Chapter VIII

The MESSAGE Methodology for Agent-Oriented Analysis and Design

Francisco J. Garijo
Telefónica I+D, Spain

Jorge J. Gómez-Sanz
Universidad Complutense de Madrid, Spain

Philippe Massonet
CETIC, Belgium

Abstract

This chapter presents the MESSAGE agent-oriented software engineering methodology and illustrates it in an analysis and design case study. The methodology covers MAS analysis and design and is intended for use in mainstream software engineering departments. MESSAGE integrates into a coherent AOSE methodology basic agent-related concepts (such as organisation, role, goal, interaction, and task) that have so far been studied largely in isolation. The MESSAGE notation extends the UML with agent knowledge-level concepts and diagrams with notations for viewing them. The proposed diagrams extend UML class and activity diagrams. The
MESSAGE analysis and design process is based on the Rational Unified Process (RUP). The methodology distinguishes high-level from detailed design. An organisation-driven approach is presented for detailed design, where the global architecture of the MAS is derived from the structure and behaviour of the organisations that interact. Experimentation with the methodology shows that it supports the analysis and design of multi-agent systems that are flexible and adapt to specific kind of changes in a heterogeneous and dynamic environment.

Introduction

The agent-oriented (AO) approach promises the ability to construct flexible systems with complex and sophisticated behaviour by combining highly modular components. When several of these components interact, the intelligence of these components—the agents—and their capacity for social interaction results in a multi-agent system (MAS) with capabilities beyond those of a simple “sum” of the capabilities of the individual agents.

The availability of agent-oriented software engineering (AOSE) methodologies is a pre-requisite for engineering commercial MASs. The MESSAGE methodology covers the MAS analysis and design phases of the software engineering lifecycle. It is designed for use in mainstream software engineering departments that develop complex distributed applications. It is a genuinely agent-oriented methodology, but also builds upon the achievements of software engineering (SE), and is consistent with current SE best practice. MESSAGE grounds agent-oriented concepts in the same underlying semantic framework as that used by UML, and uses UML-based notation whenever appropriate.

MESSAGE (Caire, Courlier et al., 2001, 2002) extends UML by contributing agent knowledge-level concepts, such as organization (Zambonelli, Jennings, & Wooldridge, 2001), role (Kendall, 2000), goal (Dardenne, Lamsweerde, & Fickas, 1993; Giorgini, Kolp, Mylopoulos, & Pistore, 2004) and task (Omicini, 2000), and diagrams with notations for viewing them. The work is illustrated in a complete agent case study going through the phases of analysis and design.

The plan of the chapter is as follows. The next section describes the MESSAGE modelling language and process, followed by a section that describes the analysis case study and illustrates some of the graphical notation proposed. The high-level design case study is then presented. Next, we illustrate the organization-driven detailed design process. The final section presents an evaluation of MESSAGE based on analysis, design, and implementation experimentations.
The MESSAGE Methodology

The MESSAGE methodology has adopted the Rational Unified Process (RUP) (Kruchten, 1999) as a generic software engineering project lifecycle framework. This allows MESSAGE to incorporate the body of software development knowledge – life cycles, activities, tasks, notations in UML (such as Use Case Models, Class Diagrams, Interaction Diagrams, State Transition Diagrams), and supporting UML tools - which can be applied for agent-oriented development. MESSAGE follows the iterative and incremental approach of RUP that involves progressive refinement of the requirements, plan, design, and implementation. It focuses on the elaboration phase where the core system architecture and high-risk factors need to be identified. The architecture is elaborated using a collection of agent-related models and graphical notations. Specific guidelines for the creation and stepwise refinement of these models during analysis and design are also provided. In summary, MESSAGE extends UML with new agent-related concepts and graphical notations and enriches and adapts the RUP process model for agent-based analysis and design. MESSAGE also includes guidelines and examples on how to apply the methodology (Milgrom et al., 2001).

The MESSAGE refinement process is described in Figure 2. MESSAGE analysis results in a collection of models describing the system to be developed and its environment. The following five views on the models have been defined to help the modeler focus on coherent subsets of the multi-agent system: Organization, Goals/Task, Agent/Role, Interaction, and Domain. Similar divisions can be found in MAS-CommonKADS (Iglesias, Mercedes Garijo, Gonzalez, & Velasco, 1998), Vowel Engineering (Demazeau, 2001) and GAIA (Wooldridge, Jennings, & Kinny, 2000).

At this point we may remark on the distinction between view and model in this methodology, which differs from MASSIVE (Lind, 2001). A view in MASSIVE and MESSAGE could be understood as a projection of the complete model onto a particular subject. A model is a container for the concepts that specify that subject. MASSIVE considers that there is only one model whereas MESSAGE requires several. There is, for example, one Organization Model, but many views of this model that show different parts of the same concept. A view can be an acquaintance diagram showing the acquaintance relationships among different agents in the organization, or a task workflow diagram showing how functionality is distributed among different actors within an organization.

The Organization model describes the overall structure of the MAS. It shows concrete entities (Agents, Organizations, Roles, Resources) in the system and its environment and coarse-grained relationships between them (aggregation, power, and acquaintance relationships). The Goal/Task model describes the goals the MAS and its constituent agents try to attain and the tasks they must accomplish in order to satisfy the goals.
The Agent/Role model focuses on the individual agents and roles. For each agent/role, it uses schemata supported by diagrams to describe its characteristics, such as the Goals for which an agent is responsible, the events it needs to sense, the resources it controls, the Tasks it knows how to perform, “behaviour rules,” and the like. The agent model contains a detailed and comprehensive description of each individual agent and role within the MAS.

The Interaction model is concerned with agent interactions with human users and other agents. For each interaction among agents/roles, the initiator, it shows the collaborators, the motivator (generally a goal the initiator is responsible for), the relevant information supplied/achieved by each participant, and the events that trigger the interaction, together with other relevant effects of the interaction (e.g., an agent becoming responsible for a new goal).

The Domain (information) model shows the domain-specific entities and relations that are relevant for the system under development.

The above models contain the data that defines the system and that is represented in the views. The MESSAGE meta-model is defined as an M-2 Layer meta-model using the MOF four-layer architecture (OMG, 2000). The MESSAGE meta-model provides the concepts, the underlying structure, and the rules for producing application models.

Figure 1 shows the main agent-related meta-concepts (Caire, Coulier, et al., 2002; EURESCOM P907, 2003), for example, an Agent is an atomic autonomous entity that is capable of performing some (potentially) useful function. In these diagrams, each concept is related to others using labeled associations, denoting the semantics of the association.

The MESSAGE concepts extend UML concepts by means of the MOF generalization primitive. For instance, the agent concept extends the MESSAGE Entity that extends the UML classifier concept.

During design, the analysis models are refined into computational entities that can be implemented in an agent platform. Two distinct phases of design are considered: high-level and detailed design. The first phase involves identifying an agent architecture, assigning roles to agents, and describing how the application services are provided in terms of tasks. The tasks can be decomposed into direct actions on the agent’s internal representation of the environment, and communicative actions to send and receive messages according to interaction protocols. The interactions between roles identified in analysis are detailed using interaction protocols. Detailed design deals with computational representations of the entities modelled in high-level design. During this second phase, the refinement process continues to determine how entities can be implemented. In MESSAGE, two approaches are considered: organization-driven and the agent platform-driven design. This process is illustrated in Figure 2.
Figure 1. Agent-related MESSAGE meta-concepts

Figure 2. MESSAGE modelling process description
MESSAGE Concepts and Notation

The MESSAGE modelling language is based on RUP notations that are, in turn, based on UML. MESSAGE extends the basic UML concepts of Class and Association with knowledge-level agent-centric concepts and modelling views (Caire, Coulier, et al., 2001). The most important agent-related concepts are the following.

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

- **Organization**: An Organization is a group of Agents working together to a common purpose. It is a virtual entity in the sense that the system has no individual computational entity corresponding to an organization; its services are provided and purpose achieved collectively by its constituent agents. It has structure expressed through *power relationships* (e.g., superior-subordinate relationships) between constituents, plus behaviour/coordination mechanisms expressed through Interactions between constituents.

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

- **Goal**: A Goal associates an Agent with a state. If a Goal instance is present in the Agent’s working memory, then the Agent intends to bring about the state referenced by the Goal.

The main types of activity are:

- **Task**: A Task is a knowledge-level unit of activity with a single prime performer. A task has a set of pairs of Situations describing pre- and post-conditions. If the Task is performed when a pre-condition is valid, then one can expect the associated post-condition to hold when the Task is com-
pleted. Composite Tasks can be expressed in terms of causally linked sub-tasks (which may have different performers from the parent Task). Tasks can be modelled as state machines, so that, for example, UML activity diagrams can be used to show temporal dependencies of sub-tasks.

- **Interaction and Interaction Protocol**: An Interaction is characterized by more than one participant and the purpose that the participants collectively must aim to achieve. The purpose typically is to reach a consistent view of some aspect of the problem domain, to agree to terms of a service, or to exchange results of one or more services. An Interaction Protocol defines a pattern of MESSAGE exchange associated with an Interaction.

The MESSAGE notation is based on UML. Agent-related concepts are defined as level 2 meta-models using OMG’s meta-model Facility, MOF (OMG, 2000). The organization model, for example, is defined as an M-2 Layer meta-model using the MOF four-layer architecture. One of the practical advantages of this approach is that UML tools, which will help engineers to create, manage, and reuse object-oriented application models, can interpret MOF meta-models to provide tool support for analysis and design using MESSAGE.

Figure 3 shows the graphical syntax of some of the above-mentioned level-1 concepts. The Assignment meta-relationship is specialized into “Play: Agent x Role,” “Wish: Agent x Goal,” “Perform: Agent/Role x Task,” “Participates: Role x Interaction,” “Implements: Task x Service,” and “Provides: Agent/Role/Organization x Service.”

*Figure 3. MESSAGE graphical notations*
Analysis

The purpose of analysis is to produce a system specification (or analysis model) that describes the problem to be solved (i.e., the requirements). It is represented as an abstract model in order to (1) understand the problem better, (2) confirm that this is the right problem to solve (validation), and (3) facilitate the 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). MAS analysis focuses on defining the domain of discourse and describing the organizations involved in the MAS, their goals, and the roles they have defined to satisfy them.

High-level goals are decomposed and satisfied in terms of services provided by the roles. The interactions between roles that are needed to satisfy the goals are also described. The analysis models are produced by stepwise refinement.

Analysis Process

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

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

In level 1, the structure and the behaviour of entities such as organization, agents, tasks, and goals domain entities are defined. Additional levels might be defined for analyzing specific aspects of the system dealing with functional requirements and non-functional requirements, such as performance, distribution, fault tolerance, and security. There must be consistency between subsequent levels.

Several strategies are possible for refining level 0 models. Organization-centred approaches focus on analyzing overall properties, such as system structure, the services offered, global tasks and goals, main roles, and resources. The agents
needed for achieving the goals appear naturally during the refinement process. Then, cooperation, possible conflicts, and conflict resolution may be analyzed. Agent-centred approaches focus on the identification of agents needed for providing the system functionality. The most suitable organization is identified according to system requirements. Interaction-oriented approaches suggest progressive refinement of interaction scenarios that characterize the internal and external behaviour of the organization and agents. These scenarios are the source for characterizing task, goal, messages, protocols, and domain entities.

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

The experience in MESSAGE shows that the different views of the system leave the analyst free to choose the most appropriate strategy. In practice, a combination of refinement strategies with frequent feedback loops are used. The analysis process might start with the Organization View (OV), then switch to the Agent View and continue with the Interaction View. The results of the analysis of specific interaction scenarios may lead to the reconsideration of part of the OV and thus to further refinement and adaptation of OV constituents.

Design

The purpose of design is to define computational entities that represent the MAS appearing at the analysis level. In general, the artifacts produced in each of the analysis models need to be transformed into computational entities that can be implemented. Analysis entities are thus translated into subsystems, interfaces, classes, operation signatures, algorithms, objects, object diagrams, and other computational concepts.

High-Level Design Process

The design process consists of a series of iterative activities for transforming analysis models into design artifacts. The transformation process is highly dependent on the level of abstraction of the analysis entities. As analysis agents
are defined in generic terms, during design it may be necessary to refine them into one or more computational entities. There are situations in which agents identified during analysis are not implemented in design. This happens when analysis agents are refined into simple classes in design, and when agents from the analysis correspond to very well-known software components such as access systems, notification servers, database wrappers, proxies, and so on. In such situations, it is better to consider analysis agents as resources. On the other hand, analysis agents will be implemented in design when the complexity of the refinement requires an agent architecture.

MESSAGE distinguishes between high-level and detailed design to keep the models implementation independent and thus avoid taking into account the complexity of the specific concepts and constraints of a target agent platform, such as the agent architecture and the knowledge representations.

In high-level design, the analysis model is refined to produce the initial version of the MAS architecture and its computing behaviour defined with design artifacts. Four steps are proposed:

1. Assigning roles to agents. Organization roles identified in analysis should be assigned to agents in order to assess their computational tractability. Design decisions are strongly dependent on the designer’s views and experience, although heuristic criteria might be applied, for example, assigning to an agent a single complex role, grouping similar functional roles and assigning them to an agent.

2. Providing services with tasks. If any analysis workflow diagrams in the Organization Model (OM) have been defined, they can be further refined in this stage. The refinement process should specify the relationships between services and tasks, tasks and goals, and goals to roles.

3. Refining the interaction protocols. This step involves refining the interactions identified in analysis. Interactions can be modelled in terms of interaction protocols (Odell, Parunak, & Bauer, 2001) and UML state-charts. This modelling takes into account the interactions between roles, the assignment of agents to roles, and the implementation of services in terms of tasks, direct actions, and communicative actions.

4. Interaction role behaviour specification with statecharts. This step involves modelling the behaviour of the roles in an interaction protocol. The result will establish relationships between the interaction model, the agents behaviour model, and the overall functionality of the organization.
Detailed Design Process

The low-level design process assumes that the developer is thinking about possible implementations. This process implies consideration of different mappings, from high-level design concepts to computational elements provided by the target development platforms. By computational, we mean having an application program interface with an externally known behaviour. These elements may already exist, for example, as a software library, or will need to be developed from scratch. The MESSAGE methodology has been evaluated with two complementary design approaches.

The first design approach is driven by the MAS organization. The design process is driven by the Organization Model in order to assign responsibilities, to define agent interactions, and to model social knowledge.

The second approach is agent-platform-oriented and considers that each agent can be mapped to a class. This is mainly derived from the application of most agent models supported by agent building tools, such as Jade (Bellifemine, Poggi, & Rimassa, 2001), in which there is one agent class from which to derive the specific agent type.

Organization-Driven Detailed Design Process

The Organization Model provides a high-level view of the system with all the elements needed for structuring its computational entities. The organization itself may or may not be present as a computational element at the end of the design. The Organization Model shows which roles need to interact, their communication needs, and what social knowledge is required from each agent in order to satisfy the system goals, that is, the organization goals. Individual realization of tasks or service provision is easier to design than collective realization.

The steps proposed to achieve organization driven design are as follows:

- Defining the multi-agent system architecture. The Organization Model defines a preliminary architectural framework where the principal computing entities are the agents. The agents’ environment is made up of the rest of the computing entities, which are used by agents or have direct or indirect relationships with them. Design packages are structured according to the OM: Tasks, Goals, Resources, Workflows, and Agents within Organization Structures, each of which might be refined separately. They might be directly translated into computational units. The Organization Model imposes constraints upon agent relationships, common tasks, and global goals.
Figure 5 shows the acquaintance relationships in the level 0 organization diagram. The KM system interacts with two roles, the System Administrator and the Salesperson, and with two external systems (resources), the Travel Database to retrieve travel arrangements and the Booking Database to insert the bookings requested by salesperson on behalf of Travelers. Moreover, it interacts with the Administrative Team to prepare the bills that will be sent to travellers. A Salesperson interacts with Travelers to gather travel requirements and provide travel arrangements. It should be noted that the Salesperson does not interact directly with the Travel Database and the Booking Database. All these interactions are carried out through the KM system.

Goals, Roles, and Services

Organizations have high-level goals that the roles need to satisfy by providing and requesting services. The organization goals can be decomposed using goal decomposition techniques into goals that can be directly satisfied in terms of the services that are offered by a role. For example, the “Traveler Assisted” goal of the traveller is partially satisfied by the goal “TA Gathered” of the “TA Gatherer” role, together with its goals “Best TA identified” and “TA Selected” of the “TA Selector” role. The services that need to be provided to satisfy these goals are “TA Gathering” and “Best TA Selection.”
These constraints should be satisfied under behavioural changes and organizational dependencies.

- Selecting an agent architecture. Agents are designed as instantiations of a specific kind of agent architecture, whose complexity depends on the roles that have been assigned to the agents in the organization and the kind of relationships with other agents (e.g., whether interactions involve complex protocols or not). In MESSAGE, there have been experiments with cognitive (BDI agents) and reactive architectures (state-machine-based agents).

- Specifying the agent’s behaviour and interfaces. This includes the agent’s domain and social knowledge. This is defined using the structure and the relationships of the Organization Model. It supports reasoning about other agent’s actions, about the society itself, and the social constraints upon an agent’s actions (Garijo, Gómez-Sanz, Pavón, & Massonet, 2001)

- Using conventional software engineering modelling techniques can help to detail internal agent architecture behaviour. For instance, sequence diagrams can be used to clarify interactions, activity diagrams to model the sequence of states reached by agents when performing tasks, and use cases to detail the expected functionality of the final system.

- Defining the agent society infrastructure. The infrastructure consists of available resources, dependence relationships among agents, capabilities of other agents, or assigned tasks. We have called those computing entities that are not agents and are used by the agents to obtain information for achieving their objectives “resources.” Examples of resources are: databases, protocol stacks, text-to-speech translators, speech processors, visualization systems, syntactic analyzers, and the like.

The Organization Model supports designers in rationalizing and working in parallel on the refinement work. It also guides the refinement process and helps to maintain consistency among analysis entities, design entities, and system requirements.

**Agent-platform Detailed Design**

Another approach to the transition from high-level design to low-level design is to define a mapping between the implementation-independent MESSAGE analysis/design concepts and the target implementation-dependent concepts. A case study on the transition (Massonet, Deville, & Neve, 2002) between a MESSAGE design and a FIPA-compliant agent implementation toolkit was carried out with the JADE framework (Bellifemine et al., 2001) and the Jess
are several teams in the TSP. At level 0, the system under development, namely, the KM System, is seen itself as an organization that will be analyzed at level 1.

**Goal**

The goal of the TSP is to assist the traveller. This could be understood as the TSP arranging a restaurant, renting a room in a hotel, providing information on tourist places, and so forth. This goal is very generic and will be refined in level 1. Level 0 is kept simple in order to clearly separate the first idea of the system from its refinement.

**Domain**

The domain can be conveniently described using UML class diagrams to model the classes, their associations, and their attributes. Examples of classes that model domain entities and relationships are: “Travel Requirement” (TR), which is composed of a set of “Transfer Requirements” (TsR) that specify an origin, a destination, and a time frame; a “Travel Arrangement” (TA), which is composed of a set of “Transfer Arrangement” (TsA) that refers to a “Flight Occurrence.” A flight occurrence refers to a flight from an origin to a destination on a specific date. A TA can match a TR if all TsR match the TsA.

*Figure 4. Organization diagram view (structural relationships) (notation as in Figure 3)*
Delegation structure diagrams, workflow diagrams, and textual agent/role schemata are useful for describing the Agent/Role view. The term structure is used to emphasize the fact that it is an extended class diagram that describes the structure. UML provides an aggregation relation (diamond-headed line) to express a relationship between a composite entity and its constituents. However, it is rarely the case that the parent-constituent relationship is simply one of aggregation. The structural framework into which the constituents must fit is usually important. The structure concept is introduced in order to describe such frameworks in a reusable way. In the case of a goal/task diagram, it expresses how a goal of an organization is decomposed into, which are then assigned to the Organization’s constituents.

A delegation structure diagram shows how the subgoals obtained decomposing a goal of an organization are assigned to the agents/roles included in the organization. Clearly this diagram is strictly related to (and must be consistent with) both the goal decomposition diagram showing the decomposition of the organization goal and the organization diagram showing the agents/roles inside the organization.

Figure 6 shows a delegation structure diagram. Only the root and the leaves of the decomposition of the parent organization goal are shown. Similarly, a workflow diagram shows the roles in an organization that must perform the tasks necessary to implement a given service provided by the organization. An example of this type of diagram is shown later in Figure 9.

*Figure 6. Delegation structure diagram view (notation as in Figure 3)*


Interactions

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

The interaction view is typically refined through several iterations as long as new interactions are discovered. It can be conveniently expressed by means of a number of interaction diagrams. These diagrams are interaction centric (i.e., there is one such diagram for each interaction) and show the initiator, the responders, and the motivator (often a goal of the initiator) of an interaction, plus other optional information such as the trigger condition and the information achieved and supplied by each participant. Figure 7 shows as an example the interaction diagram describing the Traveling Request interaction between the TA Gatherer and the TSP Assistant roles.

The details of the interaction protocol and the messages that are exchanged between roles can be represented using an AUML sequence diagram (Bauer, Müller, & Odell, 2001). The modelling of an interaction protocol is on the border between analysis and design. The interaction can be detailed either in analysis or in design. The analysis stops here, but more iterations could be made.

Figure 7. Interaction diagram view (notation as in Figure 3)
rule-based system for reasoning (Friedman-Hill, 2002). It was illustrated on a subset of the case study described in the next section and showed how the high-level agent concepts of the analysis and design modelling language could help structure the agent implementation that is usually based on a simpler set of agent concepts. The case study showed that high-level design decisions were easier to make using MESSAGE than using the implementation-dependent concepts. This approach is not described further in this chapter.

### Analysis: Travel Agent Case Study

This section shows how the development process and notation can be applied to the case study. The first step should be to identify the level 0. This level requires identifying first the organization involved and the goals it pursues.

- **Context:** traveling from one location to another involves creating a travel plan with a very tight schedule. It might involve taking a taxi from one’s home to the airport, taking a flight to an intermediate location, taking a connecting flight to the final destination where a rented car has been booked and can be picked up to drive to the hotel where reservations have been made. Unfortunately for the traveler, many things can go wrong with a travel plan.

- **Requirements:** Given the fact that many travellers will soon have wireless terminals, the efficiency of the traveling process can be improved by developing a system (distributed both in these terminals and in the terrestrial network) that:
  - gathers travel requirements from the traveller;
  - assists in identifying and arranging relevant travel services offered by the travel service providers;
  - assists with the booking of travel tickets; and
  - monitors that the travel arrangement is carried out as planned by providing alerts and notifications of changes to arranged travels.

### Organizations

Figure 4 describes structural relationships in a level 0 organization diagram. The diagram shows that the Knowledge Management (KM) system is owned by the Travel Sales Person system (TSP). A Salesperson is part of a team and there...
Implementing Services with Tasks

Once the agents have been assigned to roles, the services can be implemented in terms of tasks. Figure 9 shows the workflow of tasks that is needed for the PTA agent to provide the “TA selection” service to the traveller. The input to the “Get TAs” task of the “TSP Assistant” role is a TR using the object flow UML notation. The output is a set of TAs that is sent to the “TA Gatherer” role, which then passes them to the “TA Selector” role to rank them.

Interaction Protocols

This step involves refining the definition of interactions made during analysis. Based on the interactions between roles, the assignment of agents to roles, and the implementation of services in terms of tasks, direct actions, and communicative actions, the interactions can be modelled in terms of interaction protocols and UML statecharts. Figure 10 shows how the interaction between the “TSP Booking Manager” and the “Airline Booking” roles can be modelled with a FIPA
request interaction protocol. A TA is passed as the content of the request message.

**Interactions Roles Defined with State Machines**

This step involves modelling the behaviour of the roles in an interaction protocol. Figure 11 shows how the “TSP Booking Manager” role’s behaviour can be modelled for the request interaction protocol. When a booking request is refused or not understood, it is diagnosed and a decision is made to either cancel the request or retry the request.

*Figure 11. TSP booking manager state chart*
High Level Design Example

The transition from analysis to high-level design requires identifying the agents for achieving the system functionality, deciding their computing model, and specifying the behaviour of the overall organization in computational terms. To achieve these goals, we will use the four step method that was presented previously in the high level design section. Examples of the four steps are described in the following sections.

Identifying Agents and Assigning Roles

Agents are identified based on the description of the organizations and the use of some heuristic: at one extreme, for each organization there can be one agent, assigned all the roles in the multi-agent application; at the other extreme, there is one agent per role.

In this case study, a personal travel agent (PTA) was created for the traveller, a TSP agent was created for the TSP organization, and an airline agent was created for the airline organization. Assigning roles to agents has an impact on the design of the interactions. For example, since the “TSP Sales Assistant” and “TSP Booking Manager” need to interact and are played by the same agent, they do not need to interact using interaction protocols. Figure 8 shows the TSP agent

Figure 8. Agent diagram view (notation as in Figure 3)
Organization-Driven
Detailed Design Example

At the detailed design stage, MESSAGE considers two approaches: the agent-platform-driven and the organization-driven. An agent-platform-driven design (Massonet et al., 2002) is based on the mapping between the high-level design concepts and the target platform. However, this may not be possible when the agent platform is not known in advance or when the agent platform does not satisfy the computing requirements of the target computing infrastructure. This section focuses on the organization-driven approach.

Organizational Model

The Organization Model defines the architectural framework for achieving design activities. Figure 12 shows the system architecture as an MAS organization. The package structure is derived from the organization meta-model.

Once the organization has been modeled, the developer can proceed with selecting an agent architecture, detailing the specification of the agent’s behaviour and defining the agent society’s infrastructure.

Figure 12. Organization-based architecture
Selecting an Agent Architecture

Selection criteria are dependent on the functionality to be achieved but also on other non-functional criteria such as adaptability, robustness, performance, security, and so forth. One solution experimented with in MESSAGE consists of defining a family of design patterns in UML, which are instances of the metamodels and incorporate computing models derived from previous engineering experiences (Garijo, Tous, Matias, Corley, & Tesselaar, 1998; Walshe et al., 2000). Design models for agents are component-based (OMG, 2002; Szyperski, 1997) and cover the more popular styles of agent architecture, such as reactive and cognitive architecture. These elements belong to conventional software engineering and their application towards the construction of an agent architecture could be further discussed. However, this knowledge is very useful for evaluating agent architectures that are available in the agent literature, such as Garijo, Bravo, Gonzalez, and Bobadilla (2003), Wooldridge and Jennings (1995), Huhns and Singh (1997), and Weiss (1999). These architectures can also be evaluated with the insight of agent patterns, such as the mediator pattern or the bidding pattern (Do, Kolp, Hoang, & Pirotte, 2003; Kolp, Do, Faulkner, & Hoang, 2004).

Figure 13. Cognitive agent pattern
Agent components provide an external view that is made up of their external interfaces and a uniform internal structure (Figure 13). There are two types of interfaces: (1) agent management interfaces providing methods to activate, stop, delete, and monitor the agent; and (2) an agent communication interface to allow agent interaction with different entities that can be either agents using agent communication languages (ACL) or computing entities in the environment (for example, Web servers, data bases, call managers, messaging servers, mail servers, directory services, and so forth).

The agent’s internal structure is formed by the necessary subsystems to perform the perception-assimilation-control-act cycle. These subsystems are defined as internal components encapsulating an agent’s specific functionality and communicating through standard interfaces. A great variety of agent models can be obtained, depending on the characteristics of their perception mechanism, their control process or their actuation model.

The personal user agent (PUA) architecture is based on the Cognitive Agent pattern (Figure 13) implementing a BDI cognitive processor. The rationale for this choice is the need for reasoning and inference mechanisms to gather travel requirements from the traveller and to provide advice and guidance in identifying and arranging relevant travel. Once the architectural model is selected, the next step is to define the knowledge needed to achieve the agent functionality.

**Detailed Specification of Agent Behaviour**

PUA behaviour specification consists of defining the corresponding knowledge in each package of the agent architecture:

- **Domain knowledge.** It is obtained by refining the corresponding analysis model classes.
- **Objectives.** PUA objectives are expressed with *and/or trees* of goals. Each goal corresponds to either an analysis goal or a refinement of an analysis goal.
- **Tasks and actions.** They represent methods or procedures to obtain the information required to achieve the objectives. Actions are computing primitives that allow the agent to get new beliefs by different mechanisms such as inference processes, acting over the environment assimilating incoming information, and communicating with other agents.
- **Strategy and decision making.** The decision model is declarative. Strategy and tactic rules might be defined for controlling the generation of objectives, choosing the most suitable tasks to achieve a particular objective, and
changing the focus of the resolution process by suspending the resolution of an objective and selecting a new one.

- Specific diagrams showing the relationships among goal, tasks, actions, and facts, which might solve the goals, are defined. These diagrams are easily translated into rules in the implementation phase.

An example is shown in Figure 14. It is interpreted as follows: The goal GatherNotificationDataG will be satisfied when valid notification data has been obtained. To get this notification two subgoals might be solved: CreateVisualResourcesNotificationDefinitionG and GatherNotificationDataAndValidateG.

**Defining the Agent Society Infrastructure**

This consists of describing the organization resources. Resources are computing entities that are not agents but that are used by the agents to obtain information for achieving their objectives. Agents can use resources based on resource patterns, which offer standard interfaces as shown in Figure 15. This facilitates the management, use, and deployment on the different processors.
Experiences and Evaluation of MESSAGE

The MESSAGE methodology was evaluated in two case studies. The case studies were conducted in two phases: Analysis and Development, that is, Design and Implementation. An assessment of the methodology based on this experience was then performed. Two case study scenarios were selected for Analysis:

- Universal Personal Assistant for Travel (UPA4T): a task-specific instantiation of a generic personal assistant application; and
- Adaptive Customer Service OSS (ACSOSS): a decentralized OSS application performing end-to-end coordination within a customer service business process.

The UPA4T case study was carried through to design and to different partial implementations.

MESSAGE’s agent-centred concepts proved to be a sufficiently complete set of construction primitives for the case study problems. Using the views of the system as building patterns helps developers obtain a complete specification of the MAS. The case studies confirmed that the methodology supports the analysis and design of multi-agent systems that are autonomous and can be deployed in a dynamic and heterogeneous environment.

MESSAGE has been the basis for new proposals, such as INGENIAS (Pavón & Gómez-Sanz, 2003), which refines the different MESSAGE meta-models and adds more tool support, and RT-MESSAGE (Julian & Botti, 2004), which adds extensions to deal with real-time constraints.
Further experience in Telefonica I+D applying MESSAGE to develop telephony-based conversational services has led to the development of an agent-based library and support tools for component creation, retrieval, management and reuse. The BOGAR_LN library (Garijo, Bravo et al., 2003) provides application developers with four categories of reusable component models: Agent Organization models, Agent models, Resource models, and Basic computing entities. Evaluation experiments showed that using MESSAGE and the BOGAR_LN library components allows substantial reduction in development time and effort (65% less). Cost reduction is achieved without minimizing or bypassing activities like design, documentation, and testing.

**Strengths and Focus of the Methodology**

The methodology supports the analysis and design of multi-agent systems that are flexible and adapt to specific kind of changes described below in a heterogeneous and dynamic environment. The case studies showed that MESSAGE supports architectures where:

- Agent instances of defined agent classes can come and go during the lifetime of a multi-agent system. This is modelled in the interaction and agent models by describing the roles that participate in an interaction and the agents capable of playing those roles.
- Services come and go during the lifetime of an MAS. This is modelled by describing which organizations/roles/agents provide a given service. However, the precise ways in which a published service can be changed during the MAS’s lifetime depend on the specifics of each agent toolkit.
- Agents can change the way they provide a service. This is modelled by separating the description of the service from the way it is implemented in terms of tasks. The service could be provided in different ways using an inference engine that optimizes certain criteria. If it has been designed to do so, an agent could even cooperate with other agents to help them provide the service that was previously provided alone during the lifetime of the system.
- Agents communicate in an environment with information heterogeneity. This is handled by a shared ontology for the MAS. This shared ontology is derived from the domain model that was captured during analysis. Agents then communicate using the common ontology to which the internal heterogeneous information of the different agents can be mapped.
• Agents display goal-oriented behaviour and thus adapt their behaviour for continuous optimization of design-time and run-time goals. This is possible by modelling which tasks an agent is capable of performing and translating them into rules for a rule-based engine or planner. There are no specific means for modelling rules in MESSAGE other than the concept of task with its pre- and post-conditions.

Most of these features are supported by the agent toolkits and were evaluated in the case studies. The assumption is made that these toolkits provide the basic agent infrastructure, such as matchmaking mechanisms for finding services, ontology support, or communication primitives.

The methodology provided sufficient separation of concerns between multi-agent considerations such as coordination and communication and the internal agent design. The MESSAGE models and associated diagrams (Organization, Goal, Agent/Role, Interaction, Workflow, and Domain) mostly support modelling the multi-agent considerations. Modeling the internal structure and behaviour of agents was mostly supported by agent pattern reuse, together with existing UML diagrams such as statecharts, activity, and class diagrams.

**Weaknesses and/or Purposeful Omissions**

The MESSAGE design modelling language is less mature than the analysis language, and the integration of the MESSAGE analysis and design processes into RUP is not fully finalized. Furthermore, some of the more difficult issues related to the dynamic behaviour of multi-agent systems were not explicitly addressed.

The question of dynamic behaviour of multi-agent systems could be relevant for many of the analysis and design modelling concepts (meta-classes and meta-associations). Some of them were identified but not studied in detail. The implicit assumption in the MESSAGE methodology is that all of the meta-model concepts are static; that is, new classes (instances of MESSAGE meta-classes) and associations (instances of MESSAGE meta-associations) cannot be added at run-time, only instances of them. This does not mean that an agent system designed with MESSAGE cannot display some degree of dynamic behaviour and adapt to specific kind of changes in the environment as described above, for example, by using rules to replan behaviour.

A methodology should provide explicit modelling support for the dynamic aspects of multi-agent systems. Consider the following examples of dynamic behaviour that are difficult to model with MESSAGE:
• An agent changes roles dynamically. In this case, an agent would be able to stop playing a role and could also start playing new roles dynamically during the lifetime of the multi-agent system.
• An agent interacting with another could be asked to play their roles (e.g., buyer and seller) in a more efficient manner by using a new interaction protocol for the same interaction.
• As the multi-agent system evolves, the ontology might have to be extended dynamically with new concepts. Agents would need to learn how to use the new ontology concepts.
• New agent classes could be added during the lifetime of the MAS providing new services that could be accessed using new interaction protocols. Existing agents would need to learn how to use these new services.

Modeling these kinds of changes was considered in the methodology but not addressed. One of the main reasons they were not addressed is that they were difficult to assess experimentally in the case studies. Most of the agent toolkits do not provide any explicit support for this kind of dynamic behaviour. Most of the above mentioned changes would require design time changes, recompilation, and redeployment of an agent-based application with most current agent-oriented toolkits. For example, in the JADE toolkit, there is no support for changing the interaction protocols, providing new agent classes, or modifying the ontology during the execution of the system. These are design-time choices that are compiled into the application and cannot be changed without modifying the design, recompiling, and redeploying.

Another issue that was not sufficiently studied was the availability of an analysis and design pattern library associated with the methodology. This aspect has been partially covered in Garijo, Bravo et al. (2003), although the number and the type of patterns should be extended. For example, specific coordination and negotiations patterns between agents could be used to allow for continuous optimization.

Although agent-centred concepts and notations have proved to be suitable for analysis and design, developers also need guidelines and support for other stages of the life cycle such as requirements, implementation, testing, and deployment.

Conclusions

Analysis focuses on describing the organizations involved in the multi-agent system, agent goals, and the roles defined to satisfy them. The high-level goals
were decomposed and satisfied in terms of services provided by roles. The interactions between roles that are needed to satisfy the goals were also described.

Agents were identified during design based on the description of the organizations and were assigned all the organization’s roles in the multi-agent application. Services could then be implemented in terms of tasks that were decomposed into direct actions on the agent’s internal representation of the environment, plus communicative actions to send and receive messages in inter-action protocols. The interactions between roles identified in analysis were detailed in terms of interaction protocols. The design description is an implementation-independent conceptual description of the system.

Organization-driven detailed design helps developers to define the computational behaviour of the overall system by stepwise refinement. The refinement process is agent centred. The organization model facilitates component-based development and reusability. It also helps to maintain consistency among analysis entities, design entities, and system requirements.

Viewing agents as reusable software components provides several advantages: (1) applications are developed by selecting and assembling the appropriate components; (2) integration and inter-operability among agents, standard component-ware technology, and supporting tools is assured; and (3) developers unfamiliar with agent concepts and notations may choose agent-based components to fulfill specific functionality of their applications. This permits agent technology to be easily assimilated into current engineering practice.

MESSAGE, as it stands, is not a complete, mature agent-oriented methodology. It does, however, make some significant practical contributions to the state of the art (Caire, Coulier et al., 2001, 2002) that are likely to influence ongoing initiatives in this area, for example, Agent UML (Odell, Parunak, & Bauer, 2000) or the FIPA modelling and methodology standardization activities. In particular, the graphical notation/diagram set, which extends UML class and activity diagrams, is a practical and concrete result that could be taken up widely.

**Acknowledgments**

This work was supported by EURESCOM (http://www.eurescom.de) Project P907. The editing of parts of this chapter was supported by the Spanish Ministry for Science and Technology (TIC2002-04516-C03-03) and FEDER, and the Walloon region under contract Convention EP1A1203000012 N DGTR 130023.


Cook, S. (2005). UML2.0 – Trying to have it both ways in UML – the Good, the Bad or the Ugly? – Perspectives from a panel of experts. Software and Systems Modelling, 4(1), 4.


Chapter IX

The INGENIAS Methodology and Tools

Juan Pavón
Universidad Complutense de Madrid, Spain

Jorge J. Gómez-Sanz
Universidad Complutense de Madrid, Spain

Rubén Fuentes
Universidad Complutense de Madrid, Spain

Abstract

INGENIAS provides a notation for modeling multi-agent systems (MAS) and a well-defined collection of activities to guide the development process of an MAS in the tasks of analysis, design, verification, and code generation, supported by an integrated set of tools—the INGENIAS Development Kit (IDK). These tools, as well as the INGENIAS notation, are based on five meta-models that define the different views and concepts from which a multi-agent system can be described. Using meta-models has the advantage of flexibility for evolving the methodology and adopting changes to the notation. In fact, one of the purposes in the conception of this methodology is to integrate progressive advances in agent technology, towards a standard for agent-based systems modeling that could facilitate the adoption of the agent approach by the software industry. The chapter presents a summary of the INGENIAS notation, development process, and support tools. The use of INGENIAS is demonstrated in an e-business case study. This case study includes concerns about the development process, modeling with agent concepts, and implementation with automated code generation facilities.