A Methodological Proposal for Multiagent Systems Development Extending CommonKADS*

Carlos A. Iglesias[†**], Mercedes Garijo[‡],
José C. González[‡] and Juan R. Velasco[‡]

[†]Dep. de Teoría de la Señal, Comunicaciones e Ing. Telemática, E.T.S.I. Telecomunicación
Universidad de Valladolid, E–47011 Valladolid, Spain
{cif@tel.uva.es}
[‡]Dep. de Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación
Universidad Politécnica de Madrid, E–28040 Madrid, Spain
{mga,jcg,juanra}@gsi.dit.upm.es

Abstract

The application of agent technology to real applications needs the development of a methodology which supports all the Software Development Life Cycle (SDLC) of an agent based system including its management. This paper proposes an extension of CommonKADS for fitting the characteristics of the agent approach into the SDLC and the definition of a new model, the coordination model, for describing the coordination protocols.

1 The need for a methodology

In spite of the great interest in the agent technology in the scientific community, and the introduction of terms such as *Agent-Based Software Engineering* (Wooldridge & Fisher, 1994) and *Multi-Agent Systems engineering* (Müller, 1992), there has been little work in defining a methodology for designing agents and agent based systems as mentioned in (Jennings, 1995; Jennings & Wooldridge, 1995; Müller, 1992).

A first approach for the definition of a general methodology for multiagent systems (MAS) is here presented, which has been developed because of the need to apply a multiagent platform to different applications.

2 The MAS-CommonKADS approach

The definition of a software engineering methodology does not usually begin from scratch, but is a refinement cycle, adding the new aspects and perspectives of the systems and languages and integrating the successful ingredients of previous methodologies. This is the approach followed here. Our methodology is called MAS-CommonKADS because it is an extension of the CommonKADS methodology, adding the aspects relevant to MAS.

2.1 CommonKADS overview

CommonKADS (de Hoog *et al.*, 1993) is a methodology designed for the development of knowledge based systems (KBS) analogous to methods of software engineering. The development of these methods has been

*This research is funded in part by the Commission of the European Communities under the ESPRIT Basic Research Project *MIX: Modular Integration of Connectionist and Symbolic Processing in Knowledge Based Systems*, ESPRIT-9119. The MIX consortium is formed by the following institutions and companies: Institute National de Recherche en Informatique et en Automatique (INRIA–Lorraine/CRIN–CNRS, France), Centre Universitaire d'Informatique (Université de Genève, Switzerland), Institute d'Informatique et de Mathématiques Appliquées de Grenoble (France), Kratzer Automatisierung (Germany), Fakultät für Informatik (Technische Universität München, Germany) and Dep. Ingeniería de Sistemas Telemáticos (Universidad Politécnica de Madrid, Spain).

**This research was partly done while the first author was visiting the Dep. Ingeniería de Sistemas Telemáticos (Universidad Politécnica de Madrid).

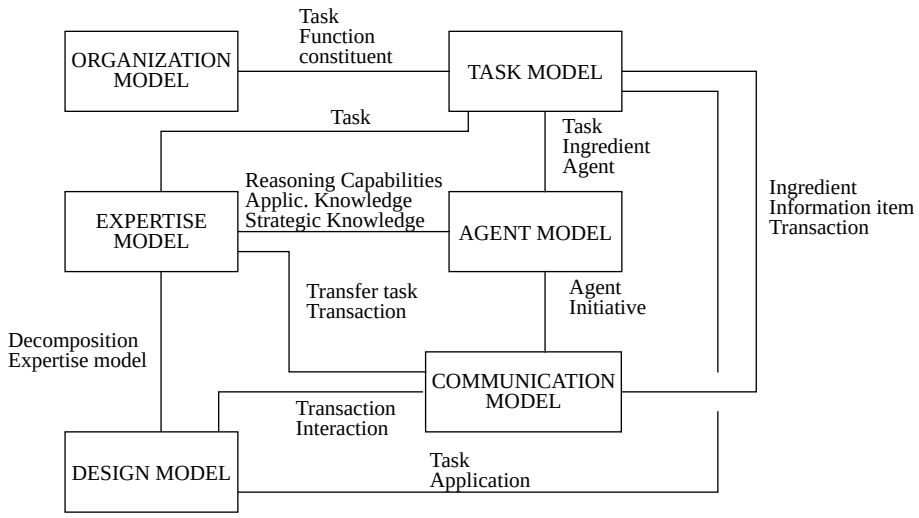

FIGURE 1: The CommonKADS Model Set from (Schreiber *et al.*, 1994; p.2)

funded by the European Community's ESPRIT program from 1983 to 1994. A succinct overview of CommonKADS can be found in (Schreiber *et al.*, 1994).

The CommonKADS methodology follows an approach to Knowledge Based System Development as the building of a number of separate models that captures salient features of the system and its environment.

The process of KBS development consists of filling a number of model templates. Associated to these model templates are model states that characterise the landmark moments in the development of the model. These landmark states allow the management of the project, whose development proceeds in a cyclic and risk-driven way.

The CommonKADS model set is shown in figure 1. There are six defined models:

- *Organisation model: (OM)* is a tool for analysing the organisation in which a KBS is going to be introduced.
- *Task model: (TM)* is used to describe at a general level the tasks that are performed or will be performed in the organisational environment where the proposed KBS will be installed in the future, and provides the frame for the distribution of tasks to the agents.
- *Agent model: (AM)* an agent is an executor of a task. It can be human, computer software or any other entity capable of executing a task. This model describes the capabilities and characteristics of the agents.
- *Communication model: (CM)* details the exchange of information between the different agents involved in executing the tasks described in the Task Model.
- *Expertise model: (EM)* this is the focus of the CommonKADS methodology and models the problem solving knowledge used by an agent to perform a task. The Expertise Model distinguishes between application knowledge and problem solving knowledge (Wielinga *et al.*, 1993; p. 10). It is divided into three sub-levels: domain level (declarative knowledge of the domain), inference level (a library of generic inference structures) and task level (ordering the inferences).
- *Design model: (DM)* while the other five models deal with the analysis of the KBS, this model is intended to describe the architecture and technical design of the KBS in order to implement it.

2.2 Limitations of CommonKADS for MAS

CommonKADS was not designed for developing MAS. The main restrictions for the direct application of CommonKADS to MAS come from the CommonKADS CM:

- The CM deals mostly with human-computer interaction. It is very restrictive for computer-computer interaction. In the same way as the *interaction model* (Kingston, 1992), previous to CommonKADS CM, both model the inputs and outputs of the tasks carried out between a user and a KBS. The primitives of a protocol for complex interactions are not considered.
- According to CommonKADS, task assignment should be performed in a fixed way. However, a restricted form of flexible task assignment can be carried out.(Wærn *et al.*, 1993; p. 19)
- The CM does not deal with multi-partner transactions in a natural manner.

In the following sections, we propose an extension of CommonKADS for MAS:

- First, how the agent approach can be integrated into the SDLC (section 3) is shown.
- Then a new model, the *coordination model* [3], is proposed (section 5). It is an alternative model to the communication model for modelling the interaction between agents. The communication model could still be used for human-computer interaction.
- Finally, an example is presented (section 6).

3 MAS-CommonKADS: software development life cycle model

The overall MAS-CommonKADS methodology for multiagent systems development follows these phases:

- *Conceptualisation.* Elicitacion task to obtain a first description of the problem and determination of use cases which can help to understand informal requirements (Potts *et al.*, 1994) and to test the system.
- *Analysis.* Determination of the requirements of our system starting from the problem statement. During this phase the following models are developed: OM, TM, AM, CM, CoM and EM.
- *Design.* How the requirements of the analysis phase can be achieved by the developing of the DM is determined here. The architecture of both the global multiagent network and each agent is also determined.
- *Coding* and *testing* of each agent.
- *Integration.* The overall system is tested.
- *Operation* and *maintenance.*

3.1 Conceptualisation

The usage of use cases (Jacobson *et al.*, 1992) has been introduced in most of the object oriented methodologies in the last few years (Regnell *et al.*, 1996), especially in the earliest stages of system development. The method of use case modelling presented in (Regnell *et al.*, 1996) has the advantage of being formalized with Message Sequence Charts, which are used for modelling the proposed coordination model.

[3] The selection of a new name is not easy because *cooperation model* was used in KADS-I and *interaction model* was used by Kingston (Kingston, 1992) in an extension to KADS-I

3.2 Analysis

The results of this phase will be the requirements specification of the composite system through the development of the models described before. Only the extensions to CommonKADS are developed in this paper. The steps of this phase are:

1. **Delimitation:** Delimit the MAS system from the external systems. This task was carried out in CommonKADS by the development of the AM. A first version of AM and CoM are obtained. The external (predefined) systems are agentified. Both speech-acts and interchanged data are modelled in the CoM. If there is user interaction, agentify the user and develop a first version of CM.
2. **Decomposition:** The system can be decomposed by the identification of more agents attending to the following guidelines (Bond & Gasser, 1988):
   - *Geographical distribution:* An agent has a unique physical address. From the problem statement it is determined whether the system is geographically distributed (e.g., an intelligent network management system). Each different physical position corresponds to a different agent.
   - *Logical distribution:* Each agent performs one or several functions in the application. An agent is able to perform a task in order to achieve a goal. The study of goal interrelationship determines the autonomy of the agents. The development of the TM can help to determine new agents. The process of assigning different goals to different agents is called the *goal strategy* (Du Bois, 1995; p.112).
   - *Knowledge distribution.* In the case of knowledge acquisition from different experts (Dieng, 1994) or the existence of different expert domains (Wooldridge *et al.*, 1991), an agent can be defined for each domain and an EM can be developed for each agent.
3. **Validation:**
   - Each time an agent is decomposed into new agents, these agents should be logically consistent with the previous definition of the agent:
     - The subagents are responsible for achieving the goals of the agent.
     - The subagents should be consistent with the CoM and maintain the same external interactions.
   - Cross validation with the other models (TM, CM, CoM).
   - At least one conflict solving method should be determined for each conflict detected in the scenarios.

3.3 Design

As a result of the analysis phase, an initial set of agents has been determined. During this phase the DM is developed. This phase is extended for MAS and consists of:

- *Application design.* The system is decomposed into sub-modules. For a MAS architecture, the most suitable agent architecture[4] is determined for each agent.
- *Architecture design.* A multiagent architecture is selected here (instead of, for example a blackboard or an object decomposition). For a MAS architecture, the infrastructure of the MAS-system (so-called *network model* (Iglesias *et al.*, 1996)) is determined. The agents (so-called *network agents*) that maintain this infrastructure are also defined.
- *Platform design.* Software and hardware that is needed (or available) for the system.

[4] *Agent architecture* is used for describing a particular agent software/hardware construction (Wooldridge & Jennings, 1995) and *agent model* for the set of requirements (skills, role, etc.) of the CommonKADS AM

3.3.1 Application design

The EM (mainly its task level) and the TM can help us to identify new agents. For each new agent identification, the steps in the decomposition and validation of the analysis should be followed.

The definition of the domain knowledge of each agent is based on the domain level of the EM. The common domain knowledge for most of the agents and the coordination knowledge is extracted from the network model (section 3.3.2).

In this phase, more agents can be identified complying with the following criteria:

- Enabling *coordination* by the allocation of scarce resources. If a resource is needed by several agents, an agent manager (so-called *internal agent* in (Rumbaugh, 1995a)) of the resource can be suitable. This heuristic has the disadvantage of increasing the dependencies between agents and the overload of interactions (Bond & Gasser, 1988).
- Achieving some of the generic goals of *cooperation* (Durfee *et al.*, 1989): duplicating tasks with different performing methods. If a task can be achieved by different problem solving methods, a common usage is to draw a goal graph with all the possibilities (Maurer & Paulokat, 1994; Mylopoulos *et al.*, 1992). After drawing this diagram, several strategies can be followed:
  - Implement all the possibilities in one agent. The agent selects the best one at run-time by means of a planning process.
  - Implement one possibility per agent, and add an *internal agent* for deciding at run time which agents are more suitable for carrying out the task and which is the best solution. The reason for subdividing an agent can be that the methods are very heterogeneous (i.e. symbolic and connectionist methods, different reasoning capabilities required, etc).
  - The global plan can be decomposed into sub-plans assigned to different agents (i.e. contract net). In this case, the commitments of the agents can be specified for prototypical interactions. This specification of both commitments and reasoning will depend on the agent theory selected.

  Once the agents have been identified, whether each agent should be modelled with a deliberative, reactive or hybrid architecture can be determined. This selection depends on the required skills of each agent reflected in the AM, as discussed in (Müller, 1992). The next step is specifying each agent using an agent language as detailed design language (DDL) if there is no suitable agent in our agent library. This description of the agent knowledge can be decomposed in defining the general ontology for all the agents (which can be imposed, for example, for limited access to resources (Brazier *et al.*, 1996)) and the particular behaviour of each agent together with the knowledge and resources needed.

3.3.2 Architecture design

The general architecture of the systems is, of course, a multiagent architecture. Here, the architecture design is subdivided up into the three levels of the network model of a multiagent architecture proposed in (Iglesias *et al.*, 1996).

1. **Network level:** The design decisions on the infrastructure of the multiagent architecture are taken. Which agents are needed to maintain the multiagent society (facilitators, knowledge managers, group coordinators, etc.) are specified. Several questions should be answered, for example:
   - Is an agent name server needed? Should it be centralised or distributed?
   - Is an agent group manager needed?
   - Is a service repository needed? Should it be dynamic or static?
   - Is a broker facility needed?
   - Which telematic protocol is more suitable (http, tcp sockets, mail, etc.)?

- What degree of security is needed?

The interactions needed to perform *network tasks* such as logging-in, logging-out, etc, can be represented in the interactions diagrams of CoM (section 5) defined as SDL tasks. In this way, these diagrams are augmented to show the design decisions.

2. **Knowledge level:** Several design decisions should be taken regarding the management of ontologies:
   - Are there distributed ontologies (Thomas *et al.*, 1995)? Is an ontology manager needed?
   - Should the agents understand different knowledge representation languages (KRLs)? Is a KRLs-translator needed?

   These public ontologies are the domain knowledge that should be common to most of the agents, and can be extracted from the domain level of the EM.

3. **Coordination level:** For sake of reusability, several authors have pointed out the advantages of defining the *Cooperation knowledge* as a reusable module (Jennings & A., 1993) or the definition of reusable coordination protocols *AgenTalk* (Ishida, 1995; Kuwabara *et al.*, 1995). This knowledge, the set of suitable primitives (Finin & Fritzson, 1994; Huang *et al.*, 1995) and the agents to form joint plans are the basis for the definition of this level. This knowledge should be merged (and specialised if needed) with the domain knowledge of each agent derived from the EM in the application design.

3.3.3 Platform design

Based on the architecture design, the agent architecture selected and the non-functional requirements, the selection of the operating system and the software and hardware platform is made.

3.4 Coding and testing

As discussed in (Wooldridge & Fisher, 1994), this is an open question, depending on the usage of a general purpose language or a formal agent language, which can be directly executable or translated to an executable form.

3.5 Global testing, operation and maintenance

The correct behaviour of the global system can be partially tested by using typical scenarios which deal with the possible conflicts and the methods for conflict resolution. Since the global behaviour of the system cannot be determined during analysis or design, because it depends on the particular commitments and agreements between the agents (Huang *et al.*, 1995), simulation of the behaviours is usually needed.

Finally, the MAS is installed and maintained.

4 Modifications to the Agent Model

The agent model has been modified in order to include the characteristic aspects of intelligent agents. The central constituent is the agent. It has five attributes: *name*, *type* (human, new system agent or predefined system agent), *subclass-of*, *role*, *position* and *groups* (agent groups the agent belongs to). Other constituents of this model are:

- *Service*: facilities offered to the rest of agents to satisfy their goals. It can perform one task of the TM, and has five attributes: name, type, task, and ingredients.

- *Goal*: objectives of the agents. The goal has the following attributes: name, description, type and ingredients. The goals can be satisfied according to the reasoning capabilities of the agent described below.
- *Reasoning capabilities*: requirements on the agent's expertise imposed by the task assignment. These are realized by one or several expertise models that model how these capabilities are achieved. How the agent reasons in its environment to achieve its goals is modelled here.
- *General capabilities*: Skills (sensors and effectors to manipulate the environment), and languages the agent understands (Agent Communication Languages and Knowledge Representation Languages).
- *Constraints*: norms, preferences and permissions of the agent. The norms and preferences have special interest in the case of MAS. For example, they are used to model when an agent decides to negotiate and are realized by one or several expertise models.

5 The Coordination Model

Since the CommonKADS CM does not provide a suitable treatment of multiagent interactions, a different CM development based on speech-acts is formulated here, which is called *coordination model (CoM)*. This is an alternative model that can be used when the agents are software agents. The CM of CommonKADS can be used to model further human-machine communication. For the CoM, human agents are agentified and the communication is carried out via speech-acts. The design of the user interface is outside the scope of this extension.

5.1 Model structure

The Coordination model contains four costituents (figure 2):

- *Conversation*: a set of interactions in order to ask for a service or request or update information[5]. It is distinguised by the name and the requested service name.
- *Interaction*: a simple interchange of messages. It has the following attributes: speech-act, agent communication language, knowledge representation language, synchronization (synchronous, asynchronous or future), transmitter, receiver and ingredients.
- *Capabilities*: the skills and knowledge of the initiator of the conversation and the other participants.
- *Protocol*: a set of rules that govern the conversation. It defines the different states and interactions allowed.

5.2 Graphical notation

Three graphical models (with the corresponding textual templates) are used to develop the CoM:

- **Message sequence charts**(Rudolph *et al.*, 1996) are used to represent the different scenarios of the use cases identified. An alternative graphical model are **event trace diagrams** as in the dynamic modelling of OMT (Rumbaugh, 1995b; Rumbaugh, 1995a; Yau *et al.*, 1995). Scenarios in the same way than in ROOM (Selic *et al.*, 1992) describe both how the system is used and the event sequences that occur inside the system (in this case, the different interactions between the agents).
- **Event flow diagrams** are used to model the generic behaviour of the agent, including all the possible interactions and the data/knowledge interchanged in these interactions.

[5]Requesting or updating information is also modelled as a service

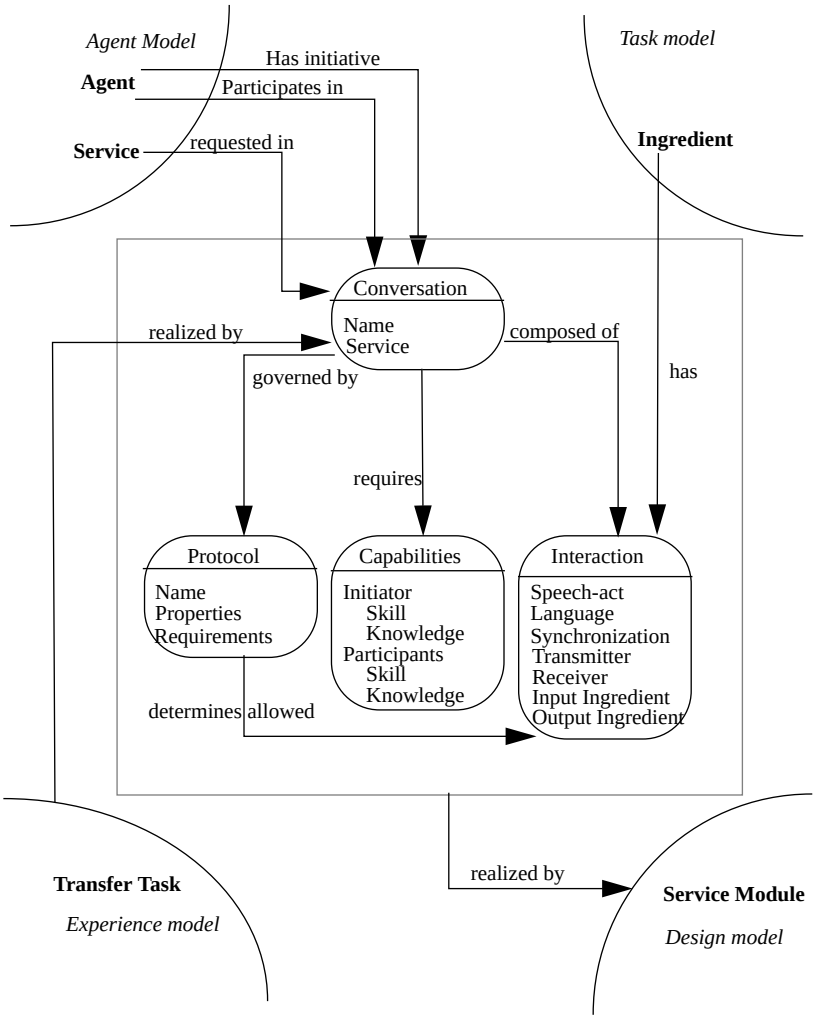

FIGURE 2: Coordination Model

- **Communicating Extended Finite-State Machines** (CEFSMs) of the formal description technique SDL (Turner, 1993) are used for modelling the control flow of the interactions. SDL has the advantages of the FDTs (unambiguous, complete, consistent, tractable specifications) and introduces the advantage of being designed for modelling communication protocols. SDL has been selected from among the available FDTs because of its clear and intuitive graphical language. From this FDT, only the CEFSMs have been used. Block diagrams of SDL are an alternative solution to the event flow diagrams of the previous point.

CEFSMs have the special characteristic for MAS of considering three kinds of events (signals) or stimuli:

- *Message events:* a message received from other agents. These signals are modelled by speech acts.
- *External events:* events from the environment through the sensors: push button, alarm firing, etc.
- *Internal events:* events generated by an agent in order to achieve a goal.

5.3 Landmark states

There are two main landmark states during the developing of this model, apart from the cross-validation between models:

- Identification and description of the conversations (services) between agents (steps 1-6 in section 5.4). During this phase we will assume that every conversation is performed following a client-server protocol. Once this landmark state is achieved (and the agent model is completed), we can produce a prototype for external and internal validation.
- Identification of the negotiation protocols needed (steps 7-9 in section 5.4). Some of the conversations can be relaxed following a negotiation protocol.

5.4 Model development cycle

The steps followed for the development of this model are:

1. Describe the prototypical scenarios between agents. These scenarios can be a further development of the scenarios determined in the conceptualisation phase for the use cases. The scenarios are described using *message sequence charts* (MSCs)(Rudolph *et al.*, 1996). An alternative representation is event trace diagrams. During this first stage, we will consider that every conversation consists of only one single interaction and the possible answer. The objective at this stage of development is to establish the set of conversations (channels) between agents.
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 date interchanged in each interaction. The EM can help us to define the interchanged knowledge structures. These interchanged data are shown in the event flow diagram between squared brackets.
4. Model each interaction with the CEFSMs of SDL specifying speech-acts as inputs/outputs of message events. Alternative representations are discussed in section 7. The inputs and outputs of the SDL process representation can be validated with the MSC diagrams.
5. Each state can be further refined. If the state represents a knowledge task, the inference templates of the CommonKADS library are very useful. While decomposing a state, it can be decomposed into different agents, and thus, the complete process must be repeated.

6. Analyse each interaction and determine its synchronisation: synchronous, asynchronous or future.
7. Determine the receivers of each service request: individual or group and if a coordination protocol such as contract-net is desired. This can be represented in SDL using the names of the agents or group names in the explicit addressing facility.
8. Determine if a cooperation protocol is needed for each conversation. The reasons for using a cooperation protocol can be, among others (Durfee *et al.*, 1989):
   - *Increasing task completion through parallelism.*
   - *Increasing the set or scope of achievable tasks by sharing resources (information, expertise, physical devices, etc.).*
   - *Increasing the likelihood of completing tasks by undertaking duplicate tasks.*
   - *Decreasing the interference between tasks by avoiding harmful interactions.*
   - Resolving conflicts via negotiation protocols. Usually these conflicts need to be assisted by a human agent. The CoM model is used for modelling the *negotiation language category* (protocol, primitives, semanticas and object structure) and partly the *negotiation process category* (procedure and behaviour) according to the classification of negotiation categories by (Müller, 1992; p.213). The other category *negotiation decision category* (preferences and strategies) should be modelled with expertise models.
9. Consult the library of cooperation protocols and reuse a protocol defined previously or define its interaction model. The utilization of HMSC (High level Message Sequence Charts) is very useful for this purpose.

From the CoM we should check the consistency with the AM:

- The source point of an arrow of the initiator agent represents a goal or an event-response.
- Each source point of an arrow is a requested service.
- Each target point of an arrow is a service the agent offers.

This model allows the complete and clear specification of individual interactions. Nevertheless, the global behaviour of the system depends on the sequence of interactions carried out by the agents.

An additional advantage of using SDL is the ability to model coordination protocols which can be reused using tasks.

6 Example

6.1 Statement of the problem

In this section we apply the MAS-CommonKADS methodology to the development of a simplified classification system (in the following SCS). This system will be running in a power plant. The system has to predict if the value of a distinguished variable (e.g., the delivered power) is going up or down or is constant. This classification is done from a set of values collected from sensors and filtered and stored every two minutes by an existing data acquisition system. For the sake of simplicity, we will deal with only two variables (e.g., boiler temperature and pressure).

The managers of the plant have no clear idea of the techniques that should be used for classification, as there is no analytical model available. They consider it a good idea to incorporate several classification modules whose results would be presented every two minutes to a human operator in the control room of the plant.

Based on the assessment received from the system, the operator decides the control actions to be applied to the plant in order to maintain it in the desired state (desired performance, delivered power, pollution level, etc.).

6.2 Analysis

Since this is a *"toy-problem"*, we will neither analyse the risk of the system development, nor the impact of the system in the organisation, but these software management issues should be considered in a real case.

6.2.1 First version of the AM, EM and TM

A first version of the AM is developed as recommended (Wærn & Gala, 1993; p.26) when there are predefined systems (in our case, the collector system). Three agents are identified as attending to geographical distribution criterion: Collector (for collecting data), Classifier and the User. For space limitations, we do not show the CommonKADS templates of any model. The TM of the whole system also shows these three functions.

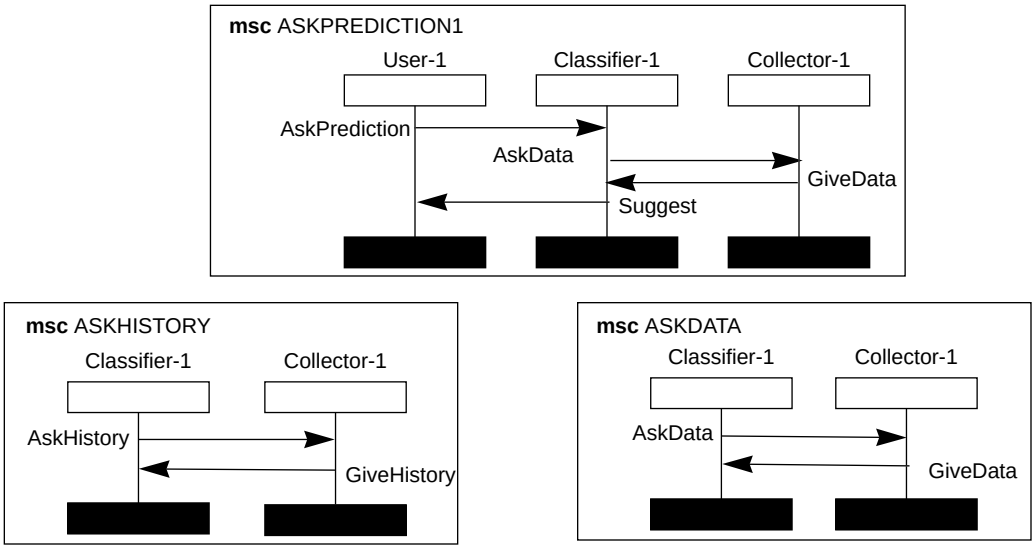

FIGURE 3: Event trace of agent conversations version 1

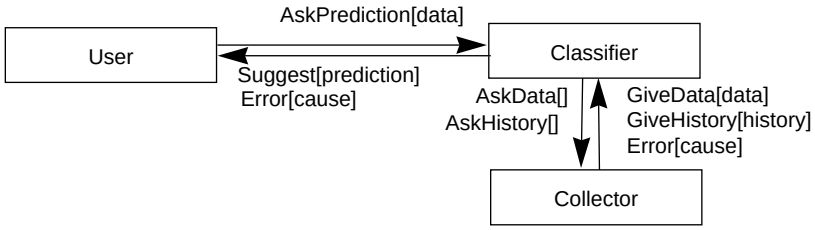

FIGURE 4: Event flow diagram version 1

6.2.2 First version of CoM

The event traces of the prototypical use cases are shown in figure 3, and the corresponding event flow diagram is shown in figure 4. The interlingua shown between squared brackets in the event flow diagram has been obtained from the domain knowledge of the EM.

Only one CEFSM is included in figure 5. This diagram is an enhancement of the first version. It includes some design issues: macro for *NetworkLogIn* and non-functional requirements such as the *Quit* button of the menu.

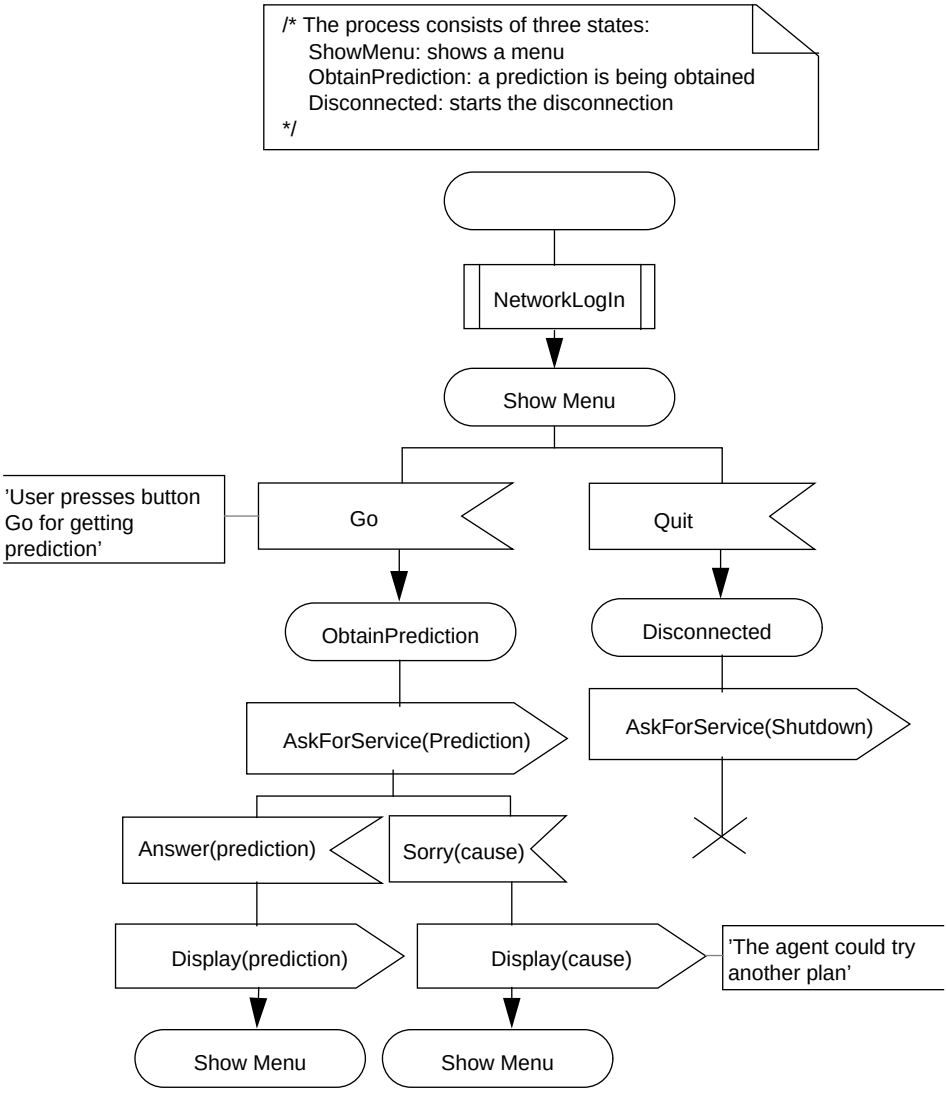

FIGURE 5: CEFSM of service User:Interaction

6.3 Design

Architecture design Since only an agent name server is needed, a centralised network agent is chosen for the system, with Yellow-Pages facilities.

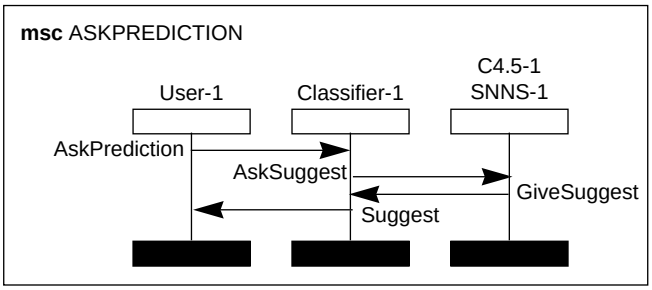

FIGURE 6: Event trace of agent conversations version 2

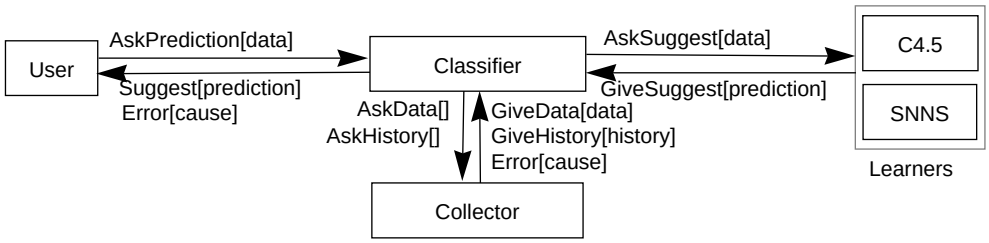

FIGURE 7: Event flow diagram version 2

Application design From the statement of the problem, two methods are considered for predicting: a neural one and a symbolic one, which will be carried out by two agents: *Ag_Neural* and *Ag_ID3*, respectively. An additional agent (*Selector*) is added to select at run-time the best prediction. The event trace and event flow diagrams of this solution are shown in figures 6 and 7. A new version of AM and CoM are needed to add the new interactions, which are not shown here for space limitations. The Agent Description Language (ADL) (González *et al.*, 1994) is selected to describe the agents. The ADL code is included in appendix A.

Platform design The MIX platform (Iglesias *et al.*, 1994; González *et al.*, 1995) is selected because of its facilities for integration of symbolic and connectionist techniques.

6.4 Coding and testing

The services are programmed in C++ using the libraries of the MIX Platform. The diagrams of the *Coordination model* are a good specification of the interactions. The platform is then tested.

7 Related work and Conclusions

The main purpose of this paper is the definition of a complete software engineering methodology to MAS. This has been made by the extension of the CommonKADS methodology in two ways:

- Integration of the multiagent characteristics in a SDLC.
- Development of a new model for interagent communication: the coordination model.

The CoM can be used in conjunction with CommonKADS (or another knowledge-engineering methodology) or on its own for describing coordination protocols for MAS systems.

SDTs (Kuwabara *et al.*, 1995; Fisher *et al.*, 1996; Barbuceanu & Fox, 1996), Petri nets (Ismail *et al.*, 1996) and statecharts (Harel, 1987; Coleman *et al.*, 1992) are different alternatives to CEFSM for dynamic modelling. The CEFSMs of SDL have been selected because they are well known, highly visual, intuitive and a good input to the implementation. The mapping between SDL and MSC concepts makes the utilization of both languages and the availability of tools easy. They seem very suitable for modelling the interactions via primitives such as KQML (Finin & Fritzson, 1994), which are modelled in SDL as signals. Its nesting facility via macros makes the reusability of protocols and its expansion easy. Nevertheless, the other alternatives (especially Petri nets and statecharts) are not excluded.

In contrast to other coordination languages (Kuwabara *et al.*, 1995; Barbuceanu & Fox, 1996) which are environments for describing and debugging coordination protocols, our CoM is not language dependent, and we have been successful in translating some of the protocols defined in (Kuwabara *et al.*, 1995; Barbuceanu & Fox, 1996). These results have shown that our model does not need (informal) textual explanation as do the others.

In order to provide *tool support*, there is no current single tool that supports all the proposed graphical models in this article. We are working now on an integrated environment for MAS systems that support all the SDLC, but it is still in the early stage.

This methodology is been applied to two research projects in different applications. Although the examples have been relatively small, we can conclude that besides the obvious advantage of using a methodology, the advantages of documentation of this methodology and its facility to describe coordination protocols should be stressed.

References

Barbuceanu, M. & Fox, M. S. (1996). Capturing and modeling coordination knowledge for multi-agent systems. *Journal on Intelligent and Cooperative Information Systems*. To appear.

Bond, A. H. & Gasser, L., (Eds.) (1988). *Readings in Distributed Artificial Intelligence*. Morgan Kaufmann Publishers: San Mateo, CA.

Brazier, F., van Eck, P., & Treur, J. (1996). Modelling cooperative behaviour for resource access in a compositional multi-agent environment. In *6th Workshop on Knowledge Engineering Methods and Languages*, Paris, France.

Coleman, D., Hayes, F., & Bear, S. (1992). Introducing objectcharts or how to use statecharts in object-oriented design. *IEEE Transactions on Software Engineering*, 18(1):9–18.

de Hoog, R., Martil, R., Wielinga, B., Taylor, R., Bright, C., & van de Velde, W. (1993). The CommonKADS model set. ESPRIT Project P5248 KADS-II/M1/DM..1b/UvA/018/5.0, University of Amsterdam, Lloyd's Register, Touche Ross Management Consultants & Free University of Brussels.

Dieng, R. (1994). Agent-based method for building a cooperative knowledge-based system. In *Proceedings of FGCS'94 Workshop on Heterogeneous Cooperative Knowledge-Bases, Lecture Notes in Computer Science (LNCS)*, Tokyo, Japan. Springer-Verlag: Heidelberg, Germany.

Du Bois, P. (1995). *The Albert II Language. On the Design and the Use of a Formal Specification Language for Requirements Analysis*. PhD thesis, Facultés Universitaires Notre-Dame de la Paix, Namur (Belgium).

Durfee, E. H., Lesser, V. R., & Corkill, D. D. (1989). Trends in cooperative distributed problem solving. *IEEE Transactions on Knowledge and Data Engineering*, 1(1).

Finin, T. & Fritzson, R. (1994). KQML: A language and protocol for knowledge and information exchange. In *Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence*, pages 126–136, Lake Quinalt, WA.

Fisher, K., Müller, J., & Pischel, M. (1996). AGenDA: A general testbed for distributed artificial intelligence applications. In O'Hare, G. M. P. & R, J. N., (Eds.), *Foundations of Distributed Artificial Intelligence*, pages 401–427. John

Wiley & Sons.

González, J. C., Velasco, J. R., & Iglesias, C. A. (1994). A common platform for the mix project: Perspective from the msm architecture. Internal report of the ESPRIT-9191 project UPM/DIT/LIA 2/94, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

González, J. C., Velasco, J. R., Iglesias, C. A., Alvarez, J., & Escobero, A. (1995). A multiagent architecture for symbolic-connectionist integration. Technical Report MIX/WP1/UPM/3.2, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

Harel, D. (1987). Statecharts: a visual formalism for complex systems. *Sci. Computer Program*, 8:231–247.

Huang, J., Jennings, N. R., & Fox, J. (1995). Agent-based approach to health care management. *Applied Artificial Ingelligence*, 9(4):401–420.

Iglesias, C. A., Alvarez, J., & Escobero, A. (1994). A multiagent architecture for symbolic-connectionist integration. Reference Manual MIX/WP1/UPM/4.0, Dep. Ingeniería de Sistemas Telemáticos, E.T.S.I. Telecomunicación, Universidad Politécnica de Madrid.

Iglesias, C. A., González, J. C., & Velasco, J. R. (1996). MIX: A general purpose multiagent architecture. In Wooldridge, M., Müller, J. P., & Tambe, M., (Eds.), *Intelligent Agents II (LNAI 1037)*, pages 251–266. Springer-Verlag: Heidelberg, Germany.

Ishida, T. (1995). Parallel, distributed, and multi-agent production systems — a research foundation for distributed artificial intelligence. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, pages 416–422, San Francisco, CA.

Ismail, M. A., Salah, A. I., & Bader, O. (1996). Petri nets as a formal representation for dynamic object modeling. In *6th Workshop on Knowledge Engineering Methods and Languages*, Paris, France.

Jacobson, I., Christerson, M., Jonsson, P., & Övergaard (1992). *Object-Oriented Software Engineering. A Use Case Driven Approach*. ACM Press.

Jennings, N. R. (1995). Agent software. In *Proc. of UNICOM Seminar on Agent Software*, pages 12–27, London, UK.

Jennings, N. R. & A., P. J. (1993). Design and implementation of archon's coordination module. In *Proc. Workshop on Cooperating Knowledge Based Systems*, pages 61–82, Keele, UK.

Jennings, N. R. & Wooldridge, M. (1995). Applying agent technology. *Applied Artificial Intelligence*, 9(6):357–370.

Kingston, J. (1992). The model of interaction. *Newsletter of BCS SGES Methodologies Interest Group*, 1. Also available from AIAI as AIAI-TR-115.

Kuwabara, K., Ishida, T., & Osato, N. (1995). AgenTalk: Coordination protocol description for multiagent systems. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95)*, page 455, San Francisco, CA.

Maurer, F. & Paulokat, J. (1994). Operationalizing conceptual models based on a model of dependencies. In *Proceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94)*, pages 508–512, Amsterdam, The Netherlands.

Müller, H. J. (1996). (Multi)- agent systems engineering. In *Second Knowledge Engineering Forum*, Karlsruhe.

Mylopoulos, J., Chung, L., & Nixon, B. (1992). Representing and using nonfunctional requirements: A process-oriented approach. *IEEE Transactions on Software Engineering*, 18(6):483–497.

O'Hare, G. M. & Jennings, N. R., (Eds.) (1996). *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons.

Potts, C., Takahashi, K., & Anton, A. (1994). Inquiry-based scenario analysis of system requirements. Technical Report GIT-CC-94/14, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA, Atlanta, GA, USA.

Regnell, B., Andersson, M., & Bergstrand, J. (1996). A hierarchical use case model with graphical representation. In *Proceedings of ECBS'96, IEEE International Symposium and Workshop on Engineering of Computer-Based Systems*.

Rudolph, E., Grabowski, J., & Graubmann, P. (1996). Tutorial on message sequence charts (MSC. In *Proceedings of FORTE/PSTV'96 Conference*.

Rumbaugh, J. (1995a). OMT: The development model. *JOOP Journal of Object Oriented Programming*, pages 8–16, 76.

Rumbaugh, J. (1995b). OMT: The dynamic model. *JOOP Journal of Object Oriented Programming*, pages 6–12.

Schreiber, A. T., Wielinga, B. J., & Van de Velde, J. M. A. W. (1994). CommonKADS: A comprehensive methodology for KBS development. Deliverable DM1.2a KADS-II/M1/RR/UvA/70/1.1, University of Amsterdam, Netherlands Energy Research Foundation ECN and Free University of Brussels.

Selic, B., Gullekson, G., McGee, J., & Engelberg, J. (1992). Room: An object-oriented methodology for developing real-time systems. In *CASE'92 Fifth International Workshop on Computer-Aided Software Engineering*, Montreal, Quebec, Canada.

Thomas, F., Pawel, S., & Bogdan, F. (1995). Meta-level-architecture for distributed second generation knowledge based systems using CORBA-standard. In *Proceedings of the Second International Symposium on Autonomous Decentralized Systems (ISADS5)*, pages 368–374, Phoenix, Arizona, USA.

Turner, K. J., (Ed.) (1993). *Using Formal Description Techniques*. John Wiley and Sons.

Wærn, A. & Gala, S. (1993). The common kads agent model. Technical report ESPRIT Project 5248 KADS-II/M4/TR/SICS/002/V.1.1, Swedish Institute of Computer Science and ERITEL.

Wærn, A., Höök, K., & Gustavsson, R. (1993). The common kads communication model. Technical report ESPRIT Project 5248 KADS-II/M3/TR/SICS/006/V.2.0, Swedish Institute of Computer Science and ERITEL.

Wielinga, B. J., van de Velde, W., Schreiber, A. T., & Akkermans, H. (1993). Expertise model definition document. deliverable DM.2a, ESPRIT Project P-5248 /KADS-II/M2/UvA/026/1.1, University of Amsterdam, Free University of Brussels and Netherlands Energy Research Centre ECN.

Wooldridge, M. & Fisher, M. (1994). Agent-based software engineering. Unpublished seminar, first presented at Daimler-Benz, Berlin, Germany.

Wooldridge, M. & Jennings, N. R. (1995). Agent theories, architectures, and languages: A survey. In Wooldridge, M. & Jennings, N. R., (Eds.), *Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890)*, pages 1–39. Springer-Verlag: Heidelberg, Germany.

Wooldridge, M., O'Hare, G. M. P., & Elks, R. (1991). FELINE — a case study in the design and implementation of a co-operating expert system. In *Proceedings of the Eleventh European Conference on Expert Systems and Their Applications*, Avignon, France.

Yau, S. S., Yeom, K., Gao, B., Li, L., & Bae, D.-H. (1995). An object-oriented software development framework for autonomous decentralized systems. In *Proceedings of the Second International Symposium on Autonomous Decentralized Systems (ISADS5)*, pages 405–411, Phoenix, Arizona, USA.

A ADL Code

```
#DOMAIN "example"
#YP_SERVER "tcp://madrazo.gti.dit.upm.es:6050"
#COMM_LANGUAGE ckrl
#ONTOLOGY example.ckrl


AGENT YP_Agent -> YPAgent
END YP_Agent

AGENT Interface -> BaseAgent
  GOALS
    Interaction: CONCURRENT interaction
  RESOURCES
    REQ_LIBRARIES: "inter-funct.C"
    REQ_SERVICES: Start_Control
END Interface

AGENT Selector -> BaseAgent
RESOURCES
REQ_LIBRARIES: "selec_funct.C"
REQ_SERVICES: give_sample;
suggest
  CONTRACT_POLICY eval_suggestion
REQ_MSG_STRUCT example::cost
  SERVICES
start_control: CONCURRENT start_control
END Selector

AGENT Collector -> BaseAgent
RESOURCES
REQ_LIBRARIES: "collec_funct.C"
INTERNAL_OBJECTS
history -> History
GOALS
get_data: CONCURRENT GetData
SERVICES
  give_sample : GiveSample
  ANS_MSG_STRUCT example::Vector;
  give_history: GiveHistory
  REQ_MSG_STRUCT example::Depth
  ANS_MSG_STRUCT example::Vector
END Collector

// *********************
// Learning Agents
// *********************

CLASS Classifier -> BaseAgent
RESOURCES
  REQ_SERVICES: Give_History
SUBSCRIBE_TO: Learning_Group
  GOALS
  daemon : CONCURRENT virtual
SERVICES
    suggest: CONCURRENT virtual
  REQ_MSG_STRUCT example::Vector
  ANS_MSG_STRUCT example::Suggestion
END Classifier

AGENT Ag_Neural -> Classifier
RESOURCES
REQ_LIBRARIES: "learn_funct.C"
  GOALS
  daemon : CONCURRENT LearningNeural
  SERVICES
  suggest: CONCURRENT SuggestionNeural
  COST SNNS_cost_function
  REQ_MSG_STRUCT example::StructFile
      ANS_MSG_STRUCT example::Cost
END Ag_Neural

AGENT Ag_ID3 -> Classifier
  RESOURCES
REQ_LIBRARIES: "learn_funct.C"
  GOALS
  daemon: CONCURRENT LearningSymbolic
  SERVICES
  suggest: CONCURRENT SuggestionSymbolic
   REQ_MSG_STRUCT example::Vector
   ANS_MSG_STRUCT example::Suggestion
  COST C45_cost_function
  REQ_MSG_STRUCT example::StructFile
    ANS_MSG_STRUCT example::Cost
END Ag_ID3
```