CHAPTER 14

The Evolution of the CooperA Platform

LORENZO SOMMARUGA
NIKOS M. AVOURIS
MARC H. VAN LIEDEKERKE

1 INTRODUCTION

Despite the considerable progress of DAI theory and foundations during the recent years, one can observe that there are still many interesting areas for investigation in the field, often described as "the open questions of Distributed Artificial Intelligence" (Gasser, 1992) (see Chapter 1). In addition, the development of DAI experimental platforms (Chapter 3) can often prove to be a long tedious process, involving many phases.

The CooperA platform, described here, has evolved in the frame of such long-standing research effort, during which many diverse research issues have been addressed. These include, among others, multiple-domain experts knowledge acquisition, heterogeneous knowledge representation, interagent communication, user interaction, flexible coordination, and domain-independent cooperation mechanisms.

This chapter presents the major results of the project, which took the form of two experimental prototypes, called CooperA (Sommaruga et al., 1989b) and CooperA-II (Sommaruga and Shadbolt, 1994). The main characteristics of these prototypes are described in an incremental manner, which demonstrates the increased complexity and sophistication of the system in the context of a five-year effort, reflecting in many aspects the shifting interests of the DAI discipline.

The driving force of the reported experimentation has been the need to meet in the most efficient and effective way the requirements of a specific real-life
application domain of high complexity. This application has provided our effort with a constant point of reference, which has given the status of a testbed to the developed software platform. Hanks et al. (1993) observe that the lack of controlled experimentation in an area of research is an indication of lack of maturity (see also Chapters 3 and 15). The fact that DAI has a long tradition of reported controlled experiments and testbeds since its early days (see Chapter 3), a tradition followed by the reported CooperA system, should be considered as an indication of quality and good prospects of our discipline.

The CooperA experiment took place in the frame of the research project Chemical Emergencies Management (ChEM) of the Commission of the European Community. The main aim of ChEM was the development of tools and techniques for the management of emergency situations involving electrical equipment which contain polychlorinated biphenyls (PCBs), a group of widely used hazardous chemical substances (see Chapter 4 for a discussion of other DAI applications). We investigated various knowledge-based system architectures in this process. We were soon driven to use DAI techniques by the high complexity of the domain, the high modularity of the domain knowledge, the presence of multiple-domain experts, and the requirements of high flexibility in cooperative decision making.

It is, however, outside the scope of this description to provide a detailed presentation of the application domain and its requirements. For more details on this subject one should refer to Sommaruga et al. (1989a) and Avouris et al. (1989), where the DChEM (distributed ChEM) application is presented, and to Avouris (1995), where an analysis of environmental problem solving in relation to DAI techniques can be found.

Instead, more emphasis is provided here to the architectural characteristics of the developed prototypes and the way issues like interagent communication, cooperation, control, heterogeneous knowledge representation, user interaction, and flexibility in agent behavior programming are addressed by CooperA and CooperA-II.

In more detail, we present first the early phases of the project, when the domain knowledge was structured in a rule-based monolithic expert system. Subsequently the multiagent CooperA architecture is presented, through which the knowledge can be distributed in multiple agents and the problem can be solved in a cooperative way. A detailed description of the platform, the structure of the Agent, the communication mechanisms, the user interface, and the CooperA User Agent is offered.

After a discussion of the limitations of this first prototype, we proceed with a description of the CooperA-II experiment results. This phase focuses on dynamic control of the agent's social behavior, through the use of cooperative heuristics. Our models are based on the assumption that cooperative behavior should be considered as a domain-independent characteristic, and therefore the agents should be provided with adequate data structures, mechanisms, and knowledge which support this behavior (see also Chapters 12 and 13). A discussion on how the developed cooperative heuristics can result in programmable
macrobehaviors, and how through them agent behavior programming can be affected, is also included.

The final part provides a comparative discussion of the various phases of the project and comments on the final results.

2 BACKGROUND AND RATIONALE FOR CooperA

The first phase of the ChEM project resulted in modelling of the domain of Chemical Emergencies Management and the development of a production-rule-based expert system (ES). The major effort during this phase has been dedicated to knowledge acquisition which involved a group of knowledge engineers, and a community of domain experts.

The ES which was decided to be built in order to support decision makers for threat estimation during management of chemical accidents had to incorporate the knowledge of experts in the various domains involved and had to reason even with uncertain and incomplete information. Relying on this expertise, the system had to provide the user with a picture of the situation at any moment during the emergency, based on the available information at that time.

The KB built during this phase contained knowledge about accidents involving the whole family of polyhalogenated aromatics, and was reported in Argentesi et al. (1987). It contained approximately 1500 production rules. The ES could make an estimation of the threat and produce a relative report. The organization of the KB and the flow of data, which can be seen in Figure 14.1, was based on a two-level structure: in the higher level the semantics of the

![Diagram](image-url)

**Figure 14.1** Organization of the KB of ChEM.
problem where defined (threat level and intermediate variables), while the lower level contained technical knowledge of the systems involved.

The fact that the knowledge involved could be organized in relatively independent units, which contribute to the solution in a dynamic way, led us to establish that a decoupling of the various modules was a necessary next phase of the project, leading to a distributed ChEM (DChEM).

This decision has been confirmed by the knowledge acquisition experience, which involved experts from different fields and backgrounds with different perspectives and sometimes conflicting contributions. This made it difficult to build one coherent KB.

Various studies and proposed architectures, influenced by the blackboard model and other DAI techniques have been proposed during this design phase (Avouris et al., 1988a,b). However, despite the intense theoretical activity in the DAI area of worldwide, most of the environments developed at that time were experimental and did not provide typical knowledge engineering tools, necessary for eliciting, structuring, and representing the knowledge to build the various modules/agents of the distributed system. Thus, the approach that we took was to extend an existing knowledge engineering environment, by building into it the structures and mechanisms for handling a community of interacting ESs. In this way, multiagent DAI techniques were introduced, also supported by the fact that a collection of cooperating agents corresponding to different domain experts could model more naturally our particular kind of problem solving.

The result of this effort has been the generic testbed for distributed KB applications, called CooperA (cooperating agents) (Sommaruga et al., 1989b).

DChEM has been used for prototyping the CooperA testbed. A community of agents has been developed and various experiments with control structures and communication mechanisms have been tried.

In a first step, we split the existing KB, in five self-contained pieces of knowledge. We called each part an "expert," containing the domain expertise in a particular area. For each of them, we developed a KB system, all ESs in their own right, handling their own I/O, based on different knowledge representation paradigms such as production rules, frames, first-order logic, etc. Each expert was capable of solving a problem autonomously, soliciting the necessary input data directly from the user. However, since much of the solicited data could have been elaborated in cooperation with another expert, instead of merely asking it from the user, the CooperA environment has been used to integrate this set of loosely coupled KBs, making some additions and light modifications to the existing KBs, and transforming them into agents.

3 CooperA

The first CooperA prototype is a software environment supporting the cooperation of heterogeneous, distributed, semiautonomous KB systems presented to the user via a customized user interface.
KB systems or modules are transformed to application agents which can be incrementally and selectively integrated into one system (see Chapters 12 and 13). Each expert module incorporates a self-description mechanism, which allows the shell to integrate it in the community during the group configuration phase. Thus, various alternative expert communities can be tried for the solution of a certain problem.

The user of CooperA can interact with the community of the agents through a special user interface agent. Special attention has been paid to the user interface which incorporates active modeling of the system, so that the user can see the flows of interaction among the agents (see Chapter 21). This satisfies the "comprehension of the problem-solving activity" requirement, a serious problem of the first ChEM prototype.

One of the capabilities of the CooperA user interface is that it can visualize graphically the problem-solving activity in the system and that it can let the user have access to the tools and commands available in a graphical way. For instance, a number of system commands and utilities are available through a graphical representation in the so-called workbench window.

### 3.1 The CooperA Architecture

The structure of CooperA can be described in terms of layers (see also Chapters 12, 13, and 15). They include

1. *The CooperA Kernel*. It represents the cooperation shell and consists of a collection of initialization procedures and system facilities that each agent can use; it is built on top of an existing language.

2. *The Message-Passing Mechanism*. It supports communication between agents by means of functions which handle and control messages. Messages play an important role in the cooperation: only through them is information exchanged and hence communication made possible.

3. *The Collection of CooperA System Agents*. It is a set of agents created by CooperA independent of the application; they perform basic operations of common interest. For example, one of the CooperA system agents is the user agent which handles all the interaction with the external world.

4. *The Community of Application-Specific Agents*. An agent is the basic computational entity of CooperA; it contains the knowledge to fulfil one or more tasks and knowledge about the external world (other agents, their location, and their capabilities).

On the basis of the CooperA kernel layer and with the help of CooperA system agents, a programmer can define agents, each one representing knowledge in various domains. A community of agents is thus created in which the user is involved through the user agent, and can thus interact with the community during problem solving. Message passing has been considered to be
the most suitable communication paradigm for the flexible and dynamic creation of distributed systems (Hewitt and Lieberman, 1984).

3.2 Description of a CooperA Agent

CooperA agents are active structures that communicate by means of messages. Each agent exists in its own unique environment: its context. In this context, an agent contains its own specific application knowledge, knowledge about other agents in the external world, and the necessary information to communicate with them.

An agent has a well-defined structure, expressed through a number of attributes enumerated in Table 14.1. The most relevant are described in detail below.

Considering communication, all messages are sent or received through two queues: the outgoing-messages queue (out-q) from which the outgoing messages are transmitted, and the incoming-messages queue (in-q) that receives and handles incoming messages. For each of these queues a function is preset in order to coordinate respective actions that have to be undertaken according to the message type. For instance when an agent receives a request, it questions accordingly its own local world of knowledge or activates its rule base using its inference engine, trying to fulfil the request. Then it returns any resulting information to the requesting agent.

In order to facilitate cooperation between agents, each agent contains the attributes my-skills and unsatisfied-goals. The former contains all the knowledge relative to its own capacities. The latter contains reference to all information that might be needed but is not found in its own world, and to the sources (if known a priori) which can provide this information.

The attribute Status of an agent describes the current status of the agent’s activity during execution. It can take one of the following values: new, inactive,

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context</td>
<td>Used for representing the agent’s address</td>
</tr>
<tr>
<td>Status</td>
<td>Agent status, changing at run time</td>
</tr>
<tr>
<td>Goals</td>
<td>Agent’s current goals, used during problem solving</td>
</tr>
<tr>
<td>Has tables</td>
<td>Refers to the data structures used for acquaintance modeling and communication</td>
</tr>
<tr>
<td>Current message</td>
<td>Message currently being processed</td>
</tr>
<tr>
<td>In-q</td>
<td>Input queue where incoming messages are put, awaiting the attention of the agent</td>
</tr>
<tr>
<td>Out-q</td>
<td>Output queue where outgoing messages are put, awaiting onward transmission by the kernel</td>
</tr>
<tr>
<td>My skills</td>
<td>List of goals that the agent can satisfy, with corresponding methods to achieve them</td>
</tr>
<tr>
<td>Unsatisfied goals</td>
<td>List of goals that cannot be reached using the local rule base and the knowledge of the agent</td>
</tr>
<tr>
<td>Activation method</td>
<td>Method necessary for the activation of the agent</td>
</tr>
</tbody>
</table>
running, and waiting. The value is assigned and handled by the appropriate kernel functions.

Each agent has its own knowledge of the external world (acquaintances) that permits cooperation. This information is contained in data structures local to the agent and created dynamically during the initialization of the system, when the community of agents is defined.

During the execution of a task, an agent can be in one of the following situations: (a) in need for some information, unknown to its own world, it has to decide to what agent to address a request for that information; (b) having calculated some results, it has to decide to what agent to send these.

To solve the first problem, a CooperA agent makes use of a particular structure: the Yellow-Pages (see Chapter 13). In this table, for each unsatisfied goal, i.e., information it cannot find or compute in its own world, the agent finds a list of agents in the community who might elaborate that information, or in other words, satisfy the goal in question.

To solve the second problem, the agent consults its Interested-in-Table, where it finds a list of agents in the community that could be interested in information it can calculate by itself.

3.3 Communication Mechanism

We have already mentioned the basic role messages play in CooperA: they support communication and hence contribute to agents cooperation. Here, we detail the structure of a message, and how the communication between agents, i.e., their exchange of messages, takes place.

The CooperA message is defined as a data structure containing a number of attributes. They include identity which is a unique symbol identifying the message; type indicating the kind of message, i.e., whether it is an information request, an answer, etc.; content which, in accordance with the message-type, refers to the actions it would like to invoke, or any other knowledge or information (e.g., it can contain the name of a goal, eventually followed by a list of parameters or it can contain an answer to a request, knowledge concerning actions, procedures, plans, allocation of tasks, etc.); sender which is the identity of the sender; receiver which contains a list of agents to which the message is addressed.

Figure 14.2 shows the communication process in CooperA. Messages are created by a sending agent through appropriate kernel functions and its local knowledge. The created message is put in the out-q of the sender and a kernel demon (out-q manager), monitoring the queue, is activated. This kernel service performs the message transmission by copying the message in the in-q of the receiver. Here, an appropriate demon (in-q manager) monitoring the in-q is activated and passes control to the receiving agent, if it is available and ready for execution. If the message is tagged as an answer to a previous request, the receiver matches the message with the pending request and reasons about the relevance of the supplied answer, which eventually can update the receiver’s
world of knowledge or begin a new query. If the message is tagged as a request to the receiving agent, the receiver tries to activate the appropriate method in order to satisfy the goal specified by the message. If the message is a request addressed to the user agent, it contains all the necessary information for specifying the dialogue with the user. The actual interaction with the user is managed by a special CooperA agent, called User Agent, which transmits the user's reply back to the agents interested in it.

The message receiver will be activated if its status is inactive. If not, the received messages remain in the incoming-messages queue, waiting for the agent to be ready to process them. In general, messages are FIFO processed, but in principle it is possible to introduce some kind of priority handling in the queues.

Agents react upon receiving a message according to the behavior expressed by their procedural activation methods, and the particular type of the received message. The active part of the agent, invoked by its activation method, acts on the local domain knowledge by making inferences on it or modifying it. This knowledge is only locally visible to the agent.

Since CooperA has been designed for interactive systems, there is also provision for communication with the human user. Moreover, there is the possibility of setting alternative communication strategies that can influence the problem-solving activity and the overall convergence to a solution (see Chapter 13).

A message interpretation mechanism is inserted in the transmission phase (see Figure 14.2, CP-interpretation mechanism), in order to provide global
understanding in the exchange of message. This is accomplished by means of
communication protocol.

9 Communication Protocol CP and the Dictionary. The exchange of
formation between all agents is done through the communication protocol
CP). The CP addresses the problem of defining a common representation for
interleaving parts of KBs. The sender agent creates an outgoing message
st in its local language, i.e., using symbols which have only a local meaning.
uring the message-sending operation, a translation into a global language
ssage takes place. This global language is part of the CP. The inverse
uslation, from global to local language, takes place at the receiving agent.

CooperA uses a data structure called Dictionary and some relevant kernel
ctions during the CP-interpretation phase. The Dictionary contains a direc-
ry of associations between concepts, locally defined and used by an agent,
d global terms, defined in the frame of the communication protocol.

The structure of the Dictionary for an agent _x_ is

\[
\{\text{Dictionary}_x
\]

\[
\ldots
\]

\[
\text{local-concept}_j : \text{CP-concept}_j \text{ address}_j
\]

\[
\ldots \}
\]

where local-concept\textsubscript{j} is an attribute referring to a local symbol, CP-concept\textsubscript{j} is
the globally known name of the local concept and address\textsubscript{j} represent the local
address where local-concept\textsubscript{j} is stored. If the same CP-concept is related to a
local-concept\textsubscript{x} of an other agent, the two concepts are considered, as referring
to the same symbol in a virtual global name-space of the community of the
distributed agents.

Figure 14.3 presents an example of the use of the CP-interpretation mecha-
nism together with the corresponding Dictionary definitions about an infor-

![Diagram](image)

**Figure 14.3** Example of the definition and use of the Dictionary.
mation (William) and a function (square-root). For instance, if AG1 sends an information about BILL to AG2, BILL is first translated into WILLIAM using AG1's dictionary and then WILLIAM converted to BILLY using AG2's dictionary. Analogously, if AG2 asks AG1 to compute the square root of a number, SQUARE-ROOT is first translated into SQUARE-ROOT and then into SQRT using respectively AG2's and AG1's dictionaries.

Each agent is provided with a Dictionary, in which the part of the agent's local knowledge which could be of interest to the rest of the community is mapped into a global representation. The problem that the CooperA programmer is faced with during the development of the Dictionary structure of an agent is not a trivial one. The coherence of the distributed problem-solving activity depends a great deal on an accurate semantic and lexical mapping of the concepts defined in the frame of the various KBs. Once this mapping has been completed the operation of developing structures like Dictionary is straightforward. So far, the mapping of the knowledge between agents is entirely left up to the CooperA programmers.

The correct definition of these dictionaries permits coherent communication between heterogeneous agents. This is a necessary (but not sufficient) step toward coherent behavior of the community of agents.

**Communication with the User.** CooperA is designed as an interactive system. The control over the interaction could be either with the user or with the agent community depending on the control strategy of the particular application. Thus, an agent could ask or notify the user about information, the user could ask for information from an agent, etc.

The user-to-agent interaction has been modeled according to the agent-to-agent communication mechanism. This has been achieved through a CooperA system agent called the user agent (UA), which represents the user within the community and handles all communication between the community of agents and the external world. The UA, always present in the system, is structured in the same way as all the other CooperA agents and communicates with them through the same message-passing mechanism. It can be seen as an ordinary agent with no specific domain knowledge, but with some special skills, e.g., it can handle I/O devices, graphics primitives, etc. However, the UA, representing the user (considered as a boundless domain of knowledge), is special as far as acquaintance modeling and request message structuring is concerned. A special type of message (user request) is introduced in order to manage different types of dialogues with the UA. The dialogue style selected is based on multilevel menus, and meta-information concerning the type of menu, prompt line, possible alternative answers, help, etc., is passed through user-request messages.

The knowledge necessary for building the user dialogue is usually owned by the agent making the request. This knowledge is passed to the user agent which takes care of the graphics, windowing system, etc., for visualizing the
request and managing the dialogue. After the interaction phase, the user agent will take care of forming the reply message according to the results of the dialogue with the user and passing it to the requesting agent or the agents interested in it. The user agent also manages messages supplying information to the user and answering its requests.

**Communication Strategies in CooperA.** The CooperA user is given the possibility of experimenting with different communication strategies for solving a problem. The community's behavior can be different, depending on the selections on a set of global switches. The switch *multiple-request* determines whether the set of alternative candidates for satisfying a particular goal could be exhausted before the requesting agent decides about a reply, eventually passing the request directly to the UA. The switch *broadcast-reply* determines how an answer is communicated to the external world of an agent, communicating the result exclusively to the agent who has made the request or to all agents that might be interested in that information.

Different CooperA agents may exchange information involving uncertainty in the form of certainty factors. Meta-knowledge about the uncertainty handling mechanism is essential for an agent in order to interpret a supplied reply. CooperA provides the switch *ask-for-of* which determines whether the uncertainty management mechanism is active.

### 3.4 Data Structures for Acquaintances Modeling

In CooperA, some data structures model locally the acquaintances of an agent (see Figure 14.4).

The *Yellow-Pages* structure contains a description of all the acquaintances of the agent who are in a position to satisfy goals that the agent is interested in. The metaphor used is that of a yellow-pages-style directory. This directory is created during the initialization of the agents in a dynamic way, taking into consideration the current participants of the group of agents and the needs of the particular agent. The yellow-pages schema of agent-x therefore have the form

\[
\{ \text{Yellow-Pages}_x, \\
\text{goal-1}: \text{agent-11 agent-12} \ldots \\
\text{goal-2}: \text{agent-21 agent-22} \ldots \\
\ldots \ldots \ldots \}
\]

The *Interested-in-Table* of an agent contains information about agents who are interested in its skills.

This model is also created dynamically during the phase of the configuration of a community of agents. The form of the interested-in-table for agent-x is
\{ Interested-in-table \}_x

skill-1: interested-agent-11 interested-agent-12 \ldots
skill-2: interested-agent-21 interested-agent-22 \ldots
\ldots
\}

Kernel functions take care of creating and updating these acquaintance structures.

3.5 The CooperA Interface

Problem solving in a distributed environment can be a very complex process to explain to the user of the system (see Chapter 21). The design of the CooperA user interface (CUI) has been developed following two requirements: to provide means for configuring in an easy and direct manner a community of agents and to visualize some of the aspects of the distributed problem-solving activity.

Functionally, the CUI provides the following services: the dynamic configuration of an agents world; the initialization and visualization of a problem-
solving activity representing graphically events such as the message exchange or the changing status of the agents; the navigation in the knowledge of the agents and during problem solving; the setting of the CooperA environment where a number of global parameters can be set to influence the problem-solving strategy.

For more details and the representation of the interface one should refer to Sommaruga et al. (1989b).

### 3.6 CooperA Results

To build CooperA we have borrowed some ideas of previous research activities in the area of DAI, but we have introduced some innovative features. As the most interesting characteristics of CooperA we mention that

CooperA supports the integration of distributed KBs and provides distributed problem-solving building tools.

CooperA has a strong experimental flavor. There is provision for global settings of the environment which permit alternative communication strategies, taking into consideration management of uncertainty, etc.

The concept of knowledge navigation during problem solving represents an useful debugging tool. This allows the user to follow more closely the effects of the agents' activities.

Attempts were made to handle the problem of interaction between the heterogeneous KBs contained in the different agents. The CP has been introduced, establishing a global knowledge representation for the community of agents permitting mutual understanding of agents which still can use their local formalisms.

The search for a solution of a particular subtask during the cooperative problem-solving activity can be performed through an iterative agent invocation mechanism, which allows for the opportunistic and fault-tolerant behavior of the community.

The direct manipulation user interface of CooperA allows visualization of the distributed problem-solving activity and direct access to the commands and options of the system.

The special user interface agent UA in the agents community is responsible for user interaction and for representing the user within the community. This agent has domain knowledge on I/O devices handling, graphical systems, etc. This I/O "expert" takes care of interaction with the user, taking responsibility from the rest of the community, who need only to interact with the UA in a uniform and familiar way.

Concerning the application, DChEM was developed in CooperA as a set of distributed KBs using and refining the existing monolithic ChEM KB. The advantages from doing this were twofold. First, modularization of the knowl-
edge into different KBs corresponding to separate areas of expertise was proficient and made it easier for the knowledge engineer and the expert to follow the problem-solving activity. Second, subsequent modifications to the KBs were easier to perform thanks to this modularization.

At run time DChEM exhibited the expected behavior, one expert agent using the skills of others if needed. The graphical workbench used for an easy configuration and initialization of the system has also proved useful for the visualization of the information flow and for debugging. The ability to configure a limited world of agents, and especially the possibility to set up a world of only one expert agent, together with the UA has proven to be useful for agent debugging.

4 FROM CooperA TO CooperA-II

Some limitations of the CooperA architecture also emerged and suggested starting points for further research:

The lack of concurrence and physical distribution of the agent activities. The main effort has been given over to the design of a distributed KB system, with simulated distribution in problem solving. The use of concurrent processes and computer-networks-based open system hardware platforms should make CooperA more efficient and capable of handling a new range of applications.

The limited flexibility in coordination and cooperation. CooperA provided only implicit cooperation because of the agent’s rigid control mechanism.

The limitation of the acquaintance modelling features of the CooperA language. The models were constructed a priori and could not be modified at run time. There was no possibility to reasoning about the skills of the acquaintances and there was no mechanism for any organizational structuring of the community.

Some mechanism for automating the development of the CP. The mapping of the elements of heterogeneous KBs ought to be introduced.

The extension of the UA domain knowledge. This could support dialogues adaptable to different user profiles and characteristics by adding elements of user modelling and so improving the interaction with the user.

These limitations led us to further research resulting in the evolution of the CooperA architecture. In particular, cooperation and control emerged to be the two most pressing issues in the development of cooperative agents (see Chapter 1). As a consequence, the approach used during this phase, was to study first existing theories and experiences in the field of small group interactions and cooperation (Hewstone et al., 1988; Argyle, 1991). In particular, this effort
focused on the identification of coordination rules of the members of the group. This resulted in the creation of a knowledge base containing heuristic rules which can drive the cooperation of a group of agents. In addition, the process of defining a generic architecture for cooperative agents has been investigated, also on the basis of previous experiences. The complete analysis is described in detail in Sommaruga (1993). It has been observed that in general this process affects a number of steps. We identify four levels and meta-levels, starting from the lowest level of the real world (Real World level), going up to the computational (expert) systems (Application level), to the competence abstraction (Abstraction level), and finishes at the meta-level of cooperation (Cooperation level). The Real World level represents a real-life problem, while the Application level represents the computer programs which deal with the problem. The Abstraction level represents the definition of the agents' skills, needs, etc., and finally the Cooperation level represents where the control of the agents resides. The composition of each level of the agent on top of others allows a modular integration of the different levels of functionalities of an agent.

Earlier research has explored many of the issues at lower levels, as mentioned in Gasser (1992). However, a computational understanding of cooperative issues was relatively underresearched. The previous work could be criticized. The major criticism consists of the fact that the attempts at modeling architectures resulted in ad hoc architectures without explicit and distinct control levels for agents. The agent lacked flexibility and generality.

This stratified view offers some advantages, such as a modular and flexible architecture, and independence of the control of the agent from the application domain. On this basis, a new model for a cooperative agent is defined in CooperA-II, and a number of axioms are formalized in a Cooperative Architecture Theory (CATH) according to the level functionalities (Sommaruga, 1993; Sommaruga and Shadbolt, 1994). CATH is a formalization of fundamental premises for defining and obtaining a cooperative architecture of autonomous agents, supported by a language for defining agents and determining their behavior. A number of axioms are dictated by very general constraints on computational (intelligent) systems and others are influenced by more specific DAI and cooperative requirements and refer specifically to our cooperative architecture. They concern issues which deal, for example, with the agent structure, communality requirement for skills and needs, compositionality of local microbehaviors, compositionality of global behaviors (see Section 5.4), group dynamics, communication requirement, and goal induction (see Chapters 2 and 9).

In the next section we are only able to present an overview of the CooperA-II architecture.

1 An example of which can be found in Newell (1982). For instance, Newell's principle of rationality states that "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action."
5 CooperA-II

The architectural framework CooperA-II for the cooperation of autonomous agents has been implemented to reflect the principles of the theory and to provide an evaluative testbed. In particular, the problem of coordinating agents’ activities in order to achieve cooperative profitable behavior is addressed in CooperA-II. The introduction of a forward-reasoning control mechanism in each agent allows it to plan and coordinate tasks according to heuristic knowledge about cooperation. This has been specified as a cooperative heuristics knowledge base (CH-KB), which pragmatically expresses a number of different possible cooperative behaviors for an agent in the form of rules.

In the following description we focus on a specific overview of the cooperation of agents in the CooperA-II architecture on the basis of the complete agent model introduced in Sommaruga (1993) and Sommaruga and Shadbolt (1994). However, a number of structures and mechanisms which specialize the agent’s cooperative attitudes need to be introduced first.

The agenda is a new attribute of the agent model. It represents a knowledge structure containing items, called acts, which represent potential actions of the agent. Each agent is autonomously controlled by means of a control loop (control cycle, see below) which uses the agenda to govern the agent’s behavior. In particular, acts in the agenda are exploited by the cooperative heuristics rules in order to select the best action to be accomplished.

Only acts of a predefined type and format may be created and put on the agenda. The types of acts, currently considered in CooperA-II, are

DO act, in order to operate on execution of actions
REQUEST act, for requesting information or execution of actions to other agents
REQUESTED-TODO act, in order to adopt an external (act of) request
SUPPLY-INFO act, in order to provide information to others
INFORM act, for being informed about another agent’s information

These five types of acts are inspired by Cohen and Pernault’s operators (Cohen and Pernault, 1979). The act types are the “conversational” operators which allow agents to interact. An act-based semantics was used and integrated with the agenda mechanism.

Another particular characteristic of CooperA-II is the control cycle of an agent, which begins with the creation of the agent and permanently remains active. It consists of a loop, which is divided into three steps:

Step 1. Situation verifier, which evaluates firstly the current status of knowledge and activities of the agent. If satisfied then go to Step 2; otherwise go to step 1 and the agent remains idle.

Step 2. Select best action, which selects the next action of the agent from the
agenda according to the behaviors expressed in the CH-KB. If successful then go to step 3; otherwise go to step 1.

Step 3. Execute best action selected, which possibly produces an output in the form of a message or an action execution. Go to Step 1.

Selecting the best action is a distinctive phase in CooperA-II, and it is the aspect we will turn to later in this chapter.

Considering loci of control in CooperA-II, a number of actions can be asynchronously performed by an agent:

Executing a control cycle
Sending a message
Receiving a message
Executing one or more processes to accomplish tasks

The asynchronous nature of the actions of an agent is an important aspect of a cooperative group of agents. In fact, this could mean, for instance, that sending information (a message) may occur at the same time the agent is executing a task, receiving a message, or deciding what to do next. This aspect is a positive characteristic for cooperation which provides more autonomy in the control of the agent. Moreover, a more open control flow of the agent’s actions is allowed by the fact that independent actions are not sequentialized. This extends the limited control of a CooperA agent which was strictly sequential. In CooperA-II, asynchronous actions are obtained by means of distinct processes which manage each type of activity.

In summary, distinctively and differently from MACE’s engines, CooperA’s start-methods, Actor’s script, and other architectures, in CooperA-II an agent behaves entirely in accordance with its control mechanism and its cooperative heuristics.

The communication between the agents is provided by a message-passing mechanism integrated into the agent model. In particular, CooperA’s in-q and out-q have been adopted, and other features have been extended in order to guarantee a safe communication (see Section 3.3 and Sommaruga, 1993).

The CooperA-II platform interface is presented in Figure 14.5. A number of facilities, such as tracing of executions, menus for debugging and system commands, continuous updating of the status of agents’ tasks, and others, are provided in order to facilitate the agent developer in creating, testing, and running a cooperative application.

A number of attributes of the agent model of CooperA are maintained in CooperA-II, while others are added and extended. Among them, it is interesting to note the dependency between skill/need definition and cooperation. In fact,

Here, “at the same time” could mean in parallel or concurrently, depending on the hardware platform used for the implementation.
Figure 14.5 The CooperA-II platform with DChEM2 application.
by means of interleaving skills and needs of different agents, cooperation in a
group is influenced.

The dynamic composition of a group of agents is another interesting feature
which CooperA-II inherits from CooperA. The structure of an agent group can
be dynamically changed by adding new computational individuals and hence
capabilities (skills). Modules, in the form of agents, can be tested individually
before their integration in a group. This allows extreme modularity of the
system. Moreover, agents of different granularity may be developed and inte-
grated in CooperA-II. An agent description language is provided in order to
support the definition of agents of various complexity ranging from large grain
to small grain.

Two applications in different real domains were developed in CooperA-II
using the cooperative heuristics approach: the distributed chemical emergencies
manager 2 (DChEM2) and Libra.

The DChEM2, a revised version of the DChEM system, is a distributed
cooperative agents system prototype which simulates a particular aspect of the
management of a chemical accident. The aspect deals with the evaluation of
the type of the chemical substance involved in an electrical hazard. DChEM2
consists of a group of three agents which interact with a user in order to
determine the nature of a substance involved in a possible chemical accident.
We have three agents, each one performing a distinct role. The site-expert (SE)
agent, which can estimate the initial substance according to data about the site.
The accident-manager (AM) agent, which accomplishes the task of determining
the initial substance according to its expertise in managing chemical accidents.
Finally, the OBSERVER agent, which allows the user to interact with the
system and provides all the input data by means of requests to the user. The
task of finding out what kind of substance is involved in a chemical accident
is inserted in a more complex system for monitoring such an accident. A request
by the user for the computation of the initial substance to one of the agent,
starts a sequence of actions of dividing goals into subgoals, of requesting,
informing, and computing all the necessary subtasks in order to evaluate the
type of the substance. The order of the actions in this sequence is completely
determined by the application of the CHs. In Figure 14.5 a session of the
DChEM2 application, with the agents previously described, is presented.

The Libra application consists of a group of four cooperating agents which
simulate a librarian helping a user to find references about subjects, authors,
etc. During a search session in Libra, a number of interleaving tasks are per-
formed by the group of agents. First, the user formulates his or her requests
through the user agent. Then these are sent to the other agents. Two kinds of
resulting situations are possible after computation by the group of agents. Either
an acceptable retrieval strategy is found and applied, or no acceptable retrieval
strategy is available. In the first case the user will be provided with either the
desired bibliographical items or no acceptable result. In the second case various
suggested elaborations are made to the user. The activities each agent performs
are determined by the application of sets of CHs, in particular the suggestions
are determined by the pro social behavior (see pro-active m-behavior in Section 5.4).

5.1 Cooperation

The coordination of actions of a group of agents is an important issue in designing agent architecture (see Chapter 6). Joint profit should be the main goal of a group. Questions concerning what to do, when to do it and how to achieve it have to be answered. This problem has been only partially addressed in CooperA. It has been successively addressed by means of the CH model which has been integrated in the CooperA-II architecture. A detailed description of this architecture can be found in Sommaruga (1993), where a thorough description of the CH rules is also contained.

Here, we are going to focus on the influence such cooperative rules may have on the cooperative behavior of an agent. First, a general introduction to the cooperative heuristics model is given. Second, the definition of agent behaviors is used to group the cooperative heuristics into subsets according to pragmatic criteria of classification of the rules. Finally, the possibility of the composition of basic behaviors in order to create global or emergent behavior for an agent is discussed.

5.2 Cooperative Heuristics

In the agent model of CooperA-II we organized an agent architecture according to the stratified functional levels presented above. Among other levels, we have justified the introduction of "cooperative mechanisms" at the cooperation level, (see Chapters 12 and 13), in order to enhance meta-level control and coordination. Our attention has been directed to the definition of generic independent cooperative mechanisms. We have already suggested the advantages offered by designing control structures as independent as possible from the other levels of application and communication. The results are improved flexibility and reusability of the architecture.

After having defined the basic architecture and functionalities we need to complete our model, adding inferential meta-level power to each agent. Intelligence has to be encoded in the controller of the agent. In this phase, the previously described control structure and procedures are completed by filling a knowledge base of heuristics which are exploited in the control mechanism of the agent. To this aim a number of heuristic rules relating to the behavior of an agent are defined. These cooperative heuristics derived, in part, from sociology and social psychology (Hewstone et al., 1988; Argyle, 1991) and from previous DAI experience and experiments (CooperA and Wesson et al., 1988). The CHs generally relate to types of act, some of them to communication, and a number of them deal with planning and others with control.

Cooperative heuristics are represented as production rules inside each agent.
They have a left-hand side which describes all the conditions which should be verified in order to fire the rule. The conditions match facts of the agent which reflect mainly the status of knowledge of the agent itself and of its acquaintances. The actions in the right-hand side of rules can both modify the status of control knowledge of the agent and initiate the execution of an action.

Each agent has a local inference engine integrated with its local working memory. Inferences are derived in a forward-chaining fashion by facts declared in the working memory. The control cycle performed inside each agent controls the activity and manages the decision phase (see step 2). The decision phase is the one in which the agent decides what will be the next action to be executed. The choice is entirely determined by the firing of a cooperative heuristic on the basis of the existing agent's situation. In fact, the knowledge corresponding to a situation of the agent (agenda, acquaintances, status of knowledge of skills and needs, etc.) is taken into account in order to make a decision about the best action.

Currently, in CooperA-II the decision making algorithm is implemented in a data-driven way and the cooperative heuristics are implemented as OPS5 rules (Brownston et al., 1985). An OPS5 reasoning mechanism is integrated in step 2 of the controller of each agent. This allows us to define independent sets of rules and to apply them in the control system. Working memories are distinct and inference engines run independently. Therefore each agent reasons in an autonomous way and all interactions are the result of the application of some heuristics.

5.3 Classification of the Cooperative Heuristics

In the context of this description of the evolution of the CooperA platform, it is not relevant to see in detail the definition of each CH. A complete analysis of the semantics of the CHs has been carried out and described in Sommaruga (1993). It is more interesting to consider here an abstraction of the cooperative rules according to their purposes.

The cooperative heuristics so far defined are classified according to some general criteria. A number of properties, such as computational cost, continuity, etc., have been used to create a taxonomy of the CHs rules using a repertory grid method. Groups of similar CHs emerged by comparing the values of the properties. An emerging grouping of most similar cooperative heuristics consider groups and supergroups of heuristics according to their functionalities, as presented in Figure 14.6.

The main groups include

A · Execute a Task. This group contains CHs regarding the execution of a skill, in particular the CHs which deal with DO acts.
B · Acquire Info. This group contains CHs regarding the acquisition of information which deal with INFORM acts.
C · Provide Info. It is a group of CHs which concern actions of supplying information to other agents and therefore deal with SUPPLY-INFO acts.

D · Request Info. It contains CHs regarding the request for information from other agents and enumerates all the CHs about REQUEST acts.

E · Avoid Repetition. This group associates CHs which help the agent to avoid repetition of not advantageous actions.

F · Identical Act Reduction. This set associates CHs which help the agent to avoid repetition of the same action.

G · Decomposition PS Strategy. This group deals with the accomplishment of a complex skill by means of the decomposition of a goal (namely a skill) into subgoals and their aggregation through a decomposition-synthesis process.

Figure 14.6 Cooperative heuristics classification.
H. *Proactive Strategy*. It deals with the accomplishment of a skill by means of the recognition of a suitable situation in the agent which could allow a successful computation of the skill. Following this strategy the agent under certain conditions can spontaneously activate the execution of an action.

On the basis of the analysis of the groups defined we can compose similar groups into supergroups. We can distinguish the supergroup (AB), about knowing an information, the supergroup (CD), concerning interaction of the agent, the supergroup (EF), dealing with internal transformation, and finally the supergroup (GH), containing specific problem-solving strategies.

It is interesting to note that some groups of CHs express Gricean maxims (Grice, 1975). For example, quantity of information is expressed in the groups provide info (C) and request info (D); quality emerges in most of the groups; relation is considered in avoid repetition (E), identical act reduction (F); and proactive strategy (H); manner is expressed in provide info (C) and request info (D).

### 5.4 Agent Behaviors and Cooperative Heuristics

Another grouping of the CHs may be created according to the global effects a subset of CHs can produce inside an agent. The global effect a subset of CHs produces in an agent is called the agent micro behavior (*m*-behavior) (Chapters 9, 11, and 20). Various micro behaviors can be composed generating a macro behavior of an agent, which is usually considered as the emergent behavior of the agent. The interaction and coordination of the behaviors of different agents allows the integration of individuals in the form of a cooperative group of agents. In this section we describe a number of *m*-behaviors which have been defined in CooperA-II. A summary is presented in Table 14.2. Finally, examples of their structure are presented.

*Micro Behaviors*. First, basic functional activities of an agent have to be mapped into default CHs which need to be always present in the agent. They deal with managing simple instances of act types like REQUEST, SUPPLY-INFO, DO, and INFORM. They also manage forced actions (actions which may have to be recomputed). They are also used to avoid duplication of identical acts. Such heuristics are referred to as default m-behavior.

Second, some CHs reflect the need for an agent not to repeat the execution of actions already accomplished. These mainly regard skills which have been satisfied by the agent when there is no need to evaluate them again. In particular, they consider acts of type REQUEST and DO when the related infor-

---

3Gricean maxims can be summarized by the four conversational categories summarized in the "cooperative principle." They deal with quantity, quality, relation, and manner.
TABLE 14.2 Summary of m-Behaviors and Their Composing Cooperative Heuristics

<table>
<thead>
<tr>
<th>m-Behavior</th>
<th>Cooperative Heuristics Subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repetitive</td>
<td>One-request-act-RE, one-do-act-RE, Request-act-2-RE</td>
</tr>
<tr>
<td>Reactive</td>
<td>Divide-do-act, reduce-or-trees, Supply-requested-info</td>
</tr>
<tr>
<td>Proactive</td>
<td>Do-an-executable-skill</td>
</tr>
<tr>
<td>Resend-request</td>
<td>Resend-request</td>
</tr>
<tr>
<td>Uncond-goal-adoptive</td>
<td>Adopt-request</td>
</tr>
<tr>
<td>Selfish-goal-adoptive</td>
<td>Selfish-adopt-request-1, Selfish-adopt-request-2</td>
</tr>
</tbody>
</table>

Information is not yet known by the agent. Moreover, they ignore requests to other agents for information which is already available to the agent. The corresponding acts are in this case removed from the agenda. They are here referred to as nonrepetitive m-behavior.

Third, some CHs reflect the need for an agent to repeat the execution of actions already accomplished. These regard, for instance, skills which have been already attained by the agent but need to be executed again on different values or arguments. As with the previous group they consider acts of type REQUEST and DO, but they ignore the status of the related information (known or not). They are referred to as repetitive m-behavior.

Fourth, the group called reactive m-behavior deal with the solution to complex problem by using a decomposition problem-solving strategy. Some CHs manage to split a complex skill of an agent into subactions. Moreover, the reactive m-behavior is also characterized by allowing an agent to answer information which has been previously requested by another agent. For instance, this could be the case where an answer (corresponding to the completion of a skill) needs to be sent back to the requester.

Proactive m-behavior deals with the solution to complex problems by spontaneously activating tasks. The execution of a skill is started spontaneously when the situation of the agent allows it. This behavior reflects the fact that activating a skill whenever possible may generate a generous agent which always tries to do what it can in order to solve a problem.

An agent may need to send again a request for information which has not been answered after a certain time delay. A resend-request m-behavior is therefore defined.

Finally, two other m-behaviors regard the adoption of goals of an agent. This problem is associated with managing REQUESTED-TODO acts. It can be approached by two types of solutions: unconditional adoption of goals and
TABLE 14.2 Summary of m-Behaviors and Their Composing Cooperative Heuristics

<table>
<thead>
<tr>
<th>m-Behavior</th>
<th>Cooperative Heuristics Subset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>One-inform-act, one-supply-info-act, identical-do-acts,</td>
</tr>
<tr>
<td></td>
<td>identical-request-act, just-informed-request, inform-act-1,</td>
</tr>
<tr>
<td></td>
<td>inform-act-2, supply-info-act-1, supply-info-act-2, inform-act- if-waited,</td>
</tr>
<tr>
<td></td>
<td>Supply-act-if-not-waited, force-a-do-act, force-a-request-act</td>
</tr>
<tr>
<td>Nonrepetitive</td>
<td>One-request-act, only-one-do-act, Only-one-exec-do-act,</td>
</tr>
<tr>
<td></td>
<td>one-do-act, Already-requested-act, already-known-act,</td>
</tr>
<tr>
<td></td>
<td>Already-done-exec-act, already-sent-to, request-Act-1, request-act-2</td>
</tr>
<tr>
<td>Repetitive</td>
<td>One-request-act-RE, one-do-act-RE, Request-act-2-RE</td>
</tr>
<tr>
<td>Reactive</td>
<td>Divide-do-act, reduce-or-trees, Supply-requested-info</td>
</tr>
<tr>
<td>Proactive</td>
<td>Do-an-executable-skill</td>
</tr>
<tr>
<td>Resend-request</td>
<td>Resend-request</td>
</tr>
<tr>
<td>Uncond-goal-adoption</td>
<td>Adopt-request</td>
</tr>
<tr>
<td>Selfish-goal-adoption</td>
<td>Selfish-adopt-request-1, Selfish-adopt-request-2</td>
</tr>
</tbody>
</table>

m-Behavior is not yet known by the agent. Moreover, they ignore requests to other agents for information which is already available to the agent. The corresponding acts are in this case removed from the agenda. They are here referred to as nonrepetitive m-behavior.

Third, some CHs reflect the need for an agent to repeat the execution of actions already accomplished. These regard, for instance, skills which have been already attained by the agent but need to be executed again on different values or arguments. As with the previous group they consider acts of type REQUEST and DO, but they ignore the status of the related information (known or not). They are referred to as repetitive m-behavior.

Fourth, the group called reactive m-behavior deal with the solution to complex problem by using a decomposition problem-solving strategy. Some CHs manage to split a complex skill of an agent into subactions. Moreover, the reactive m-behavior is also characterized by allowing an agent to-answer information which has been previously requested by another agent. For instance, this could be the case where an answer (corresponding to the completion of a skill) needs to be sent back to the requester.

Proactive m-behavior deals with the solution to complex problems by spontaneously activating tasks. The execution of a skill is started spontaneously when the situation of the agent allows it. This behavior reflects the fact that activating a skill whenever possible may generate a generous agent which always tries to do what it can in order to solve a problem.

An agent may need to send again a request for information which has not been answered after a certain time delay. A resend-request m-behavior is therefore defined.

Finally, two other m-behaviors regard the adoption of goals of an agent. This problem is associated with managing REQUESTED-TODO acts. It can be approached by two types of solutions: unconditional adoption of goals and
conditional adoption of goals. An uncond-goal-adoption m-behavior has been defined in order to deal with the direct transformation of a REQUESTED-TODO act into a DO act inside the agenda of an agent. This can be considered as a direct adoption of a goal by the agent. In addition, a selfish-goal-adoption m-behavior has been defined in order to deal with the transformation of a REQUESTED-TODO act into a DO act only under certain conditions. In this case a selfish agent is assumed to do the transformation only when there are no other acts of different type. Thus, the selfish agent adopts external requests for a local skill only when it has finished its own activities. We have to note that the approach of conditional adoption of goals is open to a variety of different conditional cases. Only one of them is mentioned above as an example.

Similarly, other m-behaviors which give an agent the possibility of behaving in various ways and coping with different situations could be defined.

In conclusion, some examples of the composition of the m-behaviors so far defined are presented.

During the initialization phase of an agent (i.e., its creation), a behavior can be attributed to an agent by means of a declaration. This will allow the agent to be initialized with some cooperative heuristics sets which will later influence its activity.

For example, the declaration

\[(ag-behave agent_x 're-active 'non-repetitive 'uncond-goal-adoption)\]

generates an agent which will behave according to all the CHs corresponding to the reactive, nonrepetitive and uncond-goal-adoption m-behaviors listed in Table 14.2. In particular, the agent will deal with the solution to a complex problem by using a decomposition problem-solving strategy, it will avoid repeat actions like requesting or computing already known information, it will adopt unconditionally external requests for attaining a local skill.

A further example is

\[(ag-behave agent_y 'pro-active 'repetitive).\]

This declaration generates an agent which will solve complex problems by spontaneously activating tasks, where the execution of a skill is started spontaneously when the situation allows it, and will allow it to repeat the execution of actions.

Again, by declaring

\[(ag-behave ag_z 're-active 'repetitive 're-send-request 'uncond-goal-adoption)\]

the agent being defined will deal with the solution to a complex problem by using a decomposition problem-solving strategy, it will allow repeated actions like requesting or recomputing known information, it can send a request for
information which has not been answered after a certain time delay and it will adopt unconditionally external requests for attaining a local skill.

5.5 Examples of Cooperative Heuristics and Their Application

In this section we provide first a description of some cooperative heuristics which are employed in CooperA-II, and successively an example of their application to a real problem-solving session of DChEM2.

Here, some significant CHs have been selected and described in detail just to give an idea of how they can contribute to a particular kind of m-behavior. Each cooperative heuristic is described with a short explanation of its purpose, presenting the effect of the rule triggering. In addition, the conditions for the rule being triggered are given. Moreover, an important property of the cooperative heuristics which affects the continuity of step 2 of the control cycle is underlined for each heuristic. This property distinguishes if a rule is final or intermediate. A final rule generates the selection of the best action and completes the decision phase of step 2 (see Section 5). An intermediate rule generates modification to the situation of the agent and allows the decision making to proceed until explicitly halted.

The following cooperative heuristics are presented with respect to the groups of m-behaviors in Table 14.2.

Default

identical-request-acts. This rule deletes duplicates of REQUEST acts from the agenda. If two REQUEST acts for the same information are on the agenda, then the most recent REQUEST act, which constitutes a duplicate of the previous one, is removed from the agenda. This is an intermediate heuristic.

just-informed-request. A request for any information that has just arrived to an agent is ignored. If a REQUEST act and an INFORM act about the same information exist on the agenda, then the REQUEST act is removed from the agenda and the INFORM act is immediately executed. This is a final heuristic.

Nonrepetitive

already-requested-act. This rule ignores a request for an information already known by the agent. If the agenda contains a REQUEST act for an information already known by the agent, and the REQUEST act is not forced to be done, then the request is ignored by removing the REQUEST act from the agenda. This is an intermediate heuristic.

already-known-act. This rule avoids doing a skill or requesting information already known by the agent. If the agenda contains a DO act of an already known skill or a REQUEST for already known information, and the act is not
forced to be done, then it is ignored removing the DO or the REQUEST act from the agenda. This is an intermediate heuristic.

**Reactive**

**divide-do-act.** This rule creates DO or REQUEST acts which are subgoals of a skill. If the agenda contains a DO act about a complex goal which is not yet known and is divisible into subgoals, then the DO act is split into subgoals. These subgoals could be other DO acts, if they refer to (internal) skills of the agent, or REQUEST acts if they refer to (external) needs of the agent. Therefore, other DO or REQUEST acts can possibly appear onto the agenda. This is an intermediate heuristic.

**Example of the Application of the CHs.** This description completes the explanation of the CHs approach by providing an example of their use in a real application.

We assume that the computation of the initial substance type involved in an accident is the global goal which the DChEM2 group of agents has to reach. We consider that a typical real accidental situation is represented by a set of data values available to the user. They constitute a description of the accident and allow for its simulation. These values may be provided through the OBSERVER agent in any temporal or physical sequence.

The problem-solving method adopts an opportunistic strategy, as follows:

- global goal request →
- starting skills computation →
- combinations of concurrent actions
  - user providing data,
  - agents requests for data to the user according to the situation and
  - skills computations
- final result.

In this example any input could be initially given to the OBSERVER agent, the initial substance task was directly started, some of the input data values were provided randomly and spontaneously at run time by the user while AM and SE were computing their own skills concurrently.

Details of the main steps are summarized in Figure 14.7, where an overview of a solution to the DChEM2 problem through time, allows the application of the CHs in each problem-solving step to be shown. This is represented in square brackets by the name of the agent followed by the numbers of the CHs used in such step, such as [OBS: CHs 1]. These numbers are listed in the sequence of CHs presented below.
Figure 14.7 Overview of a solution to DChEM2 problem.
Sequences of CHs Used by Agents in Figure 14.8. We summarize below for each agent an ordered list of all the CHs used in the sample execution presented in Figure 14.8. Each CH is paired to the most relevant triggering information. You should use these lists as a reference to the description presented in the Figure 14.8. The names of the CHs are sufficiently intuitive to be self-explanatory about their effects.

Sequence of CHs Used in Agent SE

1. (ADOPT-REQUEST INITIAL-SUBSTANCE)
2. (DIVIDE-DO-ACT INITIAL-SUBSTANCE)
3. (REQUEST-ACT-1 IS-OBS)
4. (DIVIDE-DO-ACT IS-SITE)
5. (DIVIDE-DO-ACT DOCS)
6. (REDUCE-OR-TREES (IS-CLASS DOCS-OIL-SPEC))
7. (END-RM-OR-TREES DOCS)
8. (REQUEST-ACT-2 DOCS-OIL-SPEC)
9. (REQUEST-ACT-1 IS-CLASS)
10. (DIVIDE-DO-ACT CONTAINER)
11. (REDUCE-OR-TREES (THE-CONTAINER-TYPE))
12. (END-RM-OR-TREES CONTAINER)
13. (REQUEST-ACT-1 THE-CONTAINER-TYPE)
14. (INFORM-ACT-1 CARDS-DATA)
15. (ONLY-ONE-EXEC-DO-ACT CONTAINER)
16. (ADOPT-REQUEST CONTAINER)
17. (ALREADY-KNOWN-ACT CONTAINER)
18. (SUPPLY-REQUESTED-INFO CONTAINER)
19. (SUPPLY-INFO-ACT-1 CONTAINER)
20. (ADOPT-REQUEST IS-SITE)
21. (IDENTICAL-DO-ACTS IS-SITE)
22. (INFORM-ACT-1 CARDS-DATA)
23. (ONLY-ONE-EXEC-DO-ACT DOCS)
24. (ONLY-ONE-EXEC-DO-ACT IS-SITE)
25. (SUPPLY-REQUESTED-INFO IS-SITE)
26. (SUPPLY-INFO-ACT-1 IS-SITE)
27. (INFORM-ACT-1 IS-OBS)
28. (ONLY-ONE-DO-ACT INITIAL-SUBSTANCE)
29. (SUPPLY-REQUESTED-INFO INITIAL-SUBSTANCE)
30. (ONE-SUPPLY-INFO-ACT INITIAL-SUBSTANCE)
Figure 14.8  Agents and CHs application through time.
Sequence of CHs Used in Agent AM

1. (ADOPT-REQUEST IS-OBS)
2. (DIVIDE-DO-ACT IS-OBS)
3. (DIVIDE-DO-ACT OBS)
4. (REDUCE-OR-TREES (THE-SUBST-COLOUR THE-SUBST-STATE))
5. (END-RM-OR-TREES OBS)
6. (REQUEST-ACT-2 THE-SUBST-STATE)
7. (REQUEST-ACT-2 THE-SUBST-COLOUR)
8. (REQUEST-ACT-1 CONTAINER)
9. (DIVIDE-DO-ACT ANALYSIS)
10. (REDUCE-OR-TREES (TEST-FLAME))
11. (END-RM-OR-TREES ANALYSIS)
12. (REQUEST-ACT-1 TEST-FLAME)
13. (ADOPT-REQUEST INITIAL-SUBSTANCE)
14. (DIVIDE-DO-ACT INITIAL-SUBSTANCE)
15. (REQUEST-ACT-1 IS-SITE)
16. (INFORM-ACT-1 CARDS-DATA)
17. (INFORM-ACT-1 CARDS-DATA)
18. (INFORM-ACT-1 CONTAINER)
19. (INFORM-ACT-1 CARDS-DATA)
20. (INFORM-ACT-1 IS-SITE)
21. (INFORM-ACT-1 CARDS-DATA)
22. (ONE-DO-ACT ANALYSIS)
23. (ONLY-ONE-EXEC-DO-ACT OBS)
24. (ONLY-ONE-EXEC-DO-ACT IS-OBS)
25. (ONLY-ONE-EXEC-DO-ACT INITIAL-SUBSTANCE)
26. (SUPPLY-REQUESTED-INFO INITIAL-SUBSTANCE)
27. (SUPPLY-INFO-ACT-1 INITIAL-SUBSTANCE)
28. (ALREADY-KNOWN-ACT IS-OBS)
29. (SUPPLY-REQUESTED-INFO IS-OBS)
30. (ONE-SUPPLY-INFO-ACT IS-OBS)

Sequence of CHs Used in Agent OBSERVER

1. (REQUEST-ACT-2 INITIAL-SUBSTANCE)
2. (SUPPLY-INFO-ACT-2 CARDS-DATA)
3. (SUPPLY-INFO-ACT-2 CARDS-DATA)
4. (ADOPT-REQUEST CLASS)
5. (SUPPLY-INFO-ACT-1 CARDS-DATA)
6. (ADOPT-REQUEST OIL-SPECIF-PHRASE)
7. (ONE-DO-ACT CLASS)
8. (ONLY-ONE-DO-ACT OIL-SPECIF-PHRASE)
9. (ADOPT-REQUEST CONTAINER-TYPE)
10. (ALREADY-SENT-TO CONTAINER-TYPE)
11. (SUPPLY-INFO-ACT-1 CARDS-DATA)
12. (ADOPT-REQUEST PHYSICAL-STATE)
13. (SUPPLY-INFO-ACT-1 CARDS-DATA)
14. (ADOPT-REQUEST FLAME-TEST)
15. (ONE-DO-ACT PHYSICAL-STATE)
16. (ALREADY-SENT-TO OIL-SPECIF-PHRASE)
17. (ALREADY-SENT-TO CLASS)
18. (ADOPT-REQUEST COLOUR)
19. (ALREADY-SENT-TO COLOUR)
20. (ONE-DO-ACT FLAME-TEST)
21. (ALREADY-SENT-TO PHYSICAL-STATE)
22. (ALREADY-SENT-TO FLAME-TEST)
23. (ONE-SUPPLY-INFO-ACT CARDS-DATA)
24. (ONE-INFORM-ACT INITIAL-SUBSTANCE)
25. (ONE-INFORM-ACT INITIAL-SUBSTANCE)

The example shows the cooperative interactions which occur during the problem-solving process by applying the set of CHs. We can note that the major cooperative interaction happens during the first phases of the problem-solving activities as expressed in Figure 14.8 by the great number of crossing arrows from one agent to others.

It is possible to observe that the agents were aware of the possibility of receiving help from the others. For instance, AM and SE agents requested some of their subtasks of the initial substance skill to SE or AM, respectively, or to the OBSERVER/user. Moreover, some requests were not sent because they were already known or already dispatched, or else were duplications.

In other words, the behaviors of the group of agents reflected the compositions of the m-behaviors assigned to each agent, confirming the cooperative features explicitly designed in the CHs.

The agents adapt their problem-solving strategies to the particular conditions according to the possibilities encapsulated in the set of CHs behaviors. The simplest instance of this adaptation is the case of a need which is requested from the user by an agent, but the user spontaneously provides it in advance to this agent. Thus, the request is no longer necessary and can be removed from the agenda of the agent, saving time and resources. For instance, see CHs number 9 and 10 of agent observer in Figure 14.8.
5.6 CooperA-II Results

An evaluation of our approach, the CooperA-II architecture, and the use of CH-KB was carried out together with a comparative analysis with other well-known DAI architectures (Sommaruga, 1993) and on the basis of the two applications developed, Libra and DChEM2. This analysis highlighted that some ideas of previous CooperA research have been borrowed in CooperA-II, but also that some innovative features have been introduced. We are going to mention the main results.

Good flexible applicability to general domains and environments of the CooperA-II architecture was shown. The resulting model represents a modular and decomposable approach to cooperating autonomous agents, based on cooperative behaviors. In particular, the agent model is sufficiently general to be adopted in various domains and applications. In addition, the use of abstractions of the agents' goals (abstraction level) and the possibility of reasoning about these goals at the meta-level of cooperation provided a more effective cooperation. CooperA-II ranked significantly better than other architectures in providing facilities for agents to act coherently using problem-solving strategies, in using sophisticated group control strategies, in providing effective means to coordinate, organize, control, and reason about control.

In CooperA-II the developer can control the interpretation of events and how an agent should act in certain situations by means of the behaviors expressed in the cooperative heuristics. This explicit heuristic knowledge about cooperation provides agents in the system with a number of cooperative problem-solving strategies in different domains and under different situations within the same domain.

Thanks to the explicit representation of the CH-KB in the form of rules which can express agent m-behaviors, this knowledge could be easily extended or changed, modifying at the same time the global behavior of the agent.

6 CONCLUSIONS

The reported research describes the evolution of the CooperA architecture. In particular, it has investigated cooperative and coordinated problem solving by a set of coarse-grain knowledge-based agents. The agents involved were full-scale expert systems, each of them with distinct area of expertise and problem-solving capabilities. Integration of such knowledge-based components in a multiagent cooperative environment has proved a particularly hard task fulfilled by this research.

During the first phase of experimentation, emphasis was provided in practical issues of multiple agents integration. Thus, interagent communication protocols have been defined, and communication mechanisms devised in order to implement these protocols. The requirement for asynchronous communication has been satisfied through the introduction of message-passing mailbox-like structures and primitives. The requirement for heterogeneous knowledge integration
has been met by the introduction of translation mechanisms and semantic mapping dictionaries in the agent communication.

Subsequently, cooperative agent behavior was studied, based on the above communication constructs and mechanisms. Cooperation in this context is seen as a problem of distributed agent control (see Chapter 6). The mechanism for achieving this, during the first phase of experimentation, was a static model of the agent's skills and needs and of the group's characteristics through the "my-skills" and "yellow-pages" structures. This social knowledge is used for the control of the individual agent which executes a simple cycle of "problem-solve, if-goal-not-satisfied seek-support." The control mechanism however introduced during this phase, has proved to be highly inflexible, resulting in predetermined reactive agent behavior.

The introduction of domain-independent cooperation heuristic rules and of an agent control engine provided some remedy to this problem during the second phase of the project. The introduced CHs have been determined through a design process which involved the study of documented interacting groups and the analysis of a wide spectrum of domains with existing patterns of cooperative problem-solving characteristics. After definition and testing of a large group of CHs in the chemical emergencies domain, an analysis of emergent behaviors has been undertaken. As a result, generic agent microbehaviors have been individualized from more (domain) specific ones. Aggregates of these microbehaviors in the form of macro-behaviors have been successively identified. A language has been proposed in order to define these behaviors. In addition, a case-based-like semantics of agent acts has been adopted in order to allow consistent interactions and communications in a group of agents.

The flexibility of the system at run time has increased and a capability of experimentation with alternative agent behaviors has been incorporated in our testbed. In particular, opportunistic adaptation of agent behavior, in terms of control of activities, to different problem-solving situations emerged. In the future, it is expected that experimentation with rich coordination and meta-coordination protocols will be tried in the form of new CHs which could provide the testbed with capabilities of dynamic behavior modification at run time.

Understanding the parallel execution of heterogeneous distributed agents is a desirable feature in DAI systems. Debugging and control become extremely arduous tasks because of concurrency, nondeterminism and uncertainty in problem domain activities (see Chapter 3). A number of facilities are provided in the graphical interface of CooperA-II, such as tracing of executions, continuous updating of the status of agents' tasks, and others.

Issues related with user behavior modeling and user interaction are also expected to be studied in the future. As described above, user interaction has been a matter of experimentation during the first CooperA phase, when the user agent was developed. It is our intention in the future, to introduce the user in the flexible behavior programming environment, by incorporating alternative behaviors in the user agent, permitting the user at run time, implicitly or explicitly, to modify its behavior in relation to the problem-solving task.
Based on these experiences of developing and testing the CooperA architectures, other general results concerning architectural features are worth noting.

First, the characteristics of inherent parallelism and uniform message passing to spawn concurrency are inherited by CooperA-II from Actor languages.

In addition, distinctively and differently from MACE's engines, CooperA's start-methods, Actor's script and other architectures, in CooperA-II an agent behaves entirely in accordance with its complex control mechanism and its CHs. The developer can control the interpretation of events and how an agent should act in certain situations by means of the behaviors expressed in the CHs.

Dynamic composition of a group of agents is another interesting feature which allows, for example, extreme modularity of a system, and the testing of individual modules/agents of various complexity.

Finally, problem-solving computations can be expressed in different languages at the application level, while thanks to the abstraction level the knowledge structure is made transparent to the control level (cooperation level) of an agent.

ACKNOWLEDGMENTS

The CooperA research has been accomplished during the period 1987–1989 at the Joint Research Centre (JRC) of the Commission of European Communities, Centre of Information Technologies and Electronics, Knowledge Based Systems Lab, 21020 Ispra (VA), Italy. The CooperA-II work has been part of Sommaruga's Ph.D., as a joint collaboration of JRC-Ispra and Nottingham University, Psychology Dept., AI Group, University Park, NG9 7RD Nottingham UK, during the period 1990–1993 under CEC-grant cat.21/22.

REFERENCES


