Chapter III

The Agent-Oriented Methodology MAS-CommonKADS

Carlos A. Iglesias
Technical University of Madrid, Spain

Mercedes Garijo
Technical University of Madrid, Spain

Abstract

This chapter introduces the main concepts of the methodology MAS-CommonKADS that extends object-oriented and knowledge engineering techniques for the conceptualisation of multi-agent systems. MAS-CommonKADS defines a set of models (Agent Model, Task Model, Expertise Model, Coordination Model, Communication Model, Organisation Model, and Design Model) that together provide a model of the problem to be solved. Each of the components of the model is a generic component for the sake of reusability. Readers familiar with object-oriented analysis will find it easy to apply most of the techniques of MAS-CommonKADS in the development of multi-agent systems and will be introduced to the application of knowledge engineering techniques for specifying the knowledge of the agents.

Introduction

MAS-CommonKADS is an agent-oriented software engineering methodology that guides the process of analysing and designing multi-agent systems. MAS-CommonKADS distinguishes several development phases: *conceptualisation*, where the system is conceived as a multi-agent system and where agent properties of the system are identified; *analysis*, where different models are developed in order to analyse the system from different points of view; *design*, where the different models are operationally focussed; and *development* and *testing*, which are not addressed explicitly in the methodology.

MAS-CommonKADS (Iglesias, 1998; Iglesias, Garijo, González, & Velasco, 1998) can be used in combination with other methodologies. For example, some of its conceptualisation techniques, such as *Class-Responsibility-Collaboration* (CRC) cards (Beck & Cunningham, 1989; Wirfs-Brock, Wilkerson, & Wiener; 1990) and *User-Environment-Responsibility* (UER) techniques (Iglesias & Garijo, 1999) can be used for conceiving a system from an agent point-of-view and be combined with other methodologies such as *Rational Unified Process* (RUP) (Kruchten, 2000) or *eXtreme Programming* (XP) (Beck, 1999); or use another agent-oriented methodology. In the same way, every analysis model can be used in combination with another methodology.

MAS-CommonKADS has as one of its goals to be usable by professionals who want to include in their projects this new and exciting computation paradigm—agents. In this way, MAS-CommonKADS extends well-known modelling techniques, such as CRC cards, use cases, *Message Sequence Charts* (MSC) (ITU-Z.120, 1996) or *Specification and Description Language* (SDL) (ITU-T-Z.100, 1994) diagrams, with new perspectives driven by the agent metaphor.

This makes many of MAS-CommonKADS techniques easy to learn and practice. The recent addition of MSC and SDL diagrams to *Unified Modelling Language* (UML) (Salic, 2004) makes MAS-CommonKADS even easier to practice with standard object-oriented CASE tools that can be enhanced with stereotypes for some of the new modelling entities (for example, software actor or environment).

The MAS-CommonKADS Methodology

The origins of MAS-CommonKADS come from CommonKADS (Schreiber et al., 1999), a well-known knowledge engineering methodology, and from object-oriented methodologies such as *Object Modelling Technique* (OMT)

(Rumbaugh, Blaha, Premerlani, & Eddy, 1991), *Object-oriented Software Engineering* (OOSE) (Jacobson, Christerson, Jonsson, & Övergaard, 1992) and *Responsibility Driven Design* (RRD) (Wirfs-Brock, Wilkerson, & Wiener; 1990). In addition, it includes techniques from protocol engineering such as SDL and MSC. All these techniques are combined in order to provide support to agent developers.

MAS-CommonKADS is based on the models of CommonKADS extended and adapted to agent modelling, including the definition of a new model, the coordination model, for describing agent interactions.

The software development life cycle in MAS-CommonKADS follows the phases described below:

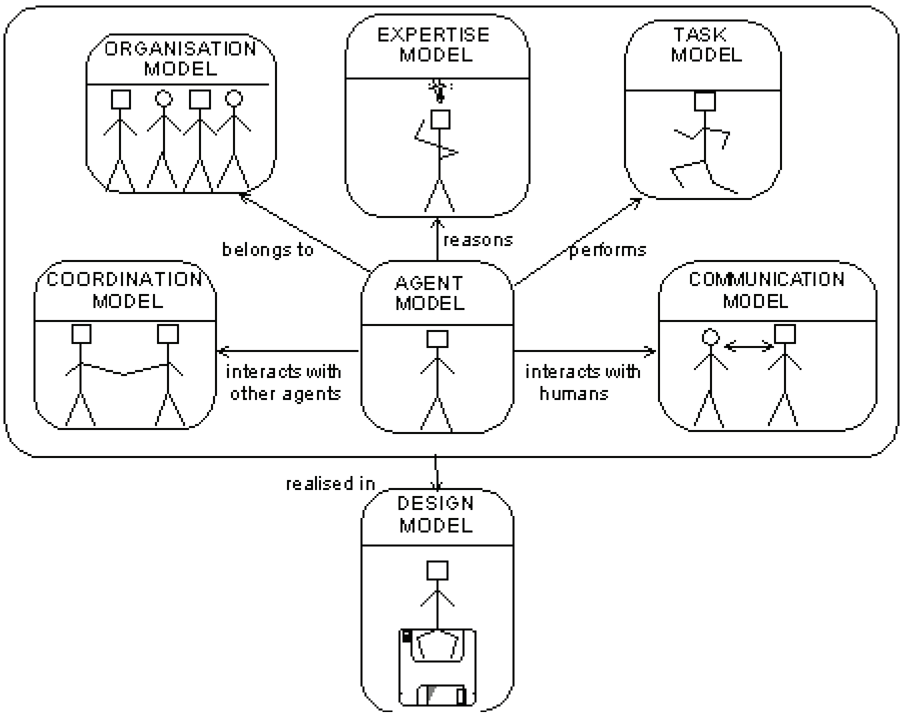
- **Conceptualisation**. Elicitation task in order to obtain a first description of the problem through the definition of a set of use cases that help to understand the system and how to test it.
- **Analysis**. The analysis phase determines the functional requirements of the system. It describes the system through the development of a set of models.
- **Design**. The design phase combines a top-down and bottom-up approach, reusing developed components and developing new ones, depending on the targeted agent platform. The design phase takes as an input the analysis models, which are then operationalised, that is, transformed into specifications (the design model) ready to be implemented. The internal architecture of every agent and the "network architecture" of the system are determined.
- **Development** and **testing**. Coding and testing tasks of the previously defined agents.
- **Operation**. Maintenance and operation of the system.

The methodology defines the following models (Figure 1):

- **Agent model** that specifies the agent characteristics: reasoning capabilities, skills (sensors/effectors), services, agent groups, and hierarchies (both modelled in the organisation model).
- **Task model** that describes the tasks that the agents can carry out: goals, decompositions, ingredients, problem-solving methods, and so forth.
- **Expertise model** that describes the knowledge needed by the agents to achieve their goals.

- **Organisation model** that describes the organisation into which the MAS is going to be introduced and the social organisation of the agent society.
- **Coordination model** that describes the conversations between agents, their interactions, protocols, and required capabilities.
- **Communication model** that details the human-software agent interactions and the human factors for developing these user interfaces. This model uses standard techniques for developing user interfaces and, consequently, it is will not discussed in this chapter.
- **Design model** that collects the previous models and consists of three submodels: network design, for designing the relevant aspects of the agent network infrastructure (required network, knowledge and telematic facilities); agent design, for dividing or composing the agents of the analysis, according to pragmatic criteria and selecting the most suitable agent architecture for each agent; and platform design, for selecting the agent development platform for each agent architecture.

Figure 1. Models of MAS-CommonKADS



Conceptualisation

The problem of conceptualisation is the first step towards the identification of the functional requirements of a system. One of the most extended techniques for getting a first idea of the system is the Use Case technique. The technique consists in identifying the possible users of the systems and the possible user goals, describing ways of achieving these user goals. These textual descriptions are the use cases. Usually, different use cases can be combined with the relationships extend (if a use case is an extension of another one) or include (if a use case is a part of another one). This technique is very simple and intuitive and has been very successful for requirements elicitation and validation.

The use case technique can also be used for conceptualising a multi-agent system. Nevertheless, autonomous agents are distinguished because they do not need a user that supervises their execution. So, while with use cases we have to answer the question, How is my system used?, we could ask ourselves about other requirements of our system such as: When and how does my system act and react to the environment? (environment cases) and What are the goals of the system? (responsibility or goal cases).

In order to conceptualise an agent-based system, two general techniques are used in MAS-CommonKADS: the UER cases technique that deals with the identification of use, reaction, and goal cases of an agent or a multi-agent system, and the enhanced Class-Collaboration-Responsibility Cards technique that deals with the identification of responsibilities, plans, and collaborations of an agent. Both techniques are complementary. The UER technique can be used for both single-agent or multi-agent systems (for identifying use, reactive, and goal cases of the whole system). The enhanced CRC cards can only be used for conceptualising multi-agent systems, since they guide the definition of collaborative scenarios.

UER Technique

The *User-Environment-Responsibility* (UER) technique (Iglesias & Garijo, 1999) combines user, environment and responsibility-driven analysis for conceptualising a system from an agent-oriented perspective. This technique can be used for conceptualising a particular autonomous agent or the general requirements of a multi-agent system. The technique analyses the system from three complementary perspectives: the user perspective, the environment perspective, and the assigned responsibility perspective.

- **User-Centred Analysis.** The potential users (called actors) of the system are identified, together with their possible tasks or functions. The result of this analysis is the set of use cases. This analysis answers the question: What are the possible uses of the multi-agent system?
- **Environment-Centred Analysis.** Agents can be situated in an environment, and this environment needs to be modelled. In particular, we are interested in modelling how the system can act and react to this environment. The result of this analysis is the set of reaction cases. This analysis answers the question: How has the multi-agent system reacted to the environment?
- **Responsibility-Driven Analysis.** In contrast to usual software systems, multi-agent systems can act proactively. The user can desire that the system has some responsibilities, that is, the user can assign some goals or responsibilities to the system and the system carries out these responsibilities without a direct demand. This analysis answers the question: What are the goals of the system? The main difference of goal cases from the use cases is that the use cases show how the system gives an answer to a user request, while the goal cases show how the system behaves when some condition is fulfilled.

The application of the UER technique introduces some of the most relevant properties of an agent system, such as reactivity and proactiveness in the conceptualisation of the system.

User-Centred Analysis

A use case describes the possible interactions or uses of a system by a user. System users are called actors and represent external entities of the system. Use cases can be combined, pointing out if a use case extends or includes a previous use case.

User-Centred Analysis consists of the following steps:

- **Identify the Actors.** It is especially relevant to identify the roles played by the actors. Each role is considered a different actor. There are two general kinds of actors: human actors (round head) and software actors (square head).
- **Identify the Use Cases.** This process can be carried out by answering the following questions:

  - What are the main tasks or functions carried out by each actor?
  - What system information is acquired, produced or changed by each actor?
  - Does any actor inform about external changes in the system environment?
  - What information is needed by each system actor?
  - Does any actor need to be informed about unexpected changes?
- Group the use cases if they are variations of the same subject (for example, 'move a heavy stone' and 'move a light stone').
- Determine the interactions of each identified use case.
- Describe the use cases using both a graphical notation and textual templates.
- Consider every possible exception that can happen during the interactions and how each affects the use cases.
- Look for relationships among the use cases: extract common parts and note if a use case adds the interactions of another use case (relationship "include") or adds information contained in another use case (relationship "extends" or "include").
- Describe the interactions of each scenario, using *Message Sequence Chart* (MSC) notation. MSC has been selected because is a standardised, formal description technique with a textual and graphical grammar. Some of the relevant features for our purposes are the availability of a language (*High-level MSC,* HMSC) for defining the phases of the interaction, and the definition of operators for expressing alternatives, exceptions, and concurrence in the same diagram. Although sequence and collaboration diagrams do not allow the expression of these issues in such an easy way, they can also be used.

Environment-Centred Analysis

The goal of environment-centred analysis is to identify the relevant objects of the environment and the possible actions and reactions of the agent. This will be used later for agent sensor modelling.

Environment-Centred Analysis consists of the following steps:

- Identify objects of the environment. These objects are shown in the use case diagram as clouds.

- Identify the possible events coming from each object and establish a hierarchy if possible.
- Identify the possible actions each agent can carry out on the environment objects.
- Describe (in natural language) the reaction cases coming from interaction with the environment. Describe in detail each possible scenario. Assess if there are several scenarios arising from the same reaction case and whether every scenario is autonomous (it is only managed by the agent that receives the stimuli) or cooperative (it is managed in cooperation with other agents).
- Group-related reactive cases with the relationships "extends" or "includes." For example, "avoid obstacle" can group different scenarios for avoiding an obstacle depending on its nature and can be avoided in an autonomous way (e.g., just going to the left of the obstacle) or in a cooperative way (e.g., asking for help to move it).
- Describe the reactive goal: its name, the activation condition (e.g., a wall very close), the deactivation condition, and the success and failure condition (whether the reaction has been effective or not).

Responsibility-Driven Analysis

Responsibility- or goal-driven analysis deals with the definition of the requirements of the system that should be fulfilled without the direct interaction with the user.

Goal-Driven Analysis consists of the following steps:

- Identify responsibilities (goals) of the system that require some action. Some hints for identifying these goals are:
  - o Look for non-functional requirements, such as time requirements (e.g., 'Give an answer within five minutes') or security requirements (e.g., 'Buy a product in a secure way'). Sometimes the agent needs to carry out special actions to achieve these goals.
  - o Describe when some internal variable of the agent can reach an undesirable value and what action should be carried out (e.g., high/low temperature or too many processes).
  - o Describe undesired states, possible failures of the system that should require action to be avoided.

- Describe the proactive goal: its name, its type (persistent, priority, etc.), the activation condition (e.g., no fuel or idle), the deactivation condition and the success and failure condition (whether the plan has been effective or not).
- Group related goals using the relationships “extends” or “includes.”

Enhanced CRC Cards and Internal Use Cases

The well-known *Class Responsibility Collaboration* (CRC) cards technique (Beck & Cunningham, 1989; Wirfs-Brock, Wilkerson & Wiener; 1990) provides a method for organising the relevant classes for modelling a system. This technique was initially used for teaching object fundamentals in a collaborative environment. The technique consists of filling in cards. Each card has a class name and two columns. The left column shows the responsibilities of the class, namely, the tasks the class can perform or knowledge it has, and the right column show the classes that collaborate to achieve these tasks or obtain this knowledge.

This technique can be easily modified from an agent perspective. A CRC is filled for each agent class. Each CRC is divided into five columns (Figure 2): goals assigned, plans for achieving these goals, knowledge needed to carry out the plans, collaborators in these plans, and services used in the collaboration. The back side of the CRC is used for annotations or extended description of the front side.

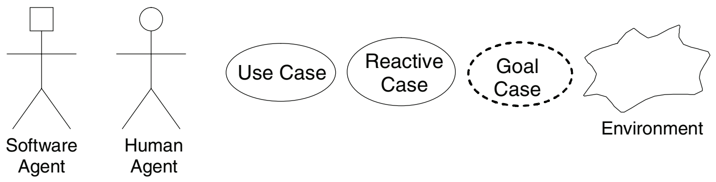
During this stage the knowledge is collected in an informal way. Later, in the analysis phase, it will be specified which role the agent plays in the task, knowledge, coordination, and communication models.

Internal use cases are also based on RDD and its CRC cards. Taking as input the use cases of the conceptualisation phase and some initial agents, we can think

Figure 2. Enhanced CRC cards

Agent:				
Goals	Plans	Knowledge	Collaborator	Service

Figure 3. User-environment-responsibility notation



that each agent "uses" other agent(s) and can use these agents with different roles. In Figure 3, the use case notation is extended for showing human agents (with the round head) and software agents (with the square head). We try to reuse such an agent from our agent library, combining in this way the top-down and bottom-up approach. The external use cases coming from the actors of the multi-agent system are decomposed in use cases that are assigned to agent roles of the system.

Analysis

The results of this phase will be the requirements specification of the MAS through the development of the models previously described, except for the design model. These models are developed in a risk-driven way, and the steps are:

- **Agent Modelling:** developing initial instances of the agent model for identifying and describing the agents.
- **Task Modelling:** task decomposition and determination of the goals and ingredients of the tasks.
- **Coordination Modelling:** developing the coordination model for describing the interactions and the coordination protocols between the agents.
- **Knowledge Modelling:** modelling of the knowledge about the domain, the agents (knowledge needed to carry out the tasks and their proactive behaviour), and the environment (beliefs and inferences of the world, including the rest of the agents).
- **Organisation Modelling:** developing the organisation model. Depending on the type of project, it may be necessary to model the organisation of the enterprise in which the MAS is going to be introduced for studying the feasibility of the proposed solution. In this case, two instances of the organisation model are developed—before and after the introduction of the MAS. This model is also used to model the software agent organisation.

The Agent Model

The agent model acts as a link between the rest of the models of MAS-CommonKADS, since it collects the capabilities and restrictions of the agents.

MAS-CommonKADS proposes different strategies that can be combined in order to identify the agents of our problem. Some of these techniques are:

- Analysis of the actors of the use cases defined in the conceptualisation phase. The actors of the use cases delimit the external agents of the system. Several similar roles (actors) can be mapped onto one agent to simplify the communication.
- Analysis of the statement of the problem. The syntactic analysis of the problem statement can help to identify some agents. The candidate agents are the subjects of the sentences, the active objects. The actions carried out by these subjects should be developed by the agents as goals (with initiative) or services (under demand).
- Usage of heuristics. The agents can be identified by determining whether there is some conceptual distance: knowledge distribution, geographical distribution, logical distribution, or organisational distribution.
- Initial task and expertise models can help us to identify the necessary functions and the required knowledge capabilities, resulting in a preliminary definition of the agents. The goals of the tasks will be assigned to the agents.
- Application of the internal use cases technique—see above.
- Application of the enhanced CRC cards (Figure 2)—see above

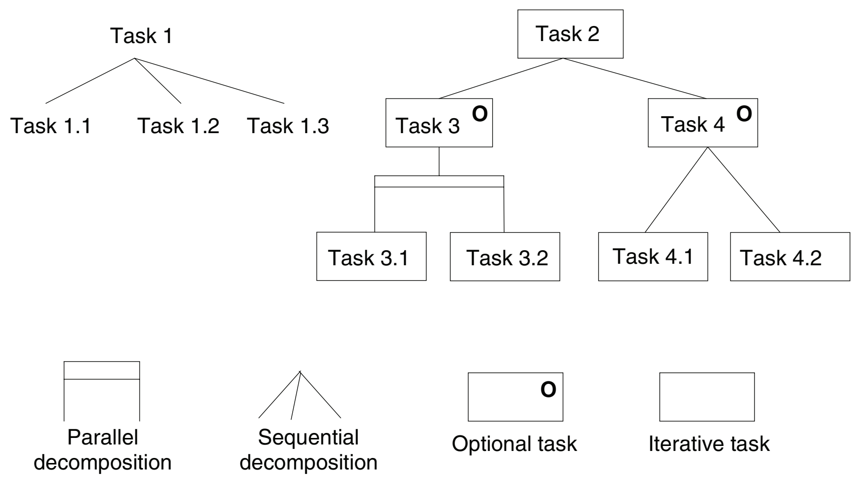
Once the agents have been identified, every agent should be further described using textual templates that collect the main characteristics of the agents, such as its name, type, role, position, a description, offered services, goals, skills (sensors and effectors), reasoning capabilities, general capabilities, norms, preferences, and permissions. The process of filling in these templates helps the engineer to review his/her understanding of the problem and serves as a means of communication with the rest of the team.

The Task Model

The task model describes all the activities (so-called tasks) that should be performed in order to achieve a goal.

Tasks are decomposed following a top-down approach and described in an "and/or" tree. The description of a task includes its name, its goal, a short description, input and output ingredients, task structure, its control, frequency of application, preconditions, and required capabilities of the performers.

Figure 4. Task diagram notation



The potential benefits of the development of this model are the documentation of the activities of the organisation before and after the introduction of the multi-agent system. This documentation serves to support the maintenance and management of changes in the organisation and to support project feasibility assessment.

The graphical notation of this model (Figure 4) follows traditional tree decomposition or, alternatively, a decomposition where optional and iterative tasks are indicated. It can be also be used to describe whether the tasks can be performed in a parallel or sequential way. Usually, the first versions of the model use just the sequential decomposition and refined versions of the model introduce gradually parallel tasks, optional tasks, or iterative tasks. Alternatively, the activity diagram of UML can be used for this model.

In case a task is knowledge intensive, it should be further developed in the expertise model. In the same way, if a task requires the agent interaction or human interaction, it should be further developed in the coordination model or communication model, respectively.

The Coordination Model

The coordination model specifies the interactions between the agents of the multi-agent system. The main components of the coordination model are the conversations between agents that are initiated to fulfill a goal in a cooperative way. Every conversation is composed of interactions (associated to speech acts) and follows a conversation protocol. In order to establish a conversation, there are some capabilities between the agents that maintain this conversation (capabilities and knowledge) that are specified in this model.

The coordination model has two milestones: (1) definition of the communication channels and building of a software prototype for testing purposes (as a mock-up); and (2) analysis of the interactions and determination of complex interactions (with coordination protocols).

The first phase consists of the following steps:

1. Describe the prototypical scenarios between agents using MSC notation. The conversations are identified, taking as input the results of the techniques used for identifying agents. During this first stage, we will consider that every conversation consists of just one single interaction and the possible answer.
2. Represent the events (interchanged messages) between agents in event flow diagrams (also called service charts). These diagrams collect the relationships between the agents via services.
3. Model the data interchanged in each interaction. The expertise model can help us to define the interchanged knowledge structures. These interchanged data are shown in the event flow diagram between square brackets.
4. Model each interaction with the state transition diagrams of *Specification and Description Language* (SDL), specifying speech-acts as inputs/outputs of message events. These diagrams can be validated with the MSC diagrams.
5. Each state can be further refined in the task or expertise model.
6. Analyse each interaction and determine its synchronisation type: synchronous, asynchronous, or future.

The second phase consists of analysing the interactions for getting more flexibility (relaxing, for example, the user requirements), taking advantage of the parallelism, duplicating tasks using different methods, or resolving detected conflicts. When a cooperation protocol is needed, we should consult the library of cooperation protocols and reuse a protocol definition. If there is no protocol suitable for our needs, it is necessary to define a new one. We can use *High-level Message Sequence Charts* (HMSC), which are very useful for this purpose. These diagrams show the road map (phases) of the protocol and how the different phases (specified with MSC) are combined. A phase can be a simple MSC or another HMSC. The processing of the interactions is described using SDL state diagrams. It is also necessary to fill in the textual protocol template specifying the required reasoning capabilities of the participants in the protocol. These capabilities can be described using one or several instances of the expertise model. The state diagrams consider three kinds of events: *message events*, which are events from other agents using message-passing; *external events*, events from the environment perceived through the sensors; and *internal events*, events that arise in an agent because of its proactive attitude. The potential benefits of the development of this model are:

- The development of the coordination model is a means for specifying the prototypical interactions between the agents working on the resolution of a problem, together with the interactions with the environment. This model is used to store the decisions of the structure of communications and the protocols associated with these communications. The usage of these descriptions is twofold: the designer can reuse protocols and scenarios, and the intelligent agent can select them at run time.
- MSC and SDL are formal description techniques with a well-defined syntax and semantic. The usage of these languages for specifying interactions in multi-agent systems have been achieved by: (1) defining one signal type for each possible speech-act (message type); (2) associating a logical expression to each state name (using commentaries); and (3) considering internal events (similar to spontaneous transitions) for changes in the mental state of the agent motivated because of its proactive attitude.

The development of this model can help in the maintenance and testing of a multi-agent system.

The Expertise Model

The expertise model, which is the focus of CommonKADS, is used for modelling the reasoning capabilities of the agents to carry out their tasks and achieve their goals. Normally, several instances of the expertise model should be developed: modelling inferences on the domain (i.e., how to predict delays in flights, taxonomies of delays and flights, etc.); modelling the reasoning of the agent (i.e., problem-solving methods to achieve a task, character of the agent, etc.); and modelling the inferences of the environment (how an agent can interpret the event it receives from other agents or from the world). When we have to develop the reasoning capabilities of an agent, we will reuse previously developed instances of the expertise model and adapt these instances to the new characteristics of the problem.

The expertise model consists of the development of the application knowledge (consisting of domain knowledge, inference knowledge, and task knowledge) and problem-solving knowledge.

The usage of this model can take advantage of the work previously developed, for example, for developing a planner.

- *Domain Knowledge* represents the declarative knowledge of the problem, modelled as concepts, properties, expressions, and relationships using the Conceptual Modelling Language (CML) or the graphical notation of the Class Diagrams of UML.
- *Inference Knowledge* represents the inference steps performed for solving a task. There is a library of generic inference structures selected by the task type (diagnosis, assessment, etc.). These generic inference structures should be adapted to the problem. The inference structure is a compound of predefined inference types (how the domain concepts can be used to make inferences, represented as ellipses) and domain roles (rectangles). This generic inference structure should be adapted to our problem. After defining the inference structure, it is instantiated into a similar diagram for the domain.
- *Task Knowledge* represents the order of the inference structures. The notation consists of inference structures and task-method inference decomposition structures.
- *Problem-Solving Method*: during the design we should specify a *Problem-Solving Method* (PSM) for each inference type: how the inference is carried out. The PSMs are arranged in libraries for reuse. Two generic kinds of PSMs are defined: *autonomous PSMs*, when the plan is carried out by the agent itself, and *cooperative PSMs*, when the PSM takes into account the participation of other agents. In addition to defining the PSMs, there should be defined its assumptions (when a PSM can be selected and when it is more suitable).

The potential benefits of the development of this model are the utilisation of a well-known knowledge-level modelling framework, which has been successfully applied in several projects, and the provision of a library of generic components, specification languages and software tools.

The Organisational Model

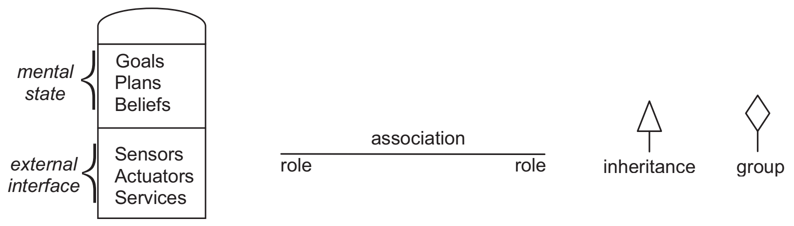
CommonKADS defines the organisation model for modelling the organisation in which the knowledge-based system is going to be introduced. Here the model is extended in the same way as the agent model for modelling the organisation of agents. This model shows the static or structural relationships between the agents, while the coordination model shows the dynamic relationships. The graphical notation of these models is based on the notation of the Class Diagrams of UML, adding a stereotype for distinguishing between agents and objects (Figure 5). The aggregation symbol is used for expressing agent groups.

The agent symbol is that of MOSES (Henderson-Sellers & Edwards, 1994). In contrast to this and to UML, the upper box does not store the defined attributes as in UML, but rather the mental state and internal attributes of an agent, such as their goals, beliefs, plans, and so forth. The lower box stores the external attributes of the agents: services, sensors, and effectors.

In addition, the relationships between agents are modelled with associations in which the roles played by the agents are described. Two special associations are considered: inheritance and group.

The organisation model is used for modelling both the human organisation where the multi-agent system is going to be developed and the multi-agent society itself. The main modelling steps are the description of agent (human and software) relationships, detailing the roles played in every relationship, and the study of the relationship of the environmental objects with the agents. In the case of software agent relationships, the model will collect the different use cases developed in the coordination model, while in the human-software agent case, the system will collect the use cases developed in the communication model. As a result of this first analysis, the organisation model will define the static and dynamic relationship between both human and software agents and the roles played by them in the different interactions (in addition to the required knowledge to be able to perform those interactions). During this process, inheritance and group relationships between software agents can be modelled as a result of the analysis.

Figure 5. Organisation model notation



The inheritance relationship between agents is defined as the union of the values of the precedent classes for each attribute. For example, an agent class has its goals and the goals of the precedent agent classes. If an agent defines an attribute as exclusive, the values are overwritten. The potential benefits of the development of this model are the specification of the structural relationships between human and/or software agents, and the relationship with the environment. The study of the organisation is a tool for the identification of possible impacts of the multi-agent system when installed. In the same way, this model can provide information about the functions, workflow, process, and structure of the organisation that allows the study of the feasibility of the proposed solutions. This model represents both class-agent diagrams and instance-agent diagrams, showing the particular relationships with the environment. In contrast with other paradigms (i.e., object-oriented), the agent-instance diagrams are frequently more relevant than the class-agent diagrams.

The Design Model

As a result of the analysis phase, an initial set of agents has been determined. During the design phase, the design model is developed. This phase is extended for multi-agent systems and consists of the following phases:

- **Agent network design:** the infrastructure of the multi-agent system (so-called network model) is determined and consists of network, knowledge, and coordination facilities. The agents (so-called network agents) that maintain this infrastructure are also defined, depending on the required facilities. Some of these required facilities can be:
  - **Network facilities:** agent name service, yellow/white pages service, de/registering and subscription service, security level, encryption and authentication, transport/application protocol, accounting service, and the like.
  - **Knowledge facilities:** ontology servers, PSM servers, knowledge representation language translators, and so forth.
  - **Coordination facilities:** available coordination protocols and primitives, protocol servers, group management facilities, facilities for assistance in coordination of shared goals, police agents for detecting misbehaviours and the control of the usage of common resources, and so forth. The result of the common facilities shared by the agents

allows the efficient communication between the agents and is expressed by an ontology, in the same way as a service ontology.

- **Agent design:** the most suitable architecture is determined for each agent, and some agents can be introduced or subdivided according to pragmatic criteria. Each agent is subdivided in modules for user-communication (from communication model), agent communication (from coordination model), deliberation and reaction (from expertise, agent, and organisation models), and external skills and services (from agent, expertise, and task models). The agent design maps the functions defined in these modules onto the selected agent architecture. The issue of designing an agent architecture is not addressed in the methodology, since the agent architecture is provided by the agent development platform.
- **Platform design:** selection of the software (multi-agent development environment) and hardware that is needed (or available) for the system.

The potential benefits of the development of this model are:

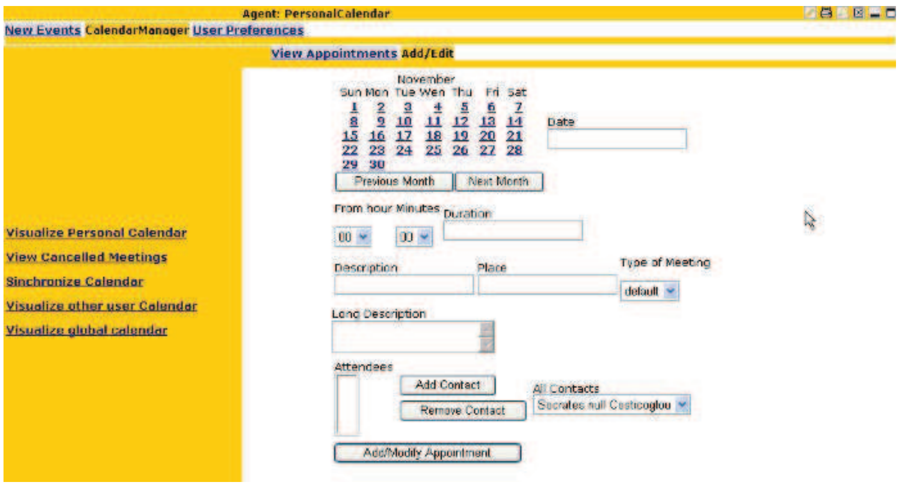
- The decisions on the selection of a multi-agent platform and an agent architecture for each agent are documented.
- The design model collects the information of the previously developed models and details how these requirements can be achieved.
- The design model for multi-agent systems determines the common resources and needs of the agents and designs a common infrastructure managed by network agents. This facilitates modularity in the design.

Example Case Study

This case study is based on the work carried out in the European IST project Collaborator 2000-30045 (Bergenti, Garijo, Poggi, Somacher, & Velasco, 2002), where a multi-agent system, called COLLABORATOR, has been developed and is capable of providing modern enterprises with a shared workspace supporting the activities of virtual teams. The description of the system in this chapter has been adapted in order to show the main characteristics of the methodology MAS-CommonKADS.

The COLLABORATOR system is designed to be used by users with different requirements, profiles, and devices. The users need to interact with each other

Figure 6. Screenshot of the Personal Calendar Agent



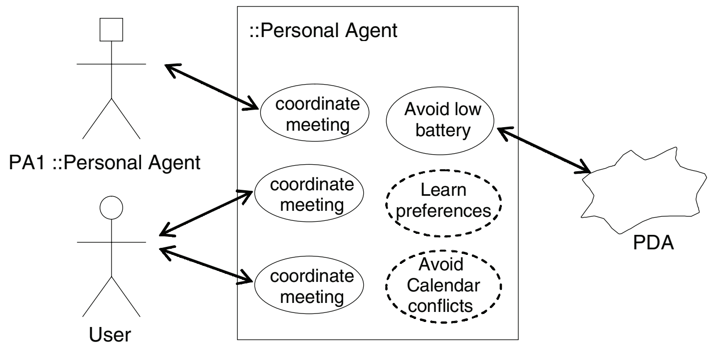
and with the system itself for preparing collaborative sessions, negotiating the dates of the sessions, and managing their personal agendas.

The system will be based on a simple Personal Calendar Agent that is in charge of managing the appointments of a user. The system will try to learn to set up a new appointment according to its user preferences. In the case of a conflict, the system will be able to negotiate with other personal agents from the other participants in the appointment. The system will be running on a PDA (Figure *6*).

Conceptualisation

The first phase of the methodology is the conceptualisation. In order to have a first understanding of the system, we can apply the UER technique resulting in the diagram shown in Figure 7. Two main actors have been identified: the user and the personal agent. The user will be able to confirm a meeting. The basic use case of the Personal Agent will be to negotiate a meeting, because its user wants to start to schedule a meeting or because another user wants to schedule a meeting with this user. This negotiation will be done through their personal agents. From the very beginning, the UER technique also takes into account some of the agent properties, such as reactivity and proactivity. In this case, one environment object is identified: the mobile phone. The agent could react to a low battery state. In addition, the agent has been assigned some responsibilities, such

Figure 7. UER cases for the Personal Agent



as avoiding calendar conflicts (two overlapping appointments) and learning user preferences, in order to be as autonomous and adaptable as possible.

For every use, goal, and reactive case, the case should be further described, describing its characteristics. For example, *Avoid low battery* is a high-priority goal that is activated when the battery charge is low, deactivated with success in case of reaching a high level of battery, and deactivated with failure in case the battery is below some very low level.

Once we have identified the UER cases, they should be described using textual templates as shown in Figure 8.

Figure 8. Textual template of the reactive case Avoid low battery

reactive-case Avoid low battery

description

The Personal Agent can detect that the PDA has a low level of battery. In this case he should try to store all the relevant data of the personal agenda and try to attract the attention of its user

activation-condition

low-battery level

deactivation-condition

recharge in process or plugged in

successful condition

personal gent hibernates before the PDA shutdowns or the use recharge the batteries

failure condition

out of battery

Figure 9. Reactive cases relationship diagram

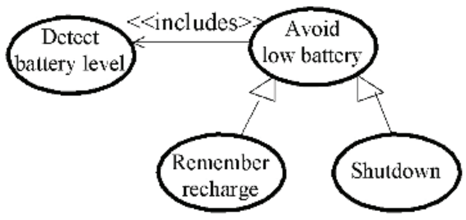


Figure 10. Extended CRC card

Agent PA		Class: Personal Agent		
Goals	**Plans**	**Knowledge**	**Collaborator**	**Service**
Avoid Calendar conflicts	Suggest User to choose	Calendar - Ontology	User	
	Try to group		Personal Agent	Change Appointment

The next step is grouping the cases and studying their relationships. In this example, we can identify two possible scenarios for avoiding low battery: periodically reminding the user to recharge the PDA (more frequently in the case of a low battery) or "shutdown" the PDA (Figure 9). We can also observe that there is a scenario included in this reactive case: the detection of the battery level.

Another technique we can use is that of extended CRC cards (Figure 10). Here, for example, we can select one goal of our agent and define the plans to achieve it, together with the required knowledge to perform the plans and the potential collaborations in these plans that could also be autonomous. In case the agent offers a standard service to other agents, we point out this service.

Analysis

After the conceptualisation phase, we have a first overview of the agents of the system. In this particular simplified system, we have previously identified one kind of agent: the Personal Agent. During the analysis phase, the engineer can develop the analysis models that are needed in its problem. Usually, agent and task model should always be developed. The development of the expertise model helps the development of intelligent agents. In the case of interaction between agents, the coordination model should be developed. The communication model is developed when there is human-agent interaction. The organisation model helps us to express structural relationships between agents or the human organisation in which the multi-agent system is going to be introduced.

The models are developed in an iterative way, developing as many instances of every model as needed and continuously refining the models. Since the models are composed of artefacts, these artefacts can be easily reused.

Agent Model

From the actors of the conceptualisation phase, we can identify one agent, the Personal Agent, which can have two roles: Coordinator or Participant. For the sake of geographic distance and adaptation to the user, every Personal Agent will be assigned to a single user.

In order to describe the Personal Agent, a textual agent template from the agent model is filled in as shown in Figure 11, which helps us think about the problem.

Figure 11. Initial textual agent template of the Personal Agent

Agent PersonalAgent
- **type**
  - Intelligent software agent
- **roles**
  - Coordinator / Participant
- **description**

  Personal Agent of a user, in charge of managing meetings in behalf of its user and inform its user of new information about the accepted meetings.
  Personal agent should classify incoming events in order to learn suitable scheduling depending on the event type and how the user handles every event type.
- **reasoning-skills**
  - **expertise**

    knowledge about the meeting domain and categories of events and other users ability to classify and capability to adapt to user preferences for scheduling
    (to be further defined in expertise model)
  - **restrictions**
    - **norms**

      A Personal Agent should notify its user of any change in an appointment.
    - **permissions**

      A Personal Agent only can be a coordinator in case its user has the rights to be coordinator

Task Model

Now we can proceed to the development of the task model. Every instance of the task model describes all the related activities to fulfill a goal. In this example, we will further elaborate one the previously identified use cases: coordinate a meeting. When a user wants to coordinate (initiate) a meeting, the user should be allowed to have the role of coordinator for a meeting in that project.

This task can be decomposed as shown in the task diagram (Figure 12). This task is initiated by the user. It should specify the foreseen length of the meeting, the project and the required expertise of the participants. The task *Collect-Coordinator-Requirements* models all the interaction between the user and the agent that can help the user to show the available options for his/her decision. Once the user has assigned the responsibility of coordinating the meeting, the personal agent should select a range of suitable dates (task 2) based on its user calendar and then select the attendees for that meeting (task 3). The next step is to negotiate with the personal agents of the participants a suitable meeting date as many times as needed (task 4, marked with an asterisk since it is an iterative task). Finally, the agent should confirm with its user the negotiated date. If it is suitable, the personal agent informs the other participants of the new appointment that it is then added to their respective calendars. If the user decides that the agreed upon date is not suitable, the user can start the task again or finish.

As seen in the previous task decomposition, the process of coordinating a meeting has been described in a general way and can be reused in other projects. Some of the tasks are further developed in other models. In particular, tasks *1 Collect Coordinator Requirements* and *5.1 Confirm User* are refined in the communication model since they involve human interaction. Task *4.2 Send CFP / Receive Answer* is refined in the coordination model since it involves agent interaction. The tasks *2. Select attendees* and *4.2. Decide Negotiate / End* are

Figure 12. Task Model for coordinating a meeting.

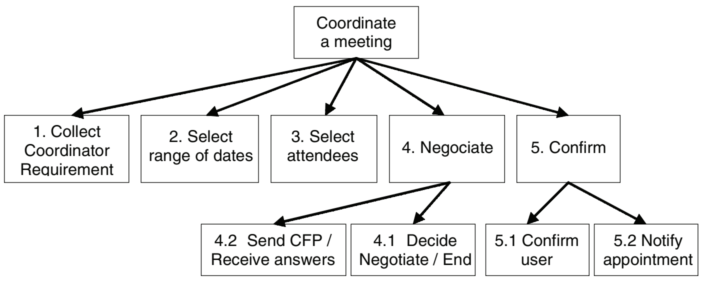


Figure 13. Task template of the task Select attendees

Task *3. Select attendees*
goal
Obtain a set of indispensable and potential participants
Description
The coordinator should select the participants for the meeting based on the project the meeting belongs to, the topic and the required expertise.
input
Characteristics of the meeting: project, required expertise (keywords), length, tentative dates (according to availability of the coordinator) and list of registered users
output
Qualified set of participants that are a subset of the registered users.
The participants are qualified as indispensable, convenient or unnecessary.
precondition
User should be authorised to act as a coordinator and there should be registered users
supertask
Decide a meeting
subtasks
Decomposition-type

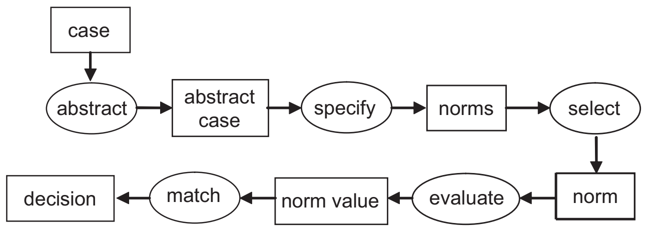
candidate tasks to be further developed in the expertise model since they may require some reasoning process.

In order to show how a particular task can be described in the task model, the task *3. Select attendees* is described in using a textual task template.

Expertise Model

Instead of detailing further the rest of the tasks of the task model in this iteration, we are going to illustrate how an instance of the expertise model can be developed in this case study. We can further develop the task *3. Select attendees* in order to see how an intelligent agent can make this decision. The expertise model (Schreiber et al., 1999) guides the knowledge-modelling process. CommonKADS has a library of generic tasks, independent of the domain, that can be reused or refined for knowledge intensive tasks. In this case, the selection of attendees can be considered as a task of Assessment. A task of assessment has the goal of finding a decision category for a case based on a set of domain-specific norms. For example, in a loan application, the assessment determines whether an application results in the loan or not (application-rejected, accepted-with-risk, or accepted). In our problem, we will have to decide whether a user would be interested and should participate in a meeting, qualifying him or her as indispensable, convenient, or unnecessary.

Figure 14. Inference diagram of PSM Assessment



The CommonKADS library drives the knowledge acquisition process. Once we have identified that this task is an *Assessment*, we can consult the CommonKADS library and get its inference structure and task method (Figure 14). The inference diagram illustrates inferences carried out in an assessment task. Ellipses represent basic inferences and boxes represent knowledge roles. Inference structures are domain independent, that is, they can be applied (reused) in different domains without modifications.

We need to map the generic knowledge role to domain classes. In our example, *case* represents data about the user and the meeting information, *abstracted case* represents an abstraction of the case. For example, we can compare the required expertise for attending the meeting and the users' expertise and determine a degree of expertise of every user for this particular meeting that can help later to classify him/her. In addition, *norms* represent the domain knowledge used in making the decision, for example, the criterion used to determine the qualification of a user, such as availability on the proposed dates. The knowledge role *norm value* represents a particular value of a norm (for example, the user is available on some of the proposed dates), while *decision categories* represents how the user is qualified (indispensable, convenient, or unnecessary).

According to this inference structure, the process of Assessment consists of obtaining an augmented case from the case we receive, that is, infer information, specifying a set of criteria (norms) that can be evaluated, and selecting one. This criterion is evaluated and, depending on this evaluation result, the user is classified (decision class).

In parallel with the selection and adaptation of the generic task, an initial domain can be modelled as shown in Figure 15. The degree-expertise is an abstraction that takes into account the keywords of the user (expertise defined by keywords) and the required expertise of the meeting.

In addition to this information, it is very important in this task to define the criteria (norms) that will be applied to decide the qualification of an attendee. Several criteria have been identified: the availability of the potential attendee on the dates

Figure 15. Domain model for selecting attendees

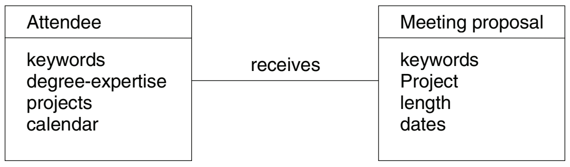
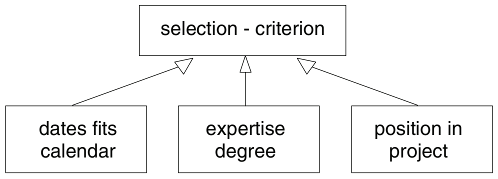


Figure 16. Hierarchy of criteria for selecting attendees



of the proposed meeting, the degree of expertise of an attendee for a meeting, and the relationship of the position of the potential attendee in the project as not member, engineer, project leader, and so forth (Figure 16).

The task method for Assessment (Figure 17) collects the control structure of the task as can be taken directly from the CommonKADS library.

Once we have defined the control and mapped domain classes onto the identified knowledge roles, we should specify the primitive inferences.

In our case study, the inference *abstract* obtains the attribute degree of expertise of the attendee by comparing the expertise of the attendee and the required expertise. This inference *abstract* can assign a degree between 0 and 1 that represents the number of required keywords that can be attributed to the attendee. The inference *specify* generates a list of applicable norms depending on the case. In this example, it just generates the list of the three identified criteria. The inference *select* selects a norm from the list, based on heuristics or randomly. The inference *evaluate* determines the degree of satisfaction (not satisfied, low, medium, high) of a particular criterion. For example, if a user is not a member of the project, the evaluation is low. The inference *match* determines the qualification based on the evaluation of all the criteria.

Once the inferences are specified, the knowledge bases can be filled. For example: *DateFitsCalendar is not-satisfied IMPLIES attendee is unnecessary*.

Figure 17. Method for assessment (Schreiber et al., 1999, p. 132)

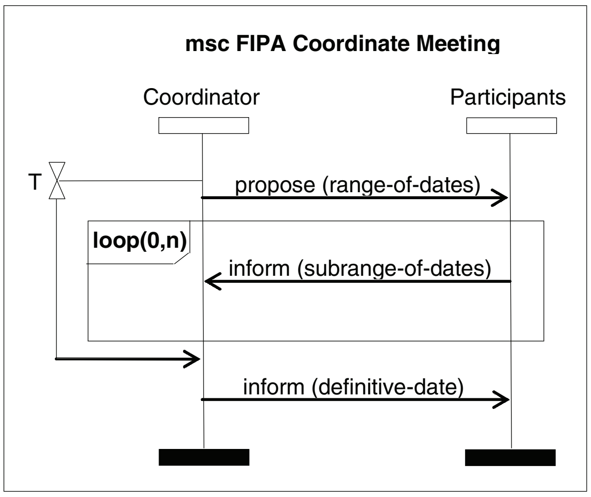
```
TASK assessment;
    ROLES:
        INPUT: case-description: "The case to be assessed";
        OUTPUT: decision: "the result of assessing the case";
END TASK assessment;
TASK-METHOD assessment-with-abstraction;
    REALIZES: assessment;
    DECOMPOSITION:
        INFERENCES: abstract, specify, select, evaluate, match;
    ROLES:
        INTERMEDIATE:
            abstracted-case: "The raw data plus the abstractions";
            norms: "The full set of assessment norms";
            norm: "A single assessment norm";
            norm-value: "Degree of satisfaction of the criterion (not satisfied,
            low, medium, high";
            evaluation-results: "List of evaluated norms";
    CONTROL-STRUCTURE:
        WHILE HAS-SOLUTION abstract(case-description -> abstracted-case)
        DO
            case-description := abstracted-case;
        END WHILE
        specify(abstracted-case -> norms);
        REPEAT
            select(norms -> norm);
            evaluate(abstracted-case + norm -> norm-value);
            evaluation-results := norm-value ADD evaluation-results;
        UNTIL
            HAS-SOLUTION match(evaluation-results -> decision);
        END REPEAT
END TASK-METHOD assessment-with-abstraction;
```

Coordination Model

Now we will show how to model interactions between agents using the coordination model. The coordination model describes the conversation between agents and identifies the services that agents offer to the rest of agents. The coordination model identifies the needed conversations, usually with the input from other models. In this case study, we have identified a task 4.1 *Send CFP / Receive Answer* and task *5.2 Notify appointment.*

Once a conversation is identified, the next step is to describe the roles of the participants in the conversation, the interactions and services, and the information interchanged in the interaction, and then analyse how every interaction can

Figure 18. Message Sequence Chart of negotiating a date



be modelled (for example, with or without negotiation, and how the agent should select one or another protocol). In the case study, there are two identified roles in this conversation (coordinator and participant). The coordinator sends to the participants a meeting description with a range of potential dates and receives in a certain period the answers with subranges of the initial range. Then, the coordinator decides a date and notifies all participants.

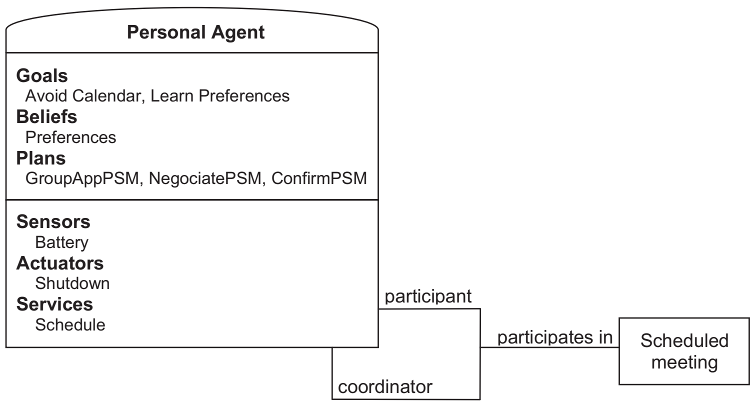
The notation of the model is based on MSC (ITU-Z.120, 1996) for the interactions and SDL (ITU-T-Z.100, 1994) for processing these interactions. Alternative notations are UML2.0 Interaction Diagrams, which incorporate most of the characteristics of MSC and SDL, and AUML Interaction diagrams (Beuer, Müller, & Odell, 2001).

In order to illustrate the application of other models, we will develop here the task of the task model. The MSC diagram is shown in Figure 18.

Organisation Model

The organisation model's agent model uses the organisation model diagram (Figure 19). In this example, there only two positions in the organisation, depending on the scheduled meeting: Coordinator or Participant. The organisational

Figure 19. Organisation diagram of collaborator



model could also be used for modelling the human positions in the company and resolve eventual conflicts according to its hierarchical position.

Design

The design model describes the components that satisfy the requirements of the analysis models. There are three main decisions to be carried out in this model: platform design, agent network design, and agent design that are collected in a textual template as shown in Figure 20.

In this case study, the users will be able to see the scheduled meetings in their PDAs and on the Web. The platform selected was JADE (*http://www.jade.cselt.it*) and the extension BlueJADE (Cowan & Griss, 2002) for deploying JADE as a service of the application server JBoss (http://www.jboss.org).

The *design of the agent network* consists of identifying a set of agents that provide the network, coordination, and knowledge facilities. In this case study, attending to design considerations, since the users can be off-line, there is a need to define a new agent, the *Session Manager Agent* (SMA) that is woken by the application server when needed (to initiate a new session). In this way, the task model developed as part of the task model of the Personal Agent is now reassigned to the Session Manager Agent. We can identify another service, the

Figure 20. Design textual templates for Session Manager Agent (SMA) objects

Platform *Collaborator*
Languages: Java, Jess
Software: JBoss, JADE, Jess

Network *Collaborator*
Network-services – FIPA Standard (AMS, DF)
Knowledge-services: none
Coordination services: Resource Manager Service, Session Manager Agent

Agent System *Session ManagerAgent*
Subsystems:
user-interaction *SMAInterface* (tasks 1, 5.1; communication model)
reasoning *SMAKB* tasks 2,3, 4.2, 5.2, expertise model
agent-interaction *CoordinatorBehaviour* (task 4.1 and 5.2), main-task
SMABehaviour (Coordinating task).

Resource Manager Service, which is used to share and store the calendars persistently between Personal Agents. Regarding the design of this agent, collecting the requirements from the analysis models (Agent, Task, Communication, Coordination, Expertise, and Organisation), this agent has no sensors, FIPA communications capabilities (FIPA 2002), and only simple reasoning capabilities. The architecture selected for this agent is a deliberative one, which can be implemented with JADE using Jess Behaviours.

Conclusions

This chapter has shown the main modelling concepts of MAS-CommonKADS and how the agent metaphor can be combined with standard object-oriented and knowledge engineering techniques.

MAS-CommonKADS analyses the multi-agent system from different perspectives (performed tasks, reasoning, agent interaction, user interaction, multi-agent structure) that provide a description easy to implement in a particular multi-agent framework.

The main strengths of the methodology are its simplicity, extending in a natural way the standard software engineering methods. In addition, MAS-CommonKADS takes into account reusability at all levels of the models, making it easy to reuse analyses and designs from previous projects. The main weaknesses of the methodology are its limited support in design, testing, and coding.

There are two supporting tools for applying the methodology, a Java CASE tool and a Plug-In for Rational Rose.

MAS-CommonKADS has been applied by our research groups in projects of intelligent network management and development of hybrid, symbolic-connectionist systems. Other researchers have also successfully applied MAS-CommonKADS for flights reservation (Arenas & Barrera-Sanabria, 2002), development of organisational memories (Arenas & Barrera-Sanabria, 2003), e-learning (Salcedo, 2003), paper review assistant (Marci de Oliveira, 2000), or simulation of stakeholders (Henesey, Notteboom & Davidsson, 2003).

Acknowledgments

We would like to thank the editors and anonymous reviewers of this chapter for their comments, all the students who have contributed to its development, and all the partners of the Collaborator project.

References

Arenas, A. & Barrera-Sanabria, G. (2002). Applying the MAS-CommonKADS methodology to the Flights reservation problem: Integrating coordinating and expertise. In *Proceedings of 5th Joint Conference on Knowledge-based Software Engineering*. IOS Press.

Arenas, A. & Barrera-Sanabria, G. (2003). Modelling intelligent agents for organisational memories. In *Knowledge-based Intelligent Information and Engineering Systems, Lecture notes in computer science 2773* (pp. 430-437). Berlin: Springer-Verlag.

Beck, K. (1999). *Extreme programming explained: Embrance change*. Boston: Addison-Wesley Professional.

Beck, K. & Cunningham, W. (1989). A laboratory for teaching object-oriented thinking. In *OOPSLA'89 Conference Proceedings*, New Orleans, LA, USA (Vol. 17, pp. 1-6).

Bauer, B., Müller, J. P., & Odell, J. (2001). Agent UML: A formalism for specifying multiagent interaction. In P. Ciancarini & M. Wooldridge (Eds.), *Agent-oriented software engineering* (pp. 91-103). Berlin: Springer-Verlag.

Bergenti, F., Garijo, M., Poggi, A., Somacher, M. & Velasco, J. R. (2002). Enhancing collaborative work through agents. In *VII Convegno dell'Associazione Italiana per l'Intelligenza Artificiale.* Available online from *http://www-dii.ing.unisi.it/aiia2002/paper.htm*

Cowan, D. & Griss, M. (2002). *Making software agent technology available to enterprise applications*, Technical report HP-2002-211, Hewlett-Packard Labs.

FIPA (2004). FIPA Agent Communication Language Specifications. Available online *http://www.fipa.org/repository/aclspecs.html*

FIPA (2004). The Foundation for Intelligent Physical Agents. Available online *http://www.fipa.org*

Henderson-Sellers, B. & Edwards, J.M. (1994). *BOOKTWO of object-oriented knowledge: The working object*. Sydney: Prentice Hall.

Henesey, L., Notteboom, T., & Davidsson, P. (2003). Agent-based simulation of stakeholders relations: An approach to sustainable port terminal management. In *Proceedings of the 13th International Association of Maritime Economist (IAME) Conference,* Busan, Korea (pp. 314-331).

Iglesias, C. A. (1998). *Definition of a methodology for the development of multiagent systems.* PhD thesis, Technical University of Madrid (in Spanish).

Iglesias, C. A. & Garijo, M. (1999). UER technique: Conceptualisation for agent-oriented development. In *Proceedings of the 3rd World Multiconference on Systemics, Cybernetics and Informatics (SCI'99) and 5th International Conference on Information Systems Analysis and Synthesis (ISAS'99)* (Vol. 5, pp. 535-540). International Institute of Informatics and Systemics.

Iglesias, C. A., Garijo, M., González, J. C., & Velasco, J. R. (1998). Analysis and design of multiagent systems using MAS-CommonKADS. In N. Callaos & M. Torres (Eds.), *Intelligent agents IV: Agent theories, architectures and languages*, LNAI 1365 (pp. 313-326). Berlin: Springer-Verlag.

ITU-T-Z.100 (1994). *CCITT specification and description language (SDL).* Technical report, ITU-T.

ITU-Z.120 (1996). *Message Sequence Chart (MSC). ITU. Z.120*. Technical report, ITU-T. Geneva.

Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G. (1992). *Object-oriented software engineering. A use case driven approach*. New York: ACM Press.

Kruchten, P. (2000). *The rational unified process: An introduction* (2nd ed.). Reading, MA: Addison-Wesley.

Marci de Oliveira, H. (2002). *Técnicas para Projeto e Implementaçao de Agentes de Software*, Master Thesis, Universidade Estadual de Maringá, Brasil.

Rumbaugh, J., Blaha, M., Premerlani, W., & Eddy, V. F. (1991). *Object-oriented modelling and design*. Upper Saddle River, NJ: Prentice-Hall.

Salcedo, P. (2003). Inteligencia Artificial Distribuida y Razonamiento basado en casos en la Arquitectura de un Sistema Basado en el Conocimiento para la Educación a Distancia. *Revista de Ingeniería Informática*, 9.

Salic, B. (2004). UML 2.0: Exploiting abstraction and automation. Available online *http://www.sdtimes.com/opinions/guestview_098.htm*

Schreiber, G., Akkermans, H., Anjewierden, A., deHoog, R., Shadbolt, N., VandeVelde, W. & Wielinga, B. (1999). *Knowledge engineering and management: The commonKADS methodology*. Cambridge, MA: MIT Press.

Wirfs-Brock, R., Wilkerson, B. & Wiener, L. (1990). *Designing object-oriented software*. Upper Saddle River, NJ: Prentice-Hall