from observation actions by the agent, or receipt of a message from another agent. The perception of new information triggers tasks and actions that update the agent's knowledge and beliefs, which may be thought of as a collection of Information Entities constituting the agent's model of the world.
- Agent decision: an agent decides what tasks and actions to perform on the basis of his objectives and knowledge and beliefs. An agent decides to perform actions that will change the state of the world in such a way that the goals it is committed to are satisfied.
- Agent reaction: the agent performs the actions that it decided upon. Actions are analogous to the private methods of an object in OO programming, and so performing an action is analogous to executing a method. Information Entities are the input and the output of actions (analogous to the arguments of methods). Input and output are not the same as pre- and post-conditions (see the GM), which are concerned with the side effects of the actions. Post-conditions describe the effects that can be expected from performing the action given that the pre-conditions are satisfied.

Note that actions can be triggered directly as a result of perception (reactive response) or as a result of the decision-making process (deliberative response).

The distinction between *agent* and *role*, is that an Agent Model describes characteristics that are inherent to an agent, whereas a *role* describes characteristics that an agent takes on because, for example, it occupies a position in the Organisation Structure.

## 5.2 Structure of Model



**Figure 11 Structure of the Agent Model**

An Agent Model consists of elements corresponding to individual agents and roles. Note that Agents and Roles share high-level characteristics with Organisations, principally identity and purpose. They differ in their internal structures and means of achieving their purpose.

### 5.2.1 Agent

An agent consists of the following attributes:

#### 5.2.1.1 Identity

This is a set of characteristics that identify this agent as distinct from other agents. An example might be that this is the agent responsible for managing the diary of a particular user.

#### 5.2.1.2 Roles

An indication of what roles the agent is able to play. This can be omitted if roles are not used in this application.

#### 5.2.1.3 Interactions with environment

Describes what the agent can perceive, what it can change, and what it can do. The following should be described:

**Sensory input:** Establishes the information the agent has direct access to over and above those described in the acquaintances and resource ownership and access sections. It includes a description of the Events and other Model Elements the

properties of which the agent is able to observe directly. It should indicate the degree of detail of information that is visible to the agent.

**Acquaintances:** An indication of the other agents that the agent can interact with to obtain information and to request to perform actions/achieve goals. The relationships with these agents must also be described in the Interaction Model.

**Resource ownership and access:** Resources for which the agent has special responsibility and access. It should describe what the agent is *able* to do and also restrictions on what it is *permitted* to do.

**Actions and tasks :** actions, tasks, protocols, etc. the agent can perform:

- actions on resources in its environment
- communicative acts (speech acts)
- internal actions (reasoning, planning, etc)

#### 5.2.1.4 Mental state and behaviour

**Purpose:** A description of the purpose/motivation of the agent in terms of goals, utility function, etc.<sup>7</sup> These terms should be further detailed in the Goal Model<sup>8</sup>.

**Behaviour:** An indication of how the agent chooses its response to events in the light of its objectives.

**Knowledge and beliefs:** A description of the main elements of the agent's mental state and inferencing capabilities.

### 5.2.2 Role

In some applications it can be useful to separate responsibilities, permissions and behaviour associated with roles that an agent can play from the characteristics that are inherent to an agent. A role is similar to an agent except that generally permissions are described rather than abilities.

#### 5.2.2.1 Identity

This is a set of characteristics that identify this role as distinct from other roles.

#### 5.2.2.2 Agent requirements

Characteristics required by an agent in order to play this role.

#### 5.2.2.3 Interactions with environment

The following should be described:

**Sensory input:** additional information that becomes available to an agent as a result of playing this role. Restrictions on available information as a result of playing this role.

---

<sup>7</sup> It may be better to make purpose a separate field. This will be re-considered in later versions of the methodology.

<sup>8</sup> At this moment the Goal Model does not cover Utility Functions. This is an issue to be considered in Deliverable 3.

**Acquaintances:** Changes to acquaintances and relationships as a result of playing this role.

**Resource ownership and access:** Changes to access permissions and ownership rights as a result of playing this role.

**Actions and tasks:** Changes to actions and tasks as a result of playing this role. An agent will not acquire new action capabilities as a result of playing a role, but it is possible it might acquire knowledge enabling it to perform tasks.

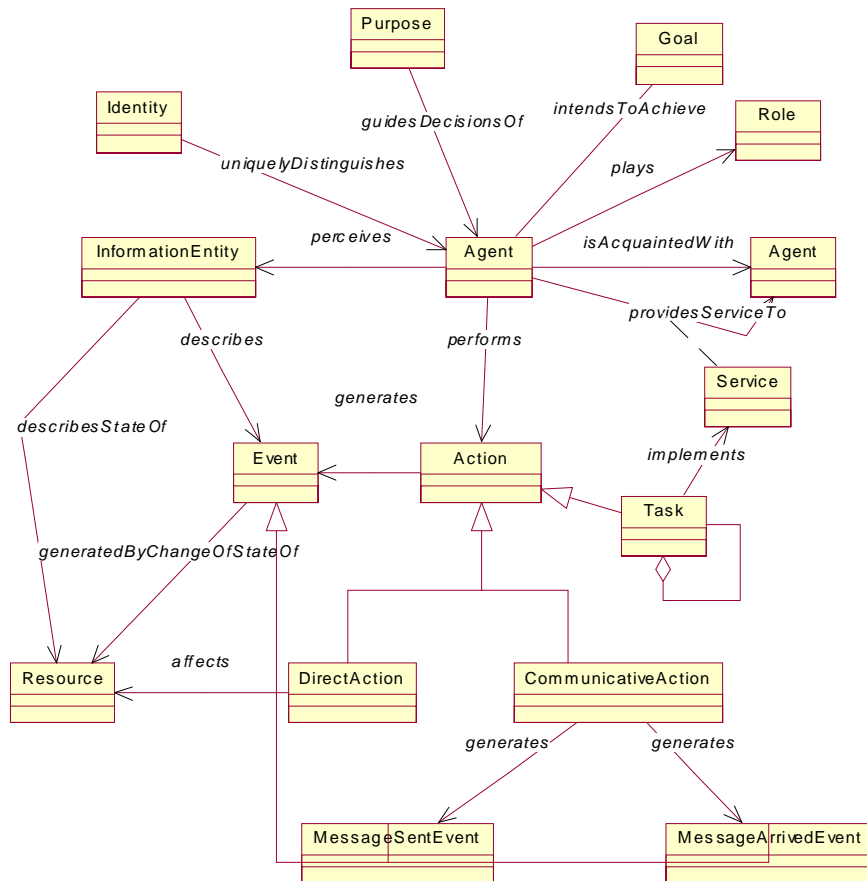
**5.2.2.4 Mental state and behaviour**

**Purpose:** additional commitments acquired by the agent as a result of playing this role.

**Behaviour:** how an agent’s behaviour is modified as a result of playing this role.

**Knowledge and beliefs:** main elements of the agent’s mental state and inferencing capabilities are modified as a result of playing this role.

**5.3 Concepts**



**Figure 12 An agent-centric view of the metamodel**

**Agent:** An Agent is an autonomous entity which characteristics are described in 1.3.2.

**Information Entities.** Agents perceive information entities. These carry two main classes of information:

- Descriptions of Events. Objects describing Events inform the agent that something has happened;
- Information about the state of entities in the Agent's world. Objects of this type inform the agent about, for example the state of some resource parameter.

There are Events corresponding to arrival and sending of messages. The act of Agent A sending a message to Agent B gives rise to two Events, corresponding to the message being sent by Agent A and the message arriving at Agent B some time later. Messages will often contain Information Entities in their content.

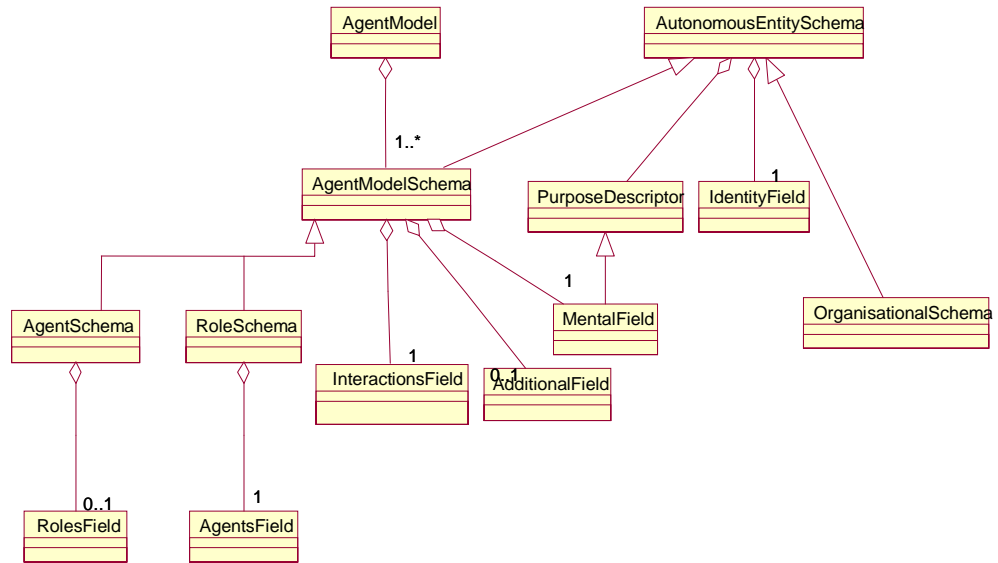
**Resource:** a resource is non-agent entity that is used by or provides information to the greater system. In the Agent Model, we are concerned with what attributes of a resource an agent can see, and what actions an agent can perform on it. In some applications, ownership of or responsibility for a resource may be assigned to an agent, with ownership special access to information and resource functions. Other agents may then need to access the resource via the owning agent. Access to a type of resource may be a pre-requisite for performing certain types of task. A resource may also be the object of goals, for example a goal to maintain a certain property of a resource within bounds. A description of resource from the perspective of an organisation is given in the section describing the OM.

**IntendsToAchieve:** an agent intends to achieve its goals (from the definition of goal). An agent's purpose is its primary goal, and goals are also implied by roles that the agent plays. In addition, an agent may adopt goals dynamically, e.g. because they are tactical sub-goals of its purpose, because they are delegated to it by an agent in authority, or because of an agreement with another agent. Goals undertaken on behalf of another agent are termed commitments. Commitments imply obligations, e.g. to inform the other agent if the goal cannot be achieved. An agent should only intend/commit to a goal if it has the capability to achieve it.

**Service:** A service is a task that an agent is (potentially) willing to perform on behalf of other agents

**Agent characteristics:** An agent has an *identity*, *means of interaction* with its environment (*sensors* and *effectors*), and a means of deciding what to do based on objectives (*goals* or *utility functions*), i.e. a means of translating purpose into behaviour given a perceived state of affairs.

### 5.4 Notations



The notation of the Agent Model is a set of Agent and Role schemas. Each schema contains a number of fields in which free-form descriptions of the relevant attributes are given. See above for the description of attributes.

**Schema and Field:** In terms used in the UML metamodel, names ending in ‘schema’ and ‘field’ denote PresentationElements (sometimes referred to as a view element), which is an element whose purpose is to provide a presentation of information for human comprehension [1]. Removing the suffix ‘Schema’ yields the name of the ModelElement of which the schema is a ‘presentation’. A field is division within a schema.

<b>AgentSchema for:</b>
<b>Identity:</b>
<b>Roles:</b>
<b>Interactions with environment:</b> Sensory input:  Acquaintances:  Resources:  Actions:

<p><b>Mental state and behaviour:</b></p> <p>Purpose:</p> <p>Behaviour:</p> <p>Knowledge and beliefs:</p>
<p><b>Additional Information:</b></p>

The recommended schema notation is simply a table with headings corresponding to the fields mentioned above as shown in the example.

## 5.5 Example

### **Scenario outline:**

A EURESCOM project supervisor (PS) is required to arrange travel to various meetings. There are EURESCOM guidelines that govern the choice, expense and type of travel the project supervisor is allowed to use. Here we consider an agent, within a travel office, which the project supervisor can give travel requirements to. The agent then explores the various transport and accommodation possibilities, within the scope of EURESCOM's travel guidelines. The agent asks the project supervisor to decide which options to pick before completing the various booking duties and sending confirmation, currency and travel documents. We then consider a particular role that this agent performs.

<p><b>AgentSchema for:</b> Eurescom Travel Agent</p>
<p><b>Identity:</b> The agent is identified by the task of arranging travel and its association with Eurescom.</p>
<p><b>Roles:</b> Travel requirements extractor role, Eurescom travel planner role, transport booker role, accommodation booker role, foreign currency ordering role, option provider role, and confirmation sender role.</p>
<p><b>Interactions with environment:</b></p> <p><i>Sensory input:</i></p> <p>User requirements and details, Eurescom travel guidelines database, transport information (timetables, prices, availability etc), accommodation information (prices,</p>

locations, availability etc), and list of contacts from whom to buy travel services.

*Acquaintances:*

See Figure 13.

*Resource ownership and access:*

1. Read-only access to Eurescom travel guidelines database and accommodation and transportation information.
2. Access to user's details such as grade is restricted for use only in conjunction with Eurescom travel guideline database.
3. Permitted access to means of guaranteeing payment, e.g. credit card number. This access is restricted for use with systems that meet a minimum security threshold defined by the user. Payment cannot be authorised without confirmation.

Access to systems to enable the electronic booking of travel services such as transport, accommodation and currency. The use of these systems is restricted such that all arrangements must be agreed by the user.

*Actions:*

- (Communicate) Get travel requirements from user (place, time, duration, preferences etc) and personal details (grade, credit card details, contact details, preferences, e.g. non-smoking, vegetarian, etc).
- (External) Finds Eurescom travel guidelines, which apply, to an employee with the user's grade (e.g. maximum cost etc).
- (Internal) Using task schemas, turn requirements into a series of tasks.
- (Internal) Constrain tasks based on users preferences and Eurescom travel guidelines.
- (External) Gather information on various options from information resources.
- (Internal) Construct and score best efforts (e.g. the ones that match constraints the best) based on the schema and constraints. Highlight where options break constraints.
- (Communicate) Suggest options to the user.
- (Communicate) Get choice from user.
- (Internal) Construct tasks in order to fully book users preferred choices.
- (Communicate/External) Book and pay for choices using contact details from information resources. These could be done via agents-to-agent interactions or via web pages.
- (Communicate) Raise any problems with the user and specify alternative choices.
- (Communicate) Send user currency requested, travel documents and confirmation.

**Mental state and behaviour:**

*Purpose:*

The agent's goal is to extract travel requirements from a user and to plan this travel,

by suggesting alternatives to the user. Such alternatives will be governed by constraints from the user's preferences and the Eurescom rules governing travel. Once the user makes the choice the agent makes the necessary arrangements and sends the user the relevant notification and documents.

*Behaviour:*

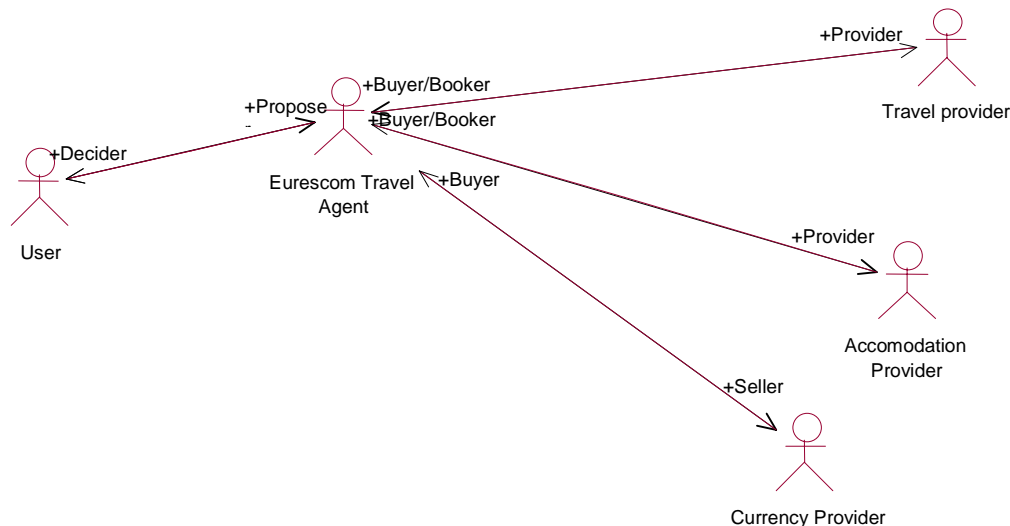
Its internal travel task schemas control the behaviour of the agent and the constraints put on this from the various sources. The agent is not able to fully arrange the travel without the user first confirming a choice.

*Knowledge and beliefs:*

- Knowledge of the grade structure in Eurescom.
- Set of travel task schemas.
- Knowledge of how to apply these schemas.
- Knowledge of how to access and use internal and external travel information sources.

**Additional Information:**

Note that the agent is not responsible for providing travel arrangements that satisfy Eurescom travel rules, but the agent must clearly show where these rules are being broken. The responsibility of satisfying these rules lies ultimately with the user, who chooses from the various options given. This allows the user to bend the rules a little, e.g. pay a little more for an upgrade, which can then be paid for personally or to argue a case with the financial director.



**Figure 13 Acquaintances**

<b>RoleSchema for:</b> Accommodation Booking Role
<b>Identity:</b> This role is characterised by the ability to book accommodation.
<p><b>Agent Requirements:</b></p> <p>An agent playing this role must be able to interface with the various electronic accommodation booking systems available.</p>
<p><b>Interactions with environment:</b></p> <p><i>Sensory input:</i></p> <p>An agent playing this role has the following additional information:</p> <ul style="list-style-type: none"> <li>• Access to accommodation information.</li> <li>• Access to accommodation booking systems.</li> </ul> <p>And the following restriction:</p> <ul style="list-style-type: none"> <li>• All accommodation requirements are restricted by the services which the booking systems can supply.</li> </ul> <p><i>Acquaintances:</i></p> <p>The acquaintances with other agents are modified by the fact that an agent playing this role can offer the accommodation booking service.</p> <p><i>Resource ownership and access:</i></p> <ul style="list-style-type: none"> <li>• Permitted access to means of guaranteeing payment, e.g. credit card number. This access is restricted for use with systems that meet a minimum security threshold defined by the user. Payment cannot be authorised without confirmation.</li> </ul> <p><i>Actions and tasks:</i></p> <p>An agent playing this role will acquire knowledge enabling it to perform:</p> <ul style="list-style-type: none"> <li>• Accommodation information gathering based on requirements, e.g. which system to book on.</li> <li>• Turning accommodation requirements into requests for booking systems.</li> <li>• Using various booking systems.</li> <li>• Collecting confirmation information.</li> </ul>
<p><b>Mental state and behaviour:</b></p> <p><i>Purpose:</i></p> <p>When an agent takes on this role its objectives are modified to include the goal that any accommodation booking requirements, which match the input restrictions, are achieved and confirmation information is collected.</p> <p><i>Behaviour:</i></p> <p>Agent behaviour is only modified in that it can now book accommodation and supply confirmation details.</p> <p><i>Knowledge and beliefs:</i></p> <ul style="list-style-type: none"> <li>• Knowledge of how to book accommodation based on an accommodation</li> </ul>

requirement.

- Knowledge of how to access and use accommodation booking systems.

**Additional Information:**

## 5.6 Guidelines

To some extent, an agent (or role) can be regarded as a point at which the goal, task/service and power relationship decompositions intersect. Thus there are three directions from which agent identification can be approached:

- goal decomposition: the system goals are analysed into sub-goals, and agents are built around groups of these
- task/service decomposition: the services that the MAS must provide are analysed, and agents are built around groups of services
- responsibility-oriented decomposition: agents are identified to handle relationships with external agents and resources. E.g. agents are identified to be responsible for the interests of particular users of the system, to manage specific resources, etc.

Alternatively one could analyse goals, services/tasks and responsibilities in parallel, then look for natural ways to cluster them. There is no 'correct' order in this process, and the analyst must use professional judgement. However it is arrived at, the end result must be self-consistent, and it must be possible from the OM and AM to trace how a set of agents collectively achieve the goals, provide the services, and handle the relationships of the system they implement.

The usual rule of thumb in top-down analysis is to decompose an entity into 3-5 constituents at a time. Thus, in cases involving many classes of agents it may be necessary to go through two or more stages of decomposition to arrive at a final set of agents. One judgement the analyst must make when performing the decomposition is whether an agent identified at an intermediate level of decomposition retains an identity when decomposed further, i.e. whether its constituents form a Multi-Agent System with a common purpose and organisational structure.

It is suggested that the agent schemas are filled out iteratively. In the early stage of analysis, basic information should be entered: the identity, purpose and main interactions and roles. The level of documentation at this stage is analogous to CRC cards. Agents identified during intermediate stages of decomposition may never get beyond this stage. Those that are realised as Multi-Agent Systems should eventually be documented according to the OM.

Roles should be introduced to identify chunks of responsibility or behaviour that are independent of an Agent's identity. Examples of circumstances where this is useful are:

- the system contains basically similar agents that differ detailed level due, for example, to them being responsible for different resources, users, or services;
- the responsibilities of an agent can change during its lifetime. This can be modelled as the agent playing different roles.

- agent's behaviour can be built up from building blocks. For example in a market-based Multi-Agent Systems, some may play selling roles, other buying roles, and some may both buy and sell.

## 6 Domain (Information) Model

### 6.1 Purpose of Model

A system under development (as whatever system) is going to deal with a specific domain. In particular the agents active in the system will have to deal, in some way, with entities that are instances of domain specific concepts. These entities are associated with Information Entities used in the other Models.

The purpose of the Domain Model is to point out the domain specific concepts that agents will need to deal with.

### 6.2 Structure of Model

The Domain Model is a model of the domain the System under development is going to deal with. The conceptualisation of this domain is assumed to be mixed object-oriented and relational.

- Object-oriented: all entities in the domain are classified in classes. Each class groups all entities with a common structure.
- Relational: a number of relations describe the mutual relationships between the entities belonging to the different classes.

The Domain Model therefore

- shows which *domain specific classes* agents need to know about.
- describes the structure of each class in terms of a number (possibly null) of *attributes*. Attributes have values that can belong to primitive types (such as integer or string) or can be instances of other domain specific classes.
- shows the *domain specific relations* that can possibly hold among the instances of the domain specific classes.

It has to be noticed that the modeled domain is assumed not to take into account the company that is going to use the System. Therefore, with reference to the example presented in section 2.2, the Domain Model will describe concepts like flight, city and hotel, and not concepts like TravelOffice or ProjectSupervisor that are on the other hand described in the Organisation Model.

### 6.3 Concepts

The following modelling concepts are used in the Domain Model.

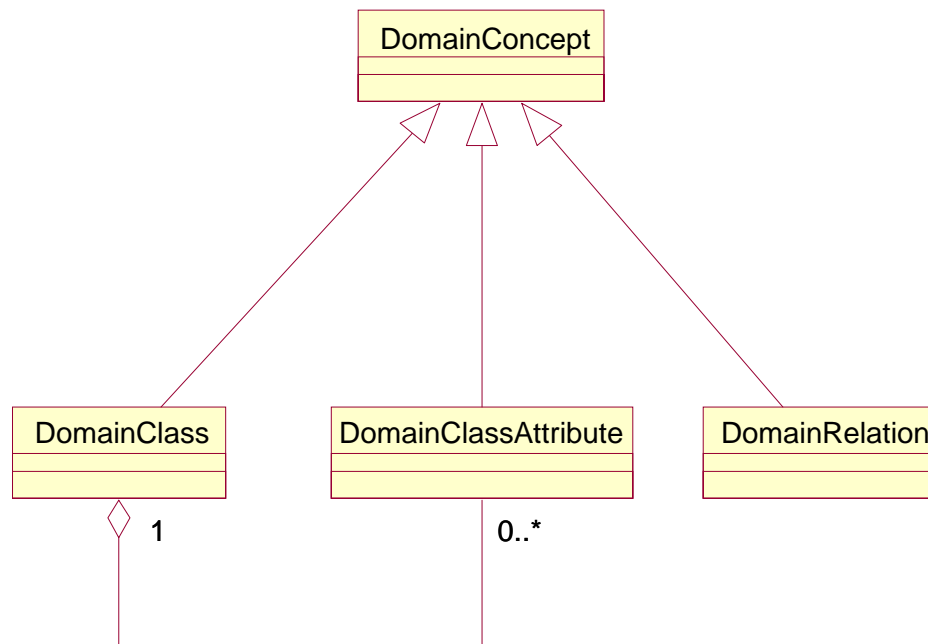
**DomainConcept:** the common ancestor of each domain specific concept.

**DomainClass:** each domain specific class is considered to be an instance of this modelling concept.

**DomainClassAttribute:** each attribute of a domain specific class is considered to be an instance of this modelling concept.

**DomainRelation:** each domain specific relation is considered to be an instance of this modelling concept.

The following class diagram summarises the modelling concepts used in the Domain Model.



**Figure 14 Modelling concepts used in the Domain Model**

## 6.4 Notations

The Domain Model can properly be represented by means of one or more UML class diagram as described below.

- Domain classes are represented as classes without operations.
- Aggregation and specialisation relationships can be used to detail the structure of Domain classes.
- Domain class attributes are represented as public attributes (for domain class attributes whose values belong to a primitive type) and association and aggregation roles (for domain class attributes whose values are instances of a domain specific class).
- Domain relations are represented as named associations.
- Constraints (expressed in OCL or more simply in natural language) can be used to further detail the meaning of DomainConcepts.

## 6.5 Example

The following class diagram depicts a simplified Domain Model for the system to be used in the EURESCOM Travel Office introduced in section 2.2.

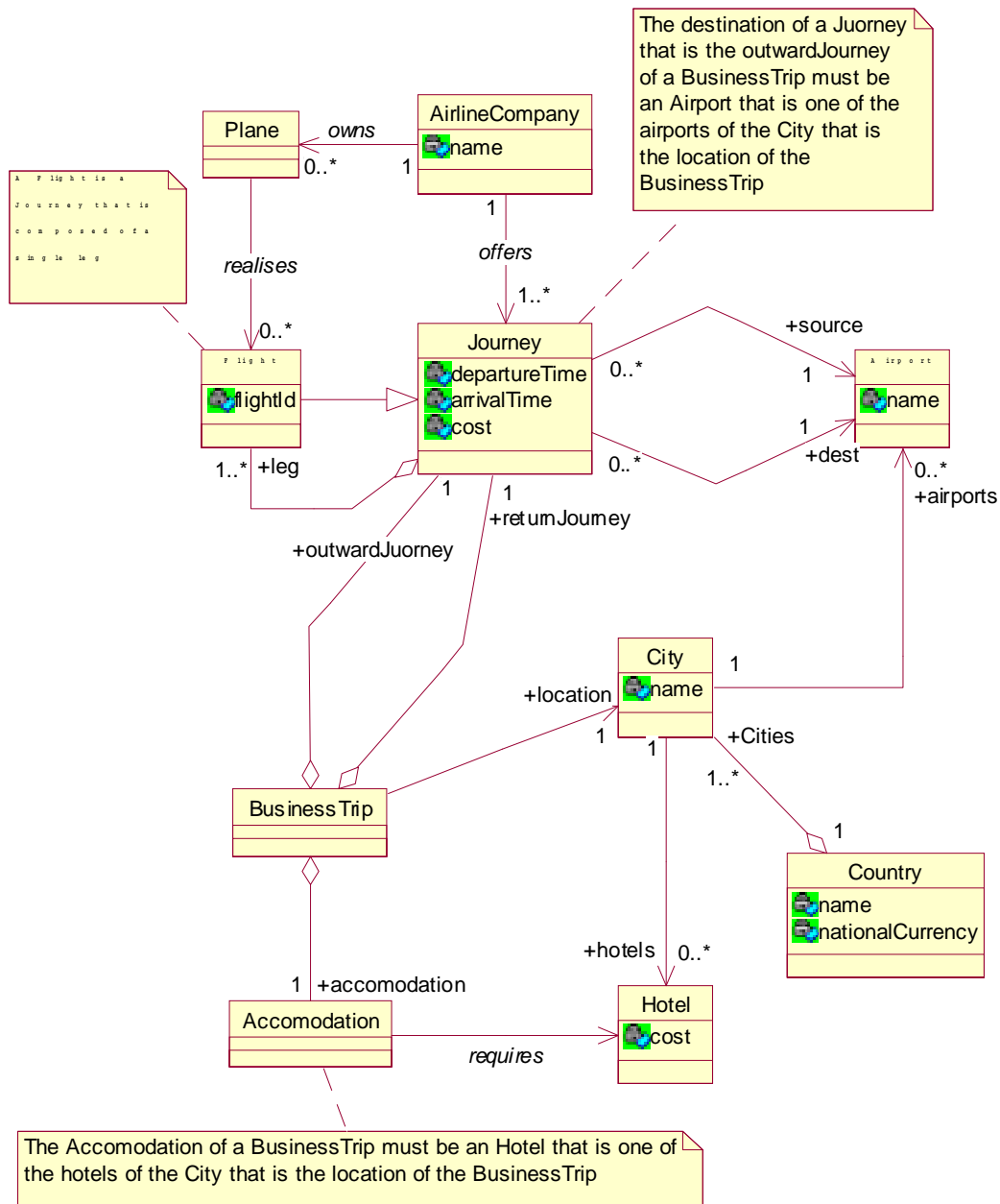


Figure 15 Example of Domain Model

## 6.6 Guidelines

The Domain Model is built in parallel to all other MESSAGE models. In fact new classes and relations are progressively added as the developer realises they are necessary/useful while he goes on in the construction of the other models. Important guidelines on how to build the Domain Model can therefore be derived from the construction of the other MESSAGE models.

Moreover the Common KADS methodology provides a number of heuristics and guidelines for the analysis of a domain. They will be analysed and possibly adopted by MESSAGE.

## 7 Interaction Model

### 7.1 Purpose of Model

The Interaction Model captures the way in which agents (or roles) exchange information with one another (as well as with their environment). The information contained in the interaction may be described in the Domain Model. The potential for complexity of Agent interactions is such that there is a clear need to be able to model interactions at different levels of granularity. The Interaction Model consists of a set of high level "Interactions" as well as a more detailed representation in terms of "Interaction Protocols".

The concept of Interaction is based on the Protocol from the Gaia methodology [3]. An Interaction can be viewed as "an exchange of information between entities with a purpose". This means that it does not represent a detailed pattern of exchanged messages, but is concerned instead with more abstract issues such as the purpose of the Interaction, which agents are involved and how it relates to the agent goals.

The more detailed interaction between agents can be modelled using the InteractionProtocol notation. This notation originated from work done in FIPA [4] and is currently under consideration for inclusion in UML 2.0. The InteractionProtocol notation is an extension to UML interaction diagrams with additional constructs to enable the modelling of richer interactions.

### 7.2 Structure of Model

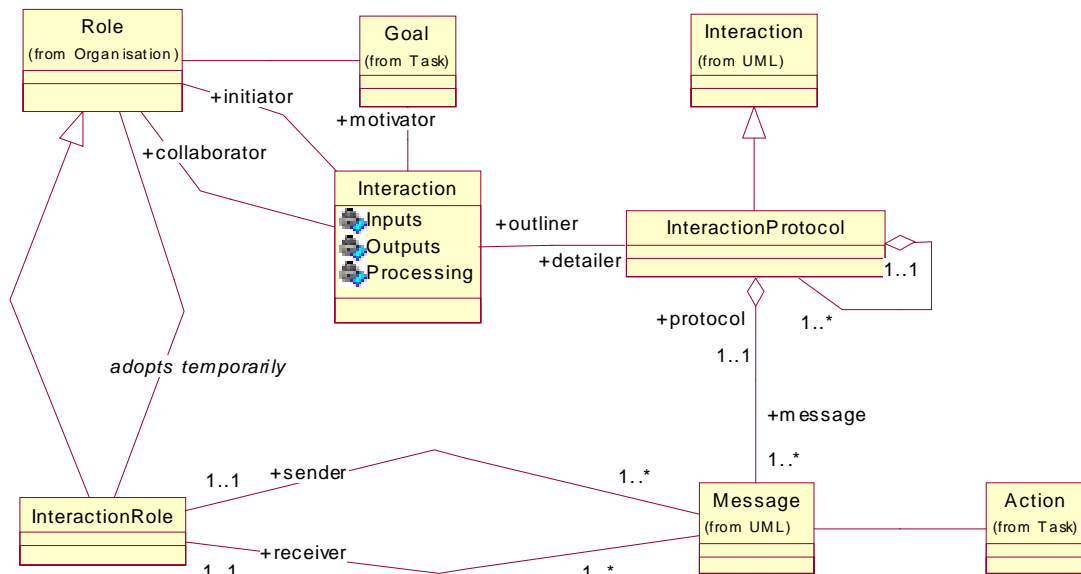
The Interaction Model is made up of a set of Interaction descriptions, a set of InteractionProtocol descriptions and of message descriptions.

The Interaction provides a high level description, while the InteractionProtocol specifies a protocol that fulfils the Interaction. InteractionProtocols contains a detailed specification of the messages interchanged between participating Agents as well as of the different allowable sequences in which messages can be interchanged.

InteractionProtocols can be nested within other InteractionProtocols so as to enable complex protocols to be constructed by an aggregation process. FIPA maintains a library of predefined, commonly required InteractionProtocols that are available as building blocks to build more complex InteractionProtocols. The overall structure is apparent from Figure 16.

### 7.3 Concepts

The Interaction, InteractionProtocol and message concepts were introduced previously as well as the way in which they are used to compose an Interaction Model. This section gives additional information regarding how the Interaction Model relates to other models as well as to concepts defined in the UML metamodel.



**Figure 16** Interaction Model Structure

**7.3.1 Interaction**

Inspired by Gaia Protocols [3], an Interaction expresses a high level pattern of message interchange between a number of agents. An Interaction is initiated by an Agent in order to achieve a particular goal. An Interaction also acts as an outline of a single InteractionProtocol which provides a set of detailed interactions that correspond to the Interaction. An Interaction has a single initiating agent as well as a set of participating Agents.

**7.3.1.1 Associations**

- detailer:** The InteractionProtocol that provides a detailed description of this Interaction.
- motivator:** The Goal that provides the motivation for the Interaction.
- initiator:** The Agent that initiates the Interaction
- collaborator:** The agents that are involved in the Interaction along with the initiator.

**7.3.2 InteractionProtocol**

An InteractionProtocol is a [4] "a communication pattern, with the allowed sequence of messages between agents having different roles, constraints on the content of the messages, and the semantics according to the semantics of the communicative acts."

**7.3.2.1 Associations**

- outliner:** The Interaction that specifies an outline of the protocol.

**message:** The sequence of messages that are contained within the protocol

### 7.3.3 InteractionRole

An InteractionRole is a type of Role. It defines a particular type of role that can be adopted by an Agent for the purposes of a particular InteractionProtocol. Examples of such roles might be buyer during a negotiation protocol. An InteractionRole can also be the source or destination for messages from or to other InteractionRoles.

## 7.4 Notations

The Gaia methodology [3] suggests a simple schema for specifying protocols. This is adopted here for the modelling of Interactions. The schema contains the following entries:

<b>Motivator</b>	<i>The Goal that motivates the Interaction</i>
<b>Initiator</b>	<i>The Role(s) responsible for starting the Interaction</i>
<b>Collaborators</b>	<i>The Roles with which the initiator interacts</i>
<b>Inputs</b>	<i>Information used by the Role initiator while enacting the protocol</i>
<b>Outputs</b>	<i>Information supplied by/to the protocol responder during the course of the interaction</i>
<b>Processing</b>	<i>A brief description of any processing that the protocol initiator performs during the course of the interaction</i>

**Table 5 Interaction Schematic Representation**

A notation for expressing InteractionProtocols is described in [4]. This is closely modelled on Interaction diagrams from the UML. An example of an InteractionProtocol diagram is shown in Figure 17.

## 7.5 Example

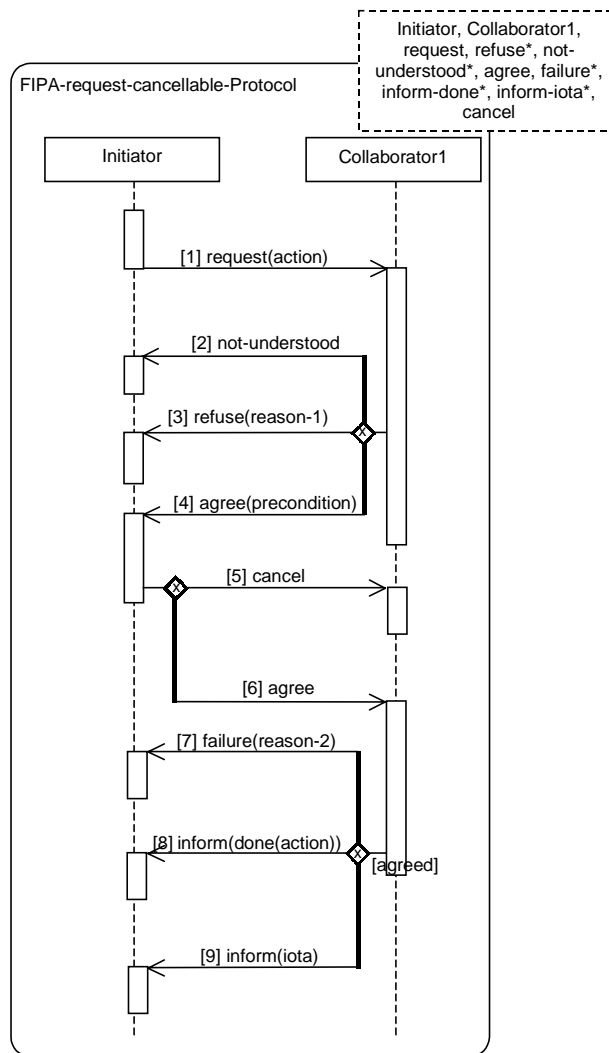
An example problem from the travel domain is discussed in [8]. As an example, suppose that a customer has booked a flight from a travel agent and wishes to cancel the booking. There may be a cancellation fee payable under these circumstances, in which case the customer may decide not to cancel the booking after all.

This Interaction Model is composed of a single Interaction specification, an InteractionProtocol diagram, and a set of message contents. The FIPA library contains the FIPA-request-cancellable protocol, which is ideally suited to this scenario.

The example model is adapted from that given in [8]:

<b>Motivator</b>	Previously booked flight has been cancelled
<b>Initiator</b>	A customer (Initiator)
<b>Collaborator</b>	The travel agent (Participant)
<b>Inputs</b>	The flight to be cancelled
<b>Outputs</b>	Conditions of cancellation (including cancellation charges)
<b>Processing</b>	Deciding whether or not to proceed with the cancellation

**Table 6 Example Interaction Schema**



**Figure 17 Example Interaction Protocol**

The detailed message contents are defined in schematic form (as below).

1	(request (action (takePayment paymentRef)) )
2	(not-understood)
3	(refuse (action (takePayment paymentRef)) (PTM-Exception)
4	(agree (action (takePayment paymentRef)) (:SETPaymentPrecondition paymentData))
5	(cancel)
6	(agree)
7	(failure (action (takePayment paymentRef)) (PTM-Exception)
8	(inform (done (action (takePayment paymentRef))))
9	(inform (iota))

**Table 7 Interaction Protocol Message Contents**

## 7.6 Guidelines

The process for constructing an Interaction Model naturally follows the structure of the metamodel. The first task is to complete an Interaction schema for the interaction. This involves identifying the Role that will initiate the interaction (initiator) and the main motivation for the interchange (motivator). This will often be a goal that the initiator wishes to satisfy. After that the developer should identify which other roles (Collaborators) will need to be involved in the interaction, what information will have to be supplied by the initiator (inputs) and by the collaborators (outputs). The final item to describe is any processing to be carried out by the initiator.

Having specified the Interaction in schematic form, the next stage is to specify a detailed InteractionProtocol that maps to the Interaction. Standard protocol definitions may already exist that can be reused to fulfil the outcome specified by the Interaction, FIPA is a source of such protocols. Otherwise a new protocol must be constructed, in which case the decisions to be made are the sequence of interchanges and any branches in the logic of the protocol. An InteractionProtocol diagram is then constructed to specify the InteractionProtocol.

Finally, the detailed syntax of each message in the InteractionProtocol needs to be defined in a schematic form.

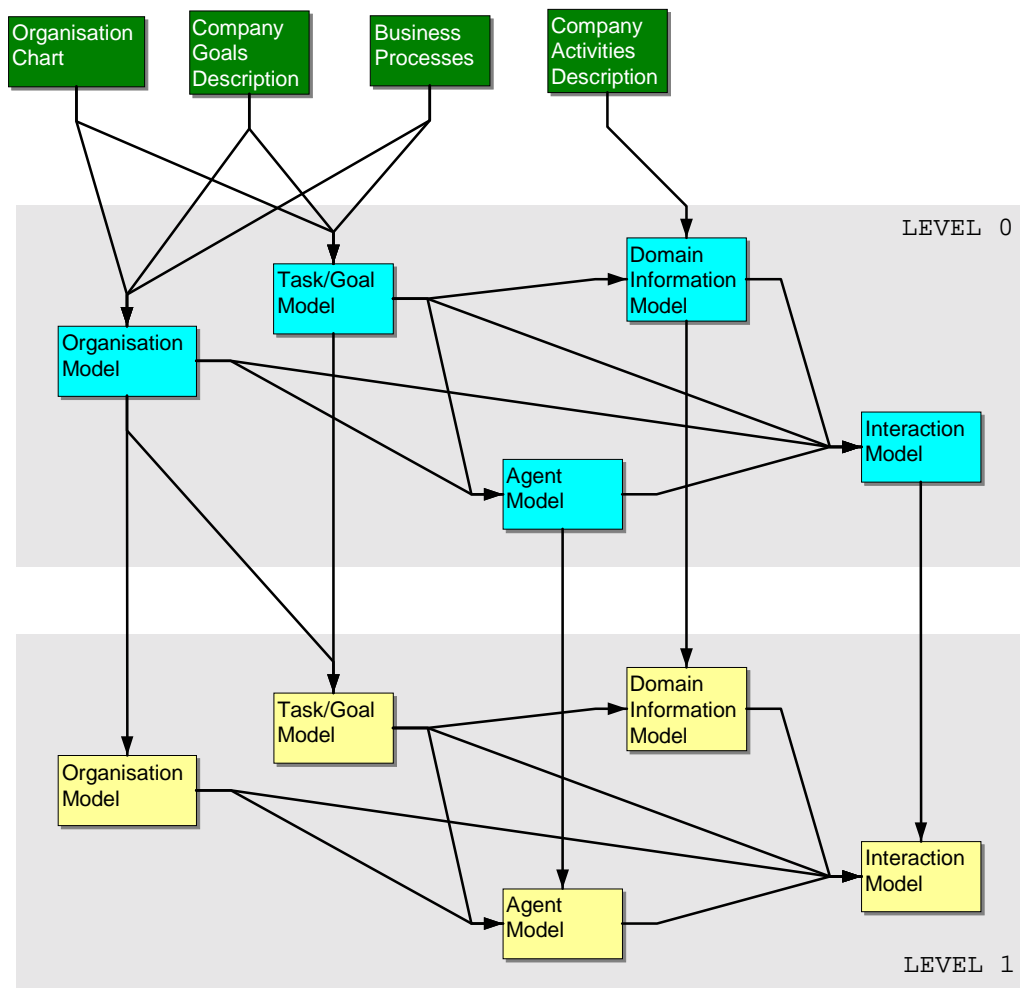
## 8 Workflows

This section describes the modelling process, with particular emphasis on the dependencies between the different models and the sequences in which the models are constructed. Sections 2.3 and 2.4 have outlined the way that the analysis models are structured into different levels and the overall order in which models are built. This is formalised into a workflow schematic as shown in Figure 18.

It is assumed that a set of the requirements will already have been defined for the MAS including functional requirements, user interface, performance, and operational, requirements.

The model components can be viewed along two dimensions. The different analysis models are arranged along one dimension, in the Figure 18 arranged horizontally.

The model also evolves along another "compositional" dimension as the modelling process proceeds to each level of successively finer compositional detail. In this diagram different levels of model are represented by a vertical arrangement. The modelling process can proceed to any number of levels. For clarity Figure 18 only shows Level 0 and Level 1.



**Figure 18 General Workflow schematic for MESSAGE Analysis activity**

## 8.1 Overview of Modelling Process

Section 2.3 describes the way that different levels within the overall model relate to one another. The basic concept here is that the Multi-Agent System (MAS) being constructed will appear from the viewpoint of an outsider to behave as a single agent. Internally, a MAS is composed of a number of Agents, each of which may in turn be a Multi-Agent System. No assumption is made from an outside viewpoint regarding the internal organisation of these groups of agents.

The top level of decomposition is referred to as Level 0. At this level the MAS is represented as a single agent. Subsequent stages of decomposition result in the creation of models at Level 1, Level 2 and so on. There must be consistency between subsequent levels. Thus the services offered by the Level 1 agents must 'add up to' the services provided by the Level 0 agents, and similarly goals and responsibilities must be accounted for.

This style of decomposition is mirrored in the way that business organisations are structured. A department within an organisation is made up of a number of individuals and yet behaves as a single entity.

## 8.2 Inputs to the Modelling Process

The first stage in the modelling process is creation of the set of Level 0 models. The main inputs to the modelling process are:

### **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.

## 8.3 Building the Level 0 models

The Level 0 model is built using this basic set of information. The first models to be built will generally be the Organisation Model and the Goal/Task model. These models then act as inputs to creating the Agent Model and the Domain Model. Finally the Interaction Model is built using input from the other models.

**Level 0 Organisation Model**

The Organisation Model aims at defining the structure and the behaviour of a group of agents (in the MAS and in the external organisation) working together to reach common goals. At Level 0 the Organisation Model provides a view of the organisational context within which the MAS operates in terms of Agents and the power relationships between them. The organisation chart is the main source of input for generating this model but inputs from the company goal description and the Business processes is also necessary.

**Level 0 Goal/Task model**

The Goal/Task model describes how high level goals (for example defining responsibilities of an Agent) are decomposed into lower level goals (e.g. ones that can be assigned to constituent agents). Similarly, it shows how high level tasks (e.g. services provided by an Agent) can be decomposed into sets of sub-tasks (e.g. services that are provided by the constituent agents).

At level 0 the Goal/Task model represents the context in which the MAS operates. It contains a complete set of goals and tasks of the organisation in which the MAS is situated. The task sub tree is augmented with workflows that show how the individual tasks fit together to form a complete process.

The inputs to creating the Goal and tasks sub trees are the Company goals description, the Organisation chart and the Business processes.

**Level 0 Agent Model**

The Agent Model consists of descriptions of the purpose, relationships, behaviour, and other attributes of individual agents and of roles. The Level 0 Agent Model models the MAS system as a whole as if it were a single Agent. Input to creating the Level 0 Agent Model comes from the Goal/Task model and the Level 0 Organisation Model. A key decision to be taken is assignment to the MAS of specific Goals and Tasks from the Goal/Task Model, in effect agents are built around groups of services that carry out the required tasks. A complementary approach is responsibility-oriented decomposition: in which agents are identified to handle relationships with external agents and resources. E.g. agents are identified to be responsible for the interests of particular users of the system, to manage specific resources, etc. This process effectively defines the boundary of the MAS in terms of responsibility.

**Level 0 Domain model**

The Domain Model (DM) captures Information about the problem domain. In the example scenario the DM would capture entities and relationships such as ticket, airline, and airports. The company activities description is a good source of information for building the DM. The Goal/Task is also useful in this regard.

**Level 0 Interaction Model**

An Interaction Model captures the way in which agents (or roles) exchange information with one another (as well as with their environment). The Level 0 Interaction Model defines the set of interactions between the MAS and the external organisation in which the MAS is situated. It requires input from all of the other four Level 0 models.

## 8.4 Building the Level 1 Models

### Level 1 Organisation Model

The Level 0 Organisation Model is decomposed further to yield the Level 1 Organisation Model. Decomposition will yield an internal structure for the MAS in terms of a set of agents and associated organisational relations between those agents.

### Level 1 Goal/Task Model

At Level 1, the Level 0 Goal/Task Model will typically be decomposed to a greater level of detail. The Level 0 Organisation Model provides information to help in this decomposition. The level of decomposition must be such that tasks and goals exist that can be assigned to agents as specified in the Level 1 Organisation Model.

### Level 1 Agent Model

The Level 0 Agent Model is decomposed into a set of Agents that as a group support the same external interfaces and behaviour as the Level 0 model. The services offered by the Level 1 agents must 'add up to' the services provided by the Level 0 agents, and similarly goals and responsibilities must be accounted for. The set of agents to be modelled is derived from the Level 1 Organisation Model. Other inputs to the modelling process are the Level 0 Agent Model and the Level 1 Goal/Task model.

### Level 1 Domain Model

The Level 1 Domain Model *extends* the Level 0 Domain Model with additional constructs. Modelling at increased levels of detail (particularly within the Level 1 Goal/Task Model) will often turn up lower level domain constructs. It should be noted that the relationship with Level 0 is not a decomposition relationship as in the case of the other models.

### Level 1 Interaction Model

The Level 1 Interaction Model defines the set of interactions of all of the Level 1 Agents with one another as well as with the external organisation in which the MAS is situated. It requires input from all of the other four Level 0 models as well as the Level 0 Interaction Model.

## 9 Guidelines

This section is intended to provide additional advice on using the methodology in addition to the Workflow description.

Although the MESSAGE methodology specifies in some detail the components of a model, the process by which the model is created is far less exact. The workflow diagram should be taken as an approximate guide as there may be a wide variation in the characteristics of different development projects. Factors that will vary from one project to another include:

- The type of organisation the system must operate within
- The number of different types of Agent within the system
- The total number of Agents within the system
- The availability of documentation on the problem domain
- The extent to which interfaces to legacy systems are required

The impact of these various factors will be to favour one of three approaches for starting the modelling process.

Looking at the Workflow diagram in Figure 18, it can be seen that either the Organisation Model or the Goal/Task model could be constructed first. The Goal/Task Model contains a Goal tree and a Task tree, either of which could be modelled first. In addition, the Domain Model could be started first. Although the Domain Model is dependent on the Goal/Task model, some projects may even benefit from beginning the modelling process on the Domain Model.

There are thus three main options when beginning to model at the analysis phase this issue was initially discussed in section 5.6. The approaches can be described as 1) goal based decomposition, 2) task based decomposition 3) responsibility oriented decomposition.

Goal based decomposition (described in section 4.5) is the approach used in the KAOS methodology. Goal decomposition starts with the overall goals of the organisation in which the Multi-Agent System operates. A single high-level goal for the Multi-Agent System is then derived which is then progressively decomposed to form a tree of sub-goals. Responsibility for attaining individual goals within the sub-tree can then be assigned to specific agents within the Multi-Agent System.

The Task based approach begins with a description of the overall task that has to be carried out and then progressively decomposes this into simpler and simpler tasks. Modelling will then follow by assignment of tasks to specific agents.

The Goal and Task decomposition approaches are similar in that they first consider what has to be done before looking at who is responsible for doing it. There are subtle differences however.

The Goal Based approach is more abstract and tends to focus the analyst on underlying reasons for the system operating as it does. Adopting this approach will tend to defer decisions on the mechanics of how the system operates and the sequence in which tasks are carried out. It will tend to be particularly useful where there is a lot of freedom to choose how the system goes about achieving its objectives. The goal-based approach will usually be most appropriate where clearly defined goals can be formulated for the Multi-Agent System.

The Task based approach tends to become focussed on procedural issues earlier than the Goal based approach. This approach will often work well where there is a clear definition of the tasks to be carried out by the organisation in which the Multi-Agent System is situated. A typical example of this situation is where the operation of the organisation is regulated by a workflow management system.

The third approach is that of responsibility oriented decomposition. Here the structure of the Multi-Agent System is based around the lines of responsibility within the organisation. Responsibility will often be based on control of resources or interfaces to users.

This type of modelling approach will often be the most appropriate where the Agent system must interface to or "wrap" legacy components such as databases. It will also often be a good approach where there is a rigid adherence to a particular reporting structure within the organisation.

There will be in general less variation in the modelling process during the creation of Agent Model and Interaction Model and so we won't discuss this in detail.

It is generally advisable to complete a set of Level 0 Models before attempting any modelling at Level 1. This should proceed by *building the Level 1 Models in the same order that the Level 0 Models were constructed.*

## 10 Conclusions

This document represents the first deliverable (D1) from the MESSAGE project P907. It has presented an initial version of a methodology for developing Agent Oriented Software. The main recommendations contained in this document were:

1. A description of a set of Models for Agent system Analysis
2. A description of a process for building these Analysis Models.

The recommended methodology was restricted to the Analysis activity so as to enable early dissemination of project results. This is intended to provide an opportunity for feedback from shareholders in time to influence later releases of the methodology. The document has not concerned itself with later activities in the development process, i.e. the design, implementation and testing activities. Deliverable D3 is intended to present a more complete and detailed methodological description.

This document began with an overview, describing the motivation for having a methodology for Agent Oriented Software development, describing what an agent oriented approach means and outlining the type of applications where an agent oriented approach might be appropriate. This section concluded with a perspective of how the recommendations made in this document fit in to a complete Agent Oriented development methodology.

The document then introduced the Analysis Models, describing the MESSAGE modelling language and its relationship with UML, and showing how each Model addresses a different aspect of the Multi-Agent System. An example problem was also presented which was used to illustrate the various modelling approaches throughout the document. There was also a brief discussion of the analysis process itself.

The five analysis Models were then each described in detail with a chapter on each. Each Model was described in terms of its purpose, its internal structure, core concepts, and modelling notations. Finally, a simple example was presented for each Model to illustrate the methodology in action.

The following Models were presented:

Domain (Information) Model, the Organisation Model, the Goal/Task Model, the Agent Model and the Interaction Model. The process for developing the Models was then presented in some detail. This was described in terms of the required modelling activities, the sequence in which the Models are constructed and the inputs to each modelling activity.

Finally, preliminary guidelines were given as an accompaniment to the process description to give additional information on how to follow the process successfully.

## 11 References

- [1] OMG Unified Modeling Language Specification Version 1.3, June 1999, Object Management Group, Inc.  
<http://www.rational.com/uml/resources/documentation/index.jtmpl>
- [2] OMG Meta Object Facility (MOF) Specification:  
<ftp://ftp.omg.org/pub/docs/ad/99-09-04.pdf>
- [3] The Gaia Methodology for Agent-Oriented Analysis and Design, Wooldridge, M., Jennings, N. R., Kinny, D., Kluwer Academic Press
- [4] Response to the OMG Analysis and Design Task Force UML 2.0 Request for Information: Extending UML for the specification of Agent Interaction Protocols, OMG 16/12/1999, Bauer, B. et al. <ftp://ftp.omg.org/pub/docs/ad/99-12-03.pdf>
- [5] Shoham AOP paper
- [6] Goal-directed requirements acquisition, Dardenne et al, Science of Computer Programming, Vol. 20, 1993
- [7] Knowledge Engineering and Management: The CommonKADS Methodology, Schreiber et al, pub. Bradford Books, 1999
- [8] Experiences in the use of FIPA agent technologies for the development of a Personal Travel Application, Donie O'Sullivan, Jorge Núñez-Suárez, Henri Brouchoud, Patrice Cros, Clair Moore, Ciara Byrne, Proceedings Agents 2000, Barcelona, June 2000
- [9] Analysis and Design of Multiagent Systems using MAS-CommonKADS, Carlos Iglesias, Mercedes Garrijo, Jose Gonzalez and Juan R. Velasco, Intelligent Agents IV: Agent Theories, Architectures and Languages, 1997, M. P. Singh, Anand Rao and M. J. Wooldridge, eds. Lecture Notes in Computer Science 1365
- [10] *Extending UML for Agents*, James Odell, H. Van Dyke Parunak, Bernhard Bauer, submitted paper, 2000. This paper describes how UML can be used and extended to Model agent systems. available for download from <http://www.jamesodell.com/ExtendingUML.pdf>
- [11] *The Rational Unified Process: An introduction*, Philippe Kruchten, pub. Addison Wesley.
- [12] Ian Sommerville, "Software Engineering", Addison Wesley, 3<sup>rd</sup> Edition 1989.
- [13] P. Bertrand, R. Darimont, E. Delor, P. Massonet, A. van Lamsweerde  
*GRAIL/KAOS: an environment for goal driven requirements engineering*  
Proceedings ICSE'98 - 20th International Conference on Software Engineering, IEEE-ACM, Kyoto, April 98

## 12 Glossary

Acquaintances	An indication of the other agents that the agent can interact with to obtain information and to request to perform actions/achieve goals. The relationships with these agents must also be described
Action Concept	<p>An action is an atomic unit of functionality to modify the state of the system and the environment.</p> <p>An action is defined in terms of</p> <ul style="list-style-type: none"> <li>• A mandatory name;</li> <li>• a mandatory <i>informal definition</i>;</li> <li>• an optional <i>pre-condition</i> defining the state that must be true before the task can be performed;</li> <li>• an optional <i>post-condition</i> defining the state that will be true after the task will be completed.</li> </ul> <p>Action inputs are Model Elements that are processed by the action. Action outputs are updates of the input Model Elements plus any new Model Element produced by the action.</p>
Additional information	The analyst may optionally provide additional information not covered by the above headings.

Agent	<p>The purpose of this definition is <b>NOT</b> to have a criterion on the basis of which it is possible to state whether a given entity (a software module, a living being, a robot....) is or is not an “agent”. On the other hand it has to be considered as <i>a preliminary list of the features that will be ascribed to the entities that the developer will choose to consider as agents.</i></p> <p>The main features in the definitions are</p> <ul style="list-style-type: none"> <li>• An agent has certain <i>knowledge</i> of the world it lives in.</li> <li>• An agent is responsible for achieving/maintaining certain <i>goals</i> that drive its behaviour.</li> <li>• An agent is able to <i>observe</i> the status of certain objects in the environment and to sense certain <i>events</i>.</li> <li>• Mutual interactions between agents are described in terms of <i>communicative actions</i> that <ul style="list-style-type: none"> <li>– Have an intrinsic well-defined <i>semantics</i> and</li> <li>– Embed a content expressed in a given <i>content language</i> and containing terms whose meaning is defined in a given <i>ontology</i></li> </ul> </li> <li>• Besides communicative actions, an agent can perform <i>direct actions</i> that affect properties of objects in the environment.</li> </ul> <p>It should be noted that Mobile Agents are considered outside the scope of MESSAGE.</p>
Agent characteristics	<p>An agent has an identity, means of interaction with its environment (sensors and effectors), and a means of deciding what to do based on objectives (goals or utility functions), i.e. a means of translating purpose into behaviour given a perceived state of affairs.</p>
Agent decision	<p>An agent decides what tasks to perform on the basis of his objectives and knowledge and beliefs. An agent decides to perform actions that will change the state of objects in such a way that the goals it is committed to are satisfied.</p>
Agent Model (AM)	<p>Consists of descriptions of the purpose, relationships, behaviour, and other attributes of individual agents and of roles. Each description gathers together information on an agent or role from other Models and adds internal detail. A role is essentially a position that may be filled by a qualified agent.</p>

Agent perception	an agent perceives its environment through Information Entities that describe Events or information on the state of the world. The Information Entities could result e.g. from observation actions by the agent, or receipt of a message from another agent. The perception of new information triggers tasks and actions that update the agent's knowledge and beliefs, which may be thought of as a collection of Information Entities constituting the agent's model of the world.
Agent reaction	The agent performs the actions that he decided to perform. Information Entities are the input and the output of actions.  The distinction between agent and role is that an Agent Model describes characteristics that are inherent to an agent, whereas a role describes characteristics that an agent takes on because it is, for example, doing a particular job.
AgentSchema	PIR2.1 identifies the main characteristics of an agent as being its identity, its means of interacting with its environment, and a mental state (goals, knowledge and beliefs, etc.), and reasoning and decision-making capability. The form of the last part is somewhat dependent on the style of Agent Model being followed. The internal behavior of an agent can be described as follows: an agent interacts with its environment and perceives events, it decides how to react to these events, and then reacts to them. These three steps of the internal behavior of an agent are described hereafter in terms of the agent metamodel:
Behaviour	A description of how an agent's behaviour is modified as a result of playing this role.
Collaborator	The agents that are involved in the Interaction along with the initiator.
Commitment Relationship	The commitment ternary relationship is defined between two agents and a goal. An agent has the obligation to establish the goal that is expected by another agent.. An agent may participate in one or several commitments. An agent that has a commitment must be capable of fulfilling it. This means that he must have the knowledge, or the ability to acquire it, and have access to Model Elements that are needed to fulfil the commitment.
Control entity	This concept characterises the way in which conflicts and control mechanisms are defined in the organisation. Taxonomies of control classes such as conflict resolution entities, norms and laws may be used at the domain level for illustrating the control policy of the organisation.
DecomposeGoal Relationship	The DecomposeGoal meta-relationship is defined between a goal and a set of goals. A Goal may be decomposed into sub-goals such that the satisfaction of the sub-goals entails the satisfaction of the parent goal. The decompose meta-relationship can only be formally shown to hold if each of the argument goals has a well-formed formalDef. Attribute value.

DecomposeTask Relationship	Tasks may be decomposed into sub-tasks. The DecomposeTask relationship is defined between a single Task and a sequence of sub-tasks. The sequence of sub-tasks defines the composition of the sub-tasks that needs to be respected for the sub-tasks to decompose the parent task. The pre-condition of the first sub-task should be implied by the parent tasks pre-condition, and the post-condition of the last sub-task should imply the parent's post-condition.
detailer	The InteractionProtocol that provides a detailed description of this Interaction.
Domain (Information) Model (DM)	Captures Information about the problem domain. In the example scenario the DM would capture entities and relationships such as ticket, airline, airport, flies to.
DomainClass	Each domain specific class is considered to be an instance of this modelling concept.
DomainClassAttribute	Each attribute of a domain specific class is considered to be an instance of this modelling concept.
DomainConcept	The common ancestor of each domain specific concept.
DomainRelation	Each domain specific relation is considered to be an instance of this modelling concept.
Goal Concept	<p>A goal is a set of states of the world that an agent is committed to achieve/maintain. Therefore a goal is a situation, but not all situations are goals. A set of states of the world is not in general a goal unless there is an agent committed to achieve/maintain this set of states.</p> <p>A goal is defined in terms of by means of</p> <ul style="list-style-type: none"> <li>a mandatory informal definition</li> <li>An optional formal definition.</li> <li>a goal structure defined in terms of sub-goals composing the goal</li> </ul> <p>A goal describes an objective for an agent<sup>9</sup>, e.g. as a consequence of playing a particular role. Informally, a goal is a state of affairs to be achieved, maintained, or avoided; or a utility function to be maximised.</p>
Goal/Task Model (GM)	describes how high level goals (for example defining purposes of an organisation) are decomposed into lower level goals (e.g. ones that can be assigned to constituent agents). Similarly, it shows how high level tasks (e.g. services provided by an organisation) can be decomposed into sets of sub-tasks (e.g. services that are provided by the constituent agents). Note that a goal describes motivation (e.g. a need to be in a particular location at a particular time), while a task describes activity (e.g. catching a flight to that destination).

<sup>9</sup> Note that a Multi-Agent System is itself formally an agent.

Identity	A property or characteristic that uniquely distinguishes an individual entity.
Information Entity	<p>Agents perceive Information Entities. These carry two main classes of information:</p> <p>Descriptions of Events. Objects describing Events inform the agent that something has happened;</p> <p>Information about the state of entities in the Agent's world. Objects of this type inform the agent about, for example the state of some resource parameter.</p> <p>There are Events corresponding to arrival and sending of messages. The act of Agent A sending a message to Agent B gives rise to two Events, corresponding to the message being sent by Agent A and the message arriving at Agent B some time later. Messages will often contain Information Entities in their content.</p>
initiator	The Agent that initiates the Interaction
IntendsToAchieve	An agent intends to achieve its goals (from the definition of goal). An agent's purpose is its primary goal, and goals are also implied by roles that the agent plays. In addition, an agent may adopt goals dynamically, e.g. because they are tactical sub-goals of its purpose, because they are delegated to it by an agent in authority, or because of an agreement with another agent. Goals undertaken on behalf of another agent are termed commitments. Commitments imply obligations, e.g. to inform the other agent if the goal cannot be achieved. An agent should only intend/commit to a goal if it has the capability to achieve it.
Interaction	Inspired by Gaia Protocols [3], an Interaction expresses a high level pattern of message interchange between a number of agents. An Interaction is initiated by an Agent in order to achieve a particular goal. An Interaction also acts as an outline of a single InteractionProtocol which provides a set of detailed interactions that correspond to the Interaction. An Interaction has a single initiating agent as well as a set of participating Agents.
Interaction Model (IM)	Describes interactions between agents and the protocols that implement them. Essentially, an Interaction defines at high level a means of agents influencing each other, e.g. negotiating the terms of a service. An example of Interaction is the dialog between a PS and the travel office that captures the PS' requirements regarding a particular business trip. The Interaction identifies the common goal of the participants (to define the travel requirements), the initiator (the PS), etc. There will be an exchange of information, questions asked, etc. as the details are gathered, but this is not important at the level of abstraction of the Interaction. Where it is necessary to prescribe an order to messages, this can be done via Protocols.

InteractionProtocol	An InteractionProtocol is [4] ”a communication pattern, with the allowed sequence of messages between agents having different roles, constraints on the content of the messages, and the semantics according to the semantics of the communicative acts.”
InteractionRole	An InteractionRole is a type of Role. It defines a particular type of role that can be adopted by an Agent for the purposes of a particular InteractionProtocol. Examples of such roles might be buyer during a negotiation protocol. An InteractionRole can also be the source or destination for messages from or to other InteractionRoles.
Interactions with environment	What the agent can perceive, what it can change, and what it can do. It consists of the following components: Sensory input and Acquaintances, Resource ownership and access, Actions and tasks.
IsInfluencedBy	A type of power relationship between agents indicating the power to change the internal state of an agent by another agents or one persuades others agents through negotiation to perform certain task and/or to send information.
IsSubordinateTo	(reverse: manages). This indicates a ‘line management’ manager-subordinate relationship. CommonKADS ([7]) refers to this as a ‘formal power relationship’.
Knowledge and beliefs	Main elements of the agent’s mental state and inferencing capabilities are modified as a result of playing this role.
message	The sequence of messages that are contained within the protocol
Model Element	Defined in the UML specification [1]
motivator	The Goal that provides the motivation for the Interaction.
Organisation Model (OM)	focuses on the structure of organisations and the relationships among the agents that they contain <sup>10</sup> . It also describes conflict resolution mechanisms, norms, etc, that enable the group of agents to function as a unit serving a common purpose.
Organisation relations	An organisation relation denotes dependencies between organisations such as subsidiary of divisions, departments, etc.

<sup>10</sup> An organisation is not an agent: An organisation is a collection of agents with a common purpose. More than just gathering the agents, it gives a structure to the collection of the agents, putting restraints on their behavior (through rules).

Organisation Structure	This meta-class characterises the components of the organisation and their relationships. Organisation Structures are defined as graphs where the nodes are made up of agents, and/or roles, and/or other organisation structures. An Organisation Structure may have assignation links to resource entities. Agents in an Organisation Structure may play different roles or collaborate with other agents to play organisation roles. Agents may have power relationships between them. This relationship illustrates control dependencies and information reporting.
Outliner	The Interaction that specifies an outline of the protocol.
Organisation Structure Schema	This contains roles in the organisation and the agents. There are two types of agents User interface agents and Information agents. One or more information agent <b>is subordinated to</b> the User agent.
Power relationship	This concept denotes the degree of influence agents (playing roles) have over each other. They allow us to construct management hierarchies (i.e. who is the boss of whom). isInfluencedBy is the weaker form indicating an indirect line of authority.
Purpose	A goal or utility function that is central to identity of entity and which guides its decisions and actions.
Purpose description	This includes the textual description of the mission of the organisation
Resource	A resource is non-agent entity that is used by or provides information to the greater system. In the Agent Model, we are concerned with what attributes of a resource an agent can see, and what actions an agent can perform on it. In some applications, ownership of or responsibility for a resource may be assigned to an agent, with ownership special access to information and resource functions. Other agents may then need to access the resource via the owning agent. Access to a type of resource may be a pre-requisite for performing certain types of task. A resource may also be the object of goals, for example a goal to maintain a certain property of a resource within bounds.
Resource ownership and access	A description of the resources for which the agent has special responsibility and access. It should describe what the agent is able to do and also restrictions on what it is permitted to do.
Roles	An indication of what roles the agent is able to play. This can be omitted if roles are not used in this application.
Satisfaction Relationship	The satisfaction relationship is defined between a Task or a Model Element and a Commitment. The relationship means that the Task pre- and post-conditions of the Model Element invariant make the Commitment true.

Schema and Field	In terms used in the UML metamodel, names ending in 'schema' and 'field' denote PresentationElements (sometimes referred to as a view element), <i>which is an element whose purpose is to provide a presentation of information for human comprehension</i> [1]. Removing the suffix 'Schema' yields the name of the ModelElement of which the schema is a 'presentation'. A field is division within a schema.
Sensory input	Establishes the information the agent has direct access to over and above those described in the acquaintances and resource ownership and access sections. It includes a description of the Events and other Information Entities the properties of which the agent is able to observe directly. It should indicate the degree of detail of information that is visible to the agent.
Service	A service is a task that an agent is (potentially) willing to perform on behalf of other agents
Task Concept	<p>A <b>task</b> is an action that can be decomposed into sub-tasks and actions; it inherits all relationships and attributes. It has a <i>task structure</i> defined in terms of sub-tasks composing the task. However, for a collection of sub-tasks to constitute a task, it is normally necessary for them to satisfy some ordering constraints (e.g. performing task A is equivalent to performing sub-task A1 and then sub-task A2). In order to formalise such ordering relationships between tasks, the following relationships between a set of events and an event are introduced. Enables: {A} enables B means that once one event in the set {A} has occurred it is possible for B to occur. precedes: {A} precedes B means that B cannot occur unless all the events in the set {A} have occurred.</p> <p>Next, the idea is introduced that (labelled) events are 'emitted' are various stages in the execution cycle of a task. It would be useful to define a standard set of such labels with well-defined significance (start, end, abort, suspend, etc.). Furthermore, (sub-)tasks may not be able to proceed until a specified event has been 'observed'. Suppose {ei} is the set of end events of a collection of tasks A, and s is the start event of another task, B. Specifying { ei } enables s means that B can start once one of A has finished. Specifying { ei } precedes s means that B can start once all of A have finished. A network of tasks with start and end events linked by enables and precedes relationships can readily be displayed using a UML activity diagram, with the precedes concept represented by a synchronisation bar.</p>
Workflow Structure Schema	This includes the task and the results to be produced by the tasks. The type of tasks to be performed by the MAS, and the results to be produced are defined in the task taxonomy, and the result taxonomy. Production relations link tasks to results