Cognitive Abilities in Agents

Beatriz López, Susana Fernández, Javier Bajo, Juan M. Corchado, Raquel Fuentetaja, Manuel Gonzalez, David Isern, Sergio Jiménez and Aïda Valls

Abstract. The aim of this chapter is to describe the cognitive abilities deployed on agents and multi-agent by using examples from applications carried out by the authors. Particularly, the following agent abilities are reviewed: problem solving, memory, decision making and learning capabilities. The results of incorporating such capabilities to agents are the enhancement of the generality and flexibility of the systems.

1. Introduction

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators ([**?**], *Chapter 2*). The agent's choice depends on the observed sequence of inputs. Thus, **intelligent** agents are those whose make the selection as well as possible according to its computational resources (*bounded optimality*). For this purpose, a variety of different agent design are possible. According to [**?**], these range from pure reflex agents to pure deliberative agents: simple reflex agents, model-state reflex agents, goal-based agents, and utility-based agents. First, simple reflex agents select actions on the basis of the current input. Second, model-based reflex agents keep an internal state that captures the history of the inputs. Such state is compared against an internal model of the world, so the agent could know which has been the resulting of its actions. Third, goal-based agents have some sort of goal that describe the situations that are desirable. In order to achieve those goals, agents use search and planning techniques. Finally, utility-based agents use utility functions to make decisions between conflicting goals, about the importance of them, or when there is some uncertainty in the goal attainment.

Agents not only gathers information from the environment in order to act, but also learn from it. They can measure the effect of its actions and modify

its behaviour from experience, adapting to the changing circumstances, updating incomplete knowledge, extending prior knowledge, etc.

Thus, intelligent agents are computational artifacts that combined many cognitive abilities in an integrated system [**?**]: knowledge representation and memory (agent's beliefs, goals, knowledge representation of internal and world states), planning, decision making, learning. Cognitive abilities can be achieved both, at the agent and at the multi-agent levels, since intelligent systems do not function in isolation. The environment provides the opportunity to interact to and learn from other agents ([**?**] *Chapter 2*). They should interact to coordinate their goals, to rely on other agents, so they do not have to learn everything.

In this chapter, we review some cognitive abilities that characterise intelligent agents on the basis of practical examples of existing systems. The organisation of the chapter follows a set of well known cognitive abilities, namely planning, memory, decision making and learning, trying to capture the different kinds of agents. An agent rarely exhibits a single cognitive ability; moreover, in a multi-agent system several agents try to provide a complementary ability to the system. Thus, in the conclusion section we discuss about the different ways of how cognitive abilities have been integrated, and which are the open issues regarding further development of them.

2. Planning as problem solving

A problem is a situation experienced by an agent as different from the situation which the agent ideally would like to be in. There are many approaches to problem solving, depending on the nature of the problem. Most often, the way of solving a problem is executing a sequence of actions (plan) that reduce the difference between the initial and the desired situation. Planning then enables some agent to achieve its goals (solve their problems) given the current state of the world.

This section focuses on automated planning techniques and their use in intelligent agents. First, some basics about planning are provided. Then, several approaches that have been used to combine automated planning and multi-agent systems are introduced. Next subsection provides an illustrative example of a cognitive architecture SAMAP [**?**] that contains a planner agent. Finally, the difficulties of building agents with problem solving capacities are discussed.

2.1. General concepts comprising planning

Any form of general problem solving has these three components:

- *A conceptual model*, i.e., the description of the problems to solve. Problem solving is concerned with the selection and organization of actions to change the state of a dynamic system so it uses a *conceptual model* able to describe dynamic systems. Most of the planning approaches take as their *conceptual model* the model of state-transitions systems but making several assumptions to make it more operational:

1. *Finite state space*. The dynamic system has a finite set of states.
2. *Deterministic world*. When an action is applicable to a state, its execution brings the dynamic system to a single other state.
3. *Static world*. The dynamic system stays in the same state until a new action is executed.
4. *Full observable world*. There is complete knowledge about the current state of the dynamic system.
5. *Restrictive goals*. The planner objective is to find a sequence of state transitions that ends at one of the states satisfying the goals.
6. *Implicit time*. Actions have no duration. The state transitions are instantaneous.

- *A representation language*, i.e., the elements used to describe the problems to solve. In automated planning these languages are notations for representing syntactic and semantically planning problem models. Most of them declaratively describe the objects, actions, states and goals through variants of the first order logic.
- *An algorithm*, i.e., the technique used to solve the problems. The algorithms used to solve planning problems are search algorithms. These algorithms explore graphs systematically trying to find a path to arrive to some goal node n_g starting from a given initial node n_0.

When an agent does not have a complete knowledge of the environment it needs some mechanisms to revise and adapt their plans. Regarding this, the planning model is extended in two different dimensions:

1. *Actions effects*. In many environments the agent can not assume their actions has deterministic effects. Actions may produce different outcomes so the planning algorithms should not look just for a sequence of actions that solve a problem but that solve it very likely.
2. *World observability*. In many environments the agent can not always have a complete description of the current state of the world where it is acting.

As follow, there is a classification of the different planning paradigms according to how these two dimensions are extended:

- Deterministic Planning: is the task of generating a plan to arrive to a state where a set of conditions, the goals, are true starting from a given state, the initial state, in a completely known environment. In these case the agents can do the planning offline, as they need no feedback from the execution of their actions in the world. Most of the research work in automated planning has focused on deterministic planning. Thanks to that, deterministic planning is now a well formalized and well characterized problem with algorithms and techniques that scale-up reasonably well.
- Probabilistic Planning: is the task of finding a robust sequence of actions to satisfy the goals in a completely observable environment where actions can produce possible different states determined by a probability distribution. Sometimes there is no plan guaranteeing always the goals satisfaction. So it

is important to maximize the probability of reaching them, and hence it is important to use information on the probabilities of different effects of agent actions. There are three main approaches to tackle planning problems in these environments: combining deterministic planning and replanning [**?**], extending classical planning [**?**] and Solving Markov Decision Processes (MDPs) [**?**].

- Contingent planning: is the task of solving a planning problem in environments where the current state is not completely known but it is possible to observe some aspects of the current state during the actions execution. Contingent Planning needs to represent planning problems with two extra functionalities: the agent handles a probability distribution over states rather than exactly the current state. These probability distribution over states are called belief states. And the agent has to describe the acquisition of information about the current state in execution time. This can be achieved by enriching the action model with sensing actions [**?**].
- Conformant planning: is the problem of finding a safe plan in a non deterministic and non observable environment. In this kinds of environments the agent has to find a sequence of actions able to achieve a goal state for all possible contingencies without any sensing during the plan execution. There are three main approaches to tackle planning problems in these environments: via Model Checking Planning [**?**], extending deterministic planning paradigms [**?**], and compiling conformant planning problems into deterministic planning problems [**?**].

2.2. Planning systems and Multi-agent systems

Multi-agent approaches and planning systems have been combined in two different senses:(1) to develop intelligent systems able to address the "whole" planning problem (deliberation, execution and control). Thus, the goal of the multi-agent system is to solve a planning problem and each individual agent has one or more abilities to carry out one or more processes involved in the achievement of this goal. And (2) in environments where individual agents generate its own plans but they should coordinate with other agents to solve dependencies and conflicts between their plans. Both approaches are motivated by the fact that usually a unique agent is not enough to deal with real-world problems.

Regarding (1), an isolated planner can not address realistic planning problems. To carry out plans in the real world, the planning tasks has to be complemented with some other processes. First, there has to be processes that interface with the sensors and actuators of the system to put the planned actions into practise. Second, as real environments usually are non-deterministic, there has to be processes that allow the monitoring of the actions' executions [**?**]. And finally, there has to be processes that modify the plans in case the actions' executions takes unexpected paths [**?**]. All these functionality can be easily implemented in a multi-agent architecture where each of these processes is fulfilled by an individual agent. An example of this approach is O-plan [**?**], where a planning agents deals with plan generation aspect, other agents are concerned with aspects such as task

elicitation, plan analysis, reactive execution, plan repair, monitoring, etc. Another example is SAMAP [**?**], whose architecture is described in the next subsection.

With respect to (2), when in real-life problems we have multiple agents having their own goals it is often impractical or undesirable to create the plan for all agents centrally. These agents may be people or companies simply demanding to plan their actions themselves, or refusing to make all information necessary for planning available to someone else. Consequently, agents are able to make their own plans independently of what the other agents are planning to do. However, in many cases dependencies between the tasks of the agents make independent planning impossible. That is, if the agents do not take into account the dependencies between their plans, then they might come into conflict when they try to execute them. To solve their dependencies, agents must coordinate their efforts. From this point of view, a *multi-agent planning problem* [**?**] is a problem where given an initial state, a set of global goals, a set of agents and for each agent a set of its capabilities and private goals, each agent should find a plan that achieves its private goals, such that these plans together are coordinated and the global goals are meet.

According to [**?**], the process of solving a multi-agent planning problem has the following phases:

- Refine the global goals or tasks until subtasks remain that can be assigned to individual agents.
- Allocate this set of subtasks to the agents.
- Define rules or constraints for the individual agents to prevent them to produce conflicting plans (coordination before planning).
- For each agent: find a plan to reach its goals (individual planning).
- Coordinate the individual plans of the agents (coordination after planning).
- Execute plans and synthesise the results of the subtasks.

There are several plan coordination methods as: *coordination throughout filtering*, to filter out those options that are incompatible with the agent's goals; *Generalize partial global planning (PGP)*, where agents cooperate because no agent has complete information; and *plan merging*, which is used for agents that are able to create a valid plan on their own. A more detailed description of these methods can be found in [**?**] and [**?**].

An example of the application of coordination techniques to planning in a multi-agent system is [**?**]. The goal of this system is to produce an applicable action sequence under complex constraints for spacecraft missions. In that system, each spacecraft subsystem is represented by a planning agent, and they cooperate with each other in order to build plans of actions needed to achieve given goals. The knowledge space is partitioned into sub-space and each agent is in charge of managing and operating upon a given subset of domain, called the agent domain. The multi-agent planning system can lead several improvements in terms of efficiency and reliability of planning activity over distributed environment.

2.3. SAMAP

SAMAP is an example of a cognitive agent architecture that combines several cognitive abilities in an integrate system. The goal of SAMAP is to build a software tool to help different people visit different cities. It captures and updates a user model about different city visits dynamically, analyses past planning behaviour of the tourist and similar tourists in the same type of visit, and selects a list of places that have a high probability to be interesting for the tourist through a case-based reasoning (CBR) approach. Then, taking into account distances, places timetables, etc., it computes a plan, and also shows how to go from one place to the next in the plan. This system is intended to work in portable devices (mobile phones, PDAs, etc,) with Internet connection.

SAMAP has been built as a multi-agent system, consisting of three main agents: user modelling and interface agent, case-based agent, and planning agent. It also integrates an ontology to facilitate the information exchange among these agents. The ontology stores information about the tourists and their preferences, the activities that can be performed in a city and the city itself.

The first step consists of building the user model. This requires the tourist to enter personal information, that is, personal data, interests and preferences about, i.e., art, monuments, meals, etc. Also, the tourist should specify which city is going to visit, under which schedule, etc. This information can be gathered by using any device with Internet connection. In order to obtain more interesting data about the tourist, the system (by means of machine learning techniques) can use past information about the same tourist (provided the tourist has used the application before). This information is stored in the ontology.

The second step consists of the generation of a list of activities that the tourist might like to perform in the current visit according to the preferences. The activities might come directly from the tourist, or automatically generated by CBR from similar tourists plans in the city or similar cities. Each activity will also be described with the expected utility that performing this activity might have for the tourist. This utility can be directly specified by the tourist, or computed by SAMAP from knowledge about similar tourists.

The last step is the computation of the tourist plan by taking into account the previously computed list of activities.

At the multi-agent level, the ontology shared by all agents, constitutes the first cognitive ability of the system (Memory). At the agent level, the User Model agent is provided with the learning abilities, and the Planning agent with the problem-solving abilities.

2.3.1. Planning agent. One of the inputs of the planning agent is the list of activities selected by the CBR agent. This list is not directly the goal of the planning problem, even if the CBR has ranked activities and made a selection, as it can still contain more places than the tourist will be able to visit because of schedule or movement constrains among places. Therefore, the planner must select which of them should be included in the plan. Moreover, the planner must schedule each

visit according to the place timetable, deal with the city map, etc. This planning task has several features that make it hard for current planners:

- **Time management:** each visit should be scheduled according to the opening hours of each place and the expected duration of the visit according to the user model. Also, it should consider the time to go from a place to another.
- **Preferences:** the ontology contains knowledge about user preferences or constrains such as time-to-eat, utilities of visiting, places, types of food...
- **Management of numerical values:** the prices of the visits, meals and transports must not exceed the available budget.
- **Locations:** the planner should indicate how to go from one place to another, that is, which transport the tourist should take. In case it is preferable to walk, the planner should specify which route the tourist should follow.
- **Goals:** three types of goals have been specified:
  - totally instantiated goals, i.e., visit a specific museum
  - partially instantiated goals, i,e., generic goals like visiting any museum
  - a metric indicating that the plan must maximize its utility

  Moreover, not all the available places must be visited, that is, not all goals will be achievable, because of scheduling constrains related to timetables (one cannot enter Prado's museum at night) or to the available time of the tourist (one cannot visit five places if there is time to visit only two). This problem is related to the over-subscription problem in planning [**?**] and scheduling [**?**].

As it is difficult to use any of the current planners for solving the whole planning problem, a hybrid system was proposed composed by a planner and some other modules, based on the ideas in [**?**]. Thus, besides the planner module, the planner agent in SAMAP is composed of four more modules: the Translator module that transforms the original list of activities into the predicates required by the planner, the Control module that coordinates the rest of modules providing the input that they need, the Selector of activities module that selects the most appropriate actions to be solved by the planner each time and the Transport module that receives an origin and a destination point and returns the transport subplan for moving a person from the origin to the destination.

As said above, the input of the system is a ranked list of activities that represents the places of interest of the tourist (including eating and leisure places) together with a number indicating their utility, i.e the *satisfaction degree* that visiting such site will potentially provide the tourist (computed by the CBR agent). Each activity can be totally or partially instantiated, as in `visit museum of Modern Art` or `visit a museum`. The output is one or more tourist plans which contain a list of scheduled visits along with the indications about how to move from one place to another. The system also computes the cost of the plans trying to maximize its quality according to the established metric. By default, the quality metric is to maximize the total utility, but it could be to diminish the cost, a combination of both or any other one.

The final step of the planning system is to store the generated plans into the ontology.

2.4. Discussion

The straightest way to endow an agent with a problem solving capacity is to integrate an existing planner in the agent architecture. However, the planning task resulting from real world problems can be computationally too hard even for state-of-art planners. So, it is necessary to help the planner somehow.

In the SAMAP particular case, the planner agent assists tourist in their visit to a city, using a PDA or a third generation mobile phone. Visits are adapted according to the tourist preferences. So the system only proposes a list of activities that could be really interesting and achievable for the tourist. The input of the planning agent is an original list of activities, selected by a CBR agent, that represents the places of interest of the tourist (including eating and leisure places) together with their utility. The output is one or more tourist plans that maximize the utility, including as many visits as possible, along with the indications about the transports needed to move between the different places (the transport subplans). This planning task resulting from the analysis of this tourist problem was a difficult task even for the state-of-art planners. Therefore, SAMAP uses an hybrid planner agent composed by a planner and other modules that reduce the task of planning itself. This way, the planning agent can solve complex problems that otherwise the planner alone could not solve.

Another solution might have been to use a CBR mechanism throughout the system. The use of a planning approach with "classical" operators was chosen because (a) there is not a CBR planning tool that allows to develop applications quickly and (b) in big cities where there are a number of different goals and ways to go from one place to another and scheduling the visits, a very rich adaptation agent could be needed. Therefore, it seems reasonable to adopt the following solution: the planner takes this information as input together with information about the city (streets and intersections, situation of each place, etc.) to compute the plan for the user.

3. Memory and BDI

The BDI model [**?**] has traditionally been used to describe the agents' mental process, as intentional entities. However the BDI model presents certain limitations. One of these limitations is the memory management. The method proposed in [**?**, **?**] facilitates the incorporation of Case-based Reasoning (CBR) systems as a deliberative mechanism within BDI agents. CBR is a type of human thinking based on reasoning about past experiences. The CBR system allows the BDI agents learn and adapt themselves, lending them a greater level of autonomy than pure BDI architecture [**?**].

In this section a deliberative agent architecture based on the integration of deliberative Beliefs-Desires-Intentions (BDI) agents within Case-Based Reasoning

systems (CBR) is presented. First some generalities are provided, and then a practical case illustrates the approach. Finally some conclusions and the future research line are presented.

3.1. CBR-BDI and CBP-BDI agent architectures

The relationship between CBR systems and BDI agents can be established by implementing cases as beliefs, intentions and desires which lead to the resolution of the problem. In the CBR-BDI agent [**?**, **?**], each state or problem description is considered as a belief; the objective to be reached or problem description once the objective has been reached may also be a belief. The intentions are plans of actions that the agent has to carry out in order to achieve its objectives, so an intention is an ordered set of actions [**?**, **?**, **?**]; each change from state to state is made after carrying out an action (the agent remembers the action carried out in the past, when it was in a specified state of similar characteristics, and takes a decision depending on the subsequent result). A desire is any of the final states reached in the past (if the agent has to deal with a situation, which is similar to one in the past, it will try to achieve a similar result to the one previously obtained). Moreover, in order to obtain a complete integration, the agent has to execute a CBR reasoning cycle. A CBR-BDI agent implements a CBR cycle by means of behaviours executed sequentially. All the behaviours concerning one of the stages of the CBR cycle is defined into an agent capability. CBR-BDI agents possess three (if the revision stage is external to the system) or four capabilities in charge of the implementation of the CBR cycle [**?**, **?**]. Each of the capabilities contains the behaviours (one or more) required to complete its corresponding task: case retrieval task (case memory indexing, similarity algorithms), solutions reuse task (adaptation algorithms to reuse past solutions or to search new solutions, planning algorithms in the case of CBP), proposed solution revision task (algorithms to evaluate the solution proposed in the reuse stage) and learning task (algorithms to update both case memory and knowledge memory). The CBP-BDI agent architecture incorporates a model based on variational calculus, providing the option of planning and replanning in execution time [**?**]. Every time the environmental changes interrupt the intentions or plans carried out by the agent in order to achieve its objectives, the agent is able to react without "undoing" the previous executed actions, and constructs a new plan [**?**, **?**, **?**].

3.2. A practical point of view

From a practical point of view, some of the architectures successfully implemented in the last years look at CBR-BDI agent architectures as frameworks. CBR-BDI agent architecture, as well as CBP-BDI agent architecture have been applied to different real problems in order to obtain a preliminary evaluation and conclusions about their behaviour working in real environments. In particular, this agent architecture has been applied to resolve a wide range of problems, such as the monitoring and evaluation of the carbon dioxide exchange rate between the ocean water surface and the atmosphere [**?**, **?**, **?**], the development of a guiding systems to

be applied not only in tourism recommendation problems [**?**] but also in a guiding system for the users of a shopping mall that helps them to identify bargains, offers, leisure activities, etc. [**?**, **?**], the automation of the management of internal mail in a department using mail robots responsible for mail delivery [**?**] or the incorporation of ambient intelligence techniques for health care in geriatric residences [**?**]. The results obtained have been very promising and demonstrate the versatility of the architecture.

3.2.1. Carbon dioxide exchange monitoring. One of the factors of greatest concern in climactic behaviour is the quantity of carbon dioxide (CO_2) present in the atmosphere. Traditionally, it has been considered that the main system regulating CO_2 in the atmosphere is the photosynthesis and respiration of plants. However, in the last decades it has been shown that the ocean plays a highly important role in the regulation of carbon quantities the full significance of which still needs to be determined. The implementation of an open multiagent system which incorporates a CBR-BDI agent, facilitates to automatically monitor the interaction between the ocean surface and the atmosphere. Initially, the system is being used in order to evaluate and predict the amount of CO_2 absorbed or expelled by the ocean in the North Atlantic. The initial results have been very successful from the technical and scientific point of view [**?**, **?**, **?**].

A multi-agent system that includes deliberative and pure reactive processes has been implemented using the SIMBA platform. The approach allows the integration of unbounded deliberative processes with critical real-time tasks. In the case study proposed for the evaluation of the hypothesis, the SIMBA architecture was integrated with both ARTIS agents, (which are capable of guiding mobile robots in real time), and CBP-BDI deliberative agents (which generate and distribute plans in the execution time of the ARTIS agents). Therefore, the deliberative agents were responsible for planning the routes that should be followed by the mobile robots, and the ARTIS agents put these plans into action until insurmountable obstacles were encountered, in which case an alternative plan is requested from the deliberative agent [**?**].

3.2.2. Plan recommending. The CBR-BDI and CBP-BDI architectures have also been used to construct a model for recommending plans in dynamic environments [**?**, **?**, **?**]. The proposal presented in [**?**, **?**] has been used to develop a guiding system for the users of a shopping mall that helps them to identify bargains, offers, leisure activities, etc. CBP-BDI is a highly appreciated tool that optimizes the time spent in the shopping mall. Both users of the tourism [**?**] and mall [**?**, **?**] applications have noticed the utility of the dynamic replanning, since it is quite usual for them change opinions/objectives in the middle of a plan.

3.2.3. Alzheimer patient monitoring. Finally, an autonomous intelligent agent has been developed for ambient intelligence health care in geriatric residences [**?**]. The agent operates in wireless devices and is integrated with complementary agents into a multi-agent system, named ALZ-MAS (ALZheimer Multi-Agent System),

capable of interacting with the environment. Ambient Intelligence (AmI) provides an effective way to create systems with the ability to adapt themselves to the context and users necessities. The vision of AmI assumes seamless, unobtrusive, and often invisible but also controllable interactions between humans and technology. The AGALZ (Autonomous aGent for monitoring ALZheimer patients) is designed to plan the nurses' working time dynamically, to maintain the standard working reports about the nurses' activities, and to guarantee that the patients assigned to the nurses are given the right care.

3.3. Discussion

In this section we have described the particular model of a BDI agent in which memory is managed though a case-based approach. This software-engineering approach, however, should be analyzed in the context of future agent designs. It is a general feeling that, in next years new proposal to design agents will emerge in order to build up a type of architecture for agents from biological and situated cognitive abilities. The combination of both approaches would allow the deployment of complex behaviors that would make intelligent agents capable of tackling with dynamic environments. Regarding cognitive abilities, it would be important to be able to develop architectures in which building up representations and using this representations to plan ahead emerge from agent-environment interaction.

Thus, future explorations in the agent field should include implementations of sufficient environmental complexity to allow for the possibility of causal correlations between ordered states of evolving organisms and ordered states of their environments. Such inclusions are likely to provide a base from which minimally cognitive artifacts can survive and constitute artificial individuals capable of the acquisition of novel information in their individual lifetimes.

4. Decision making

While problem solving techniques, as planning or searching, deal with goal achievement, decision making aims at choosing a goal among a set of conflicting goals, establishing the importance among them, or dealing with uncertainty in goal attainment. This cognitive capability is then particular important when intelligent agents support the users is the complex task of taking decisions (where to go, which place to visit, etc.).

Intelligent agents can have the cognitive ability of knowing the preferences of its owner and use this information to make recommendations in order to allow the user to take a better decision. Particularly, in this section the case of agents that give decision support when the user is faced with a set of alternatives described by means of multiple criteria is presented. First an introduction to multiple criteria decision making (MCDM) models as a cognitive ability for agents is provided. Then, it is described an example of the use of such methods into a multi-agent system called HeCaSe2 [**?**], which provides patient-oriented services in the healthcare domain.

4.1. Multiple criteria decision making

When a person has to analyse a set of alternatives with respect to a set of criteria in order to take a decision, we say that this is a multiple criteria decision making problem. In the 19th century economists and mathematicians started to study the laws and behaviour of this type of problems (Pareto, VonNeumann, Morgenstern). In the end of the 20th century it became again a widely studied area, and many methods have been developed for helping in this common human task. An MCDM problem is the one that having a set of alternatives A, with respect to a set of criteria C, either aims to find a subset of alternatives that contain the best ones (selection problem), an assignment of the alternatives into predefined categories (sorting into ordered categories or classification into unordered ones) or a ranking of the alternatives from the best to the worst (ranking problem) [**?**]. The main difficulty lies in the fact that it is an ill-defined mathematical problem because there is no objective or optimal solution for all the criteria. Thus, some trade-off must be done among the different criteria to determine an acceptable solution for the decision problem.

There exist two main approaches to the solution of a MCDM problem, which are based on multi-attribute utility theory and outranking methods. The first one, named MAUT, is based on the idea that any decision-maker attempts unconsciously to maximise some function U that aggregates the utility of each different criterion. The key issue in utility-based approaches is the determination of the marginal utility functions, U_j. These functions transform the scale of the corresponding criterion into utility values. The second approach, called outranking relations, are based in the fact that the decision-maker provides pairwise comparisons of the alternatives to determine the preference of each alternative over the other ones for each particular criterion. Each criterion usually leads to different evaluation of the alternatives, so the problem is to find a consensus in the ranking. The outranking methods are based on the definition of a concordance relation and a discordance relation. They are used to find the dominance relation over the alternatives, which is the basis for solving the decision problem.

Focusing on the MAUT-based solutions, the methods consider two stages. In the former, called aggregation stage, a global utility rating for each alternative is computed $U(a)$; then, the second stage proceeds to sort out the decision problem, selecting the best alternatives according to its rating, ranking them, or sorting or classifying them into some predefined categories.

The problem of aggregating information has been widely studied and there exist several methods to aggregate numerical values as well as linguistic terms. A description of the operators and the properties required for decision making can be found in [**?**]. Although a wide range of operators is available, in most cases, the aggregation is done using some form of weighted arithmetic mean. This operator reflects a compromise behaviour among the various criteria. However, sometimes this is not a good approach because a compensative behaviour is not desired. In [**?**], a study of this type of operators and its generalisation is done.

4.2. The HeCaSe2 system

HeCaSe2 is an agent-based system that provides healthcare services to patients and medical professionals [**?**]. Several agents that play different roles coordinate their activities to provide a user-centred assistance. In Figure 1 the architecture of HeCaSe2 is shown, which includes different medical structures organised hierarchically (doctors, nurses, medical devices, departments and the medical centre at the top level). Inside this scheme, several organisational and medical rules guide the behaviour of the agents. HeCaSe2 is designed as an open-architecture of agents. Any healthcare institution can be modelled with different topologies and organisation rules and patterns. In any case, agents that represent doctors and patients (*i.e.* users) are always required.

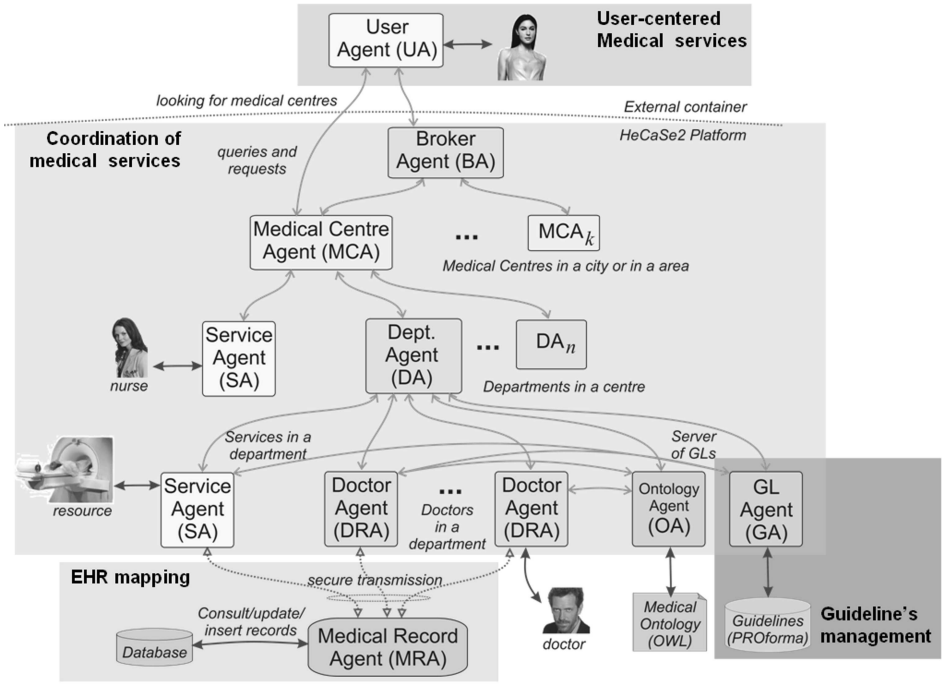

FIGURE 1. HeCaSe2 multi-agent system

In order to provide patient-centred services, the system maintains a user's profile that stores his/her preferences with respect to some aspects of the patient care. In addition, the User Agent stores the agenda of its owner (which is private and is only used by this agent).

The case of arranging an appointment between a patient and a medical unit can illustrate the possibility of including decision support abilities to the agents.

This is one of the problems that requires more communication and negotiation between the entities involved in patient care.

In HeCaSe2 the decision-making process is performed by a Doctor Agent, using a set of possible alternatives (*e.g.* proposals of appointments in different units that can perform the activity required by the doctor, such as a blood analysis). To obtain those alternatives, the system must start a communication with other agents at different levels of the hierarchy shown in Figure 1. The Doctor Agent collects all the possible appointments proposed by the different units (through the Service Agents) and then uses the patient profile to rank and filter them, in order to recommend to the patient only the best N options.

A set of five criteria to make that recommendation were selected. As it has been said, the personal preferences about these criteria are stored in the user's profile. In particular the most relevant information of any appointment proposal considered is: the day of the week, the period of the day (morning, afternoon, night), the medical centre where the test should be performed, the distance between the centre where the doctor performs the medical visit and the centre where the the test should be performed and the number of days to wait for the test. Some of that information is stored using numbers and other using a fixed set of linguistic terms. In the following section, the transformation of that data into utility values is explained.

4.3. Decision support in HeCaSe2

HeCaSe2 agents use a multi-attribute utility approach to give decision support to the patient when he/she needs an appointment for a medical test or visit. That is, each alternative is described using a set of utility values (one for each criterion). Then those marginal utilities are aggregated to obtain a global rating value. Using those ratings the system selects the best N options, that are presented to the user.

To obtain the utility value of each criterion of an alternative, the user's preferences are taken into account given by different types of transformation functions, depending on the type of value ([**?**]). For the case of numerical values (distance and waiting time), the case of lineal, polynomial, exponential and logarithmic transformation functions were studied. The conclusions shown that the exponential function is the most adequate because it allows to distinguish better the close samples. The result is a numerical value in the interval [0,1]. For the case of linguistic variables (day, period and medical centre), the user associates to each possible term in the domain of the variable, a linguistic preference value in a fixed scale S. For example, U_{period}*(morning)=low*, U_{period}*(afternoon)=high*, and U_{period}*(night)=none*. The selection of the vocabulary S is very important because it determines the degree of expressiveness of the criterion and its semantics. In addition, it definitely influences the rest of the decision making process. In [**?**], this aspect was analysed in detail. Finally, the linguistic term set used was formed by nine symmetrically ordered terms, S={*none, very low, low, almost medium, medium, almost high, high, very high, perfect*}.

As it has been said, to perform the rating of the alternatives, an aggregation operator must be used. In this case, the Linguistic OWA operator [**?**] was selected. It belongs to the family of OWA operators (Ordered Weighted Averaging operators [**?**]), whose main characteristic is the possibility to establish different decision making polices with respect to the aggregation of the values. The OWA operators are in the class of mean operators, they are idempotent, monotonic and commutative. The LOWA aggregation operator works with linguistic values instead of numbers. In [**?**], the numerical-linguistic and linguistic-numerical transformation processes are defined. Those processes are used in the HeCaSe2 system.

In the Figure 2, a representation of the decision making procedure is presented followed after receiving the set of appointment proposals: (*1*) describe each alternative with the corresponding partial utilities according to the patient's preferences, (*2*) put all the utilities into the common linguistic domain S, (*3*) perform the aggregation-based rating of alternatives received using LOWA and find the best options and (*4*) enter to the learning stage, which allows to adapt the user's preferences. This last stage is explained in the next section.

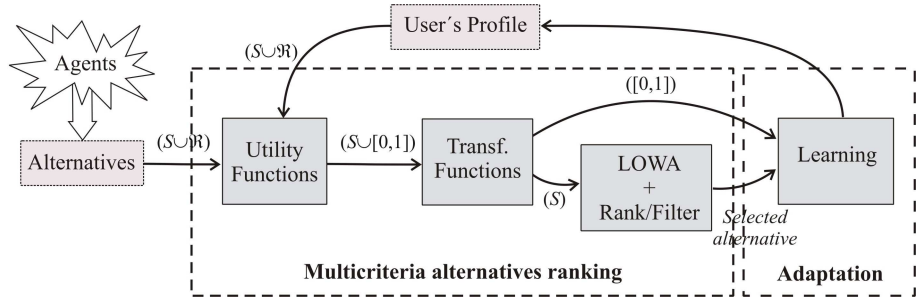

FIGURE 2. Decision making procedure implemented

4.4. User's profile adaptation

As explained above, when the user receives a set of alternatives to consider for an appointment, the list of proposals is rated and ranked according to the preferences stored in the user's profile. The following action done by the patient, through his User Agent, is the selection of the most appropriate alternative for him. This action gives to the system a very important feedback. If he selects the first option, it means that the decision support process works fine, but if he selects another alternative, it means that for some reason the algorithm has rated too low the most appropriate alternative to the user. The main goal now is to adapt the user's profile with this information, in order that if the same situation is repeated in the future, the alternative selected by the user becomes the first one.

In [**?**] the adaptation algorithm designed and used in HeCaSe2 is explained. It basically consists in comparing the appointment selected by the user (proposal a_i) with the ones that were ranked in higher positions (proposals from a_0 to $a_i - 1$).

However, the comparison of a_i to all of them are not directly performed, instead, all are automatically gathered into two clusters, and a_i is compared with the two prototypes with the purpose of identifying the closest prototype, which is selected as the target to reach. With this approach the algorithm tries to adapt the patient's profile smoothly along the time. Similarly, the user's profile is not changed completely at each decision. Only the most relevant utility values are modified (those that are more different in the target with respect to the selection made).

4.5. Discussion

This section explains how a MCDM-based method is applied to a distributed problem that requires to adapt some results to the user's preferences. All required data is scattered among different partners and the user's preferences are used both to rate and filter the results.

Adding this kind of cognitive abilities to the agents allow the system to improve the quality of its responses to the user. Moreover, it provides personalised services that make specific recommendations to each user depending on his/her profile. The key point of this approach is that the user provides some kind of information about his/her preferences, which sometimes is not possible to obtain.

In the case study presented, a method for implicitly adapting the user's profile with some feedback information obtained form the use of the system have been mentioned briefly. This is also another key point to take into account on the design of agents based on profiles. The preferences of the user may change through time, and the profile must be adapted accordingly. However, an automatic user's profile adaptation must be done carefully because different parameters must be defined which can lead to a bad updating.

Finally, it is important to mention that in MCDA there are some basic elements of the problem that influence the goodness of the final result: the set of criteria considered, the set of linguistic terms to express preferences or the aggregation operator and its parameters. First of all, the criteria must be representative enough of the information considered by the user to make the decisions. As the number of criteria increases, the performance of the aggregation and profile updating increases. The second issue is the selection of the set of linguistic terms and its semantics. It should be considered at least five different terms in the vocabulary and no more than eleven. With respect to their semantics, there are different approaches, but the most common one is the use of fuzzy membership functions, that should be tuned appropriately to the problem characteristics. Finally, the selection of the aggregation operator (which is the most important cognitive element of the process) must be according to the properties required in each particular decision problem (*i.e.* neutrality, monotonicity, compensativeness).

5. Learning

According to Minsky [**?**], agents can learn by useful changes in its workings. From observation, agents can combine several descriptions into one. Agents can also accumulate descriptions and form new concepts, or modify descriptions, as well as functions or actions. From observation, agents can also be aware of the effects of their actions, giving them the opportunity to learn about their decision making process. From an operational point of view, machine learning researchers have been dedicated to the study of how to construct computer programs that automatically improve with experience [**?**]. The incorporation of machine learning methods to agents opens the opportunity to automatically incorporate new capabilities to agents, adapt their behavior, and improve its performance.

Learning in a multi-agent system can be achieved at different levels: individual (isolated learning) or collective (interactive learning) ([**?**] *Chapter 6*). In the former agents learn to improve their individual performance. While in the latter, agents learn about other agents in order to obtain the maximum revenues from a collectivity. In this section two practical experiences of both situations are reviewed. First, a diagnosis system is explained in which an agent improve its diagnosis capabilities by both its individual experience and with the cases provided by other agents in the neighborhood. And second, a recommender system that forgets according to its individual experience and learns about the other agents' behavior is described.

5.1. Diagnosing from experience

In [**?**] a case-based agent was proposed to diagnose acute strokes. Medical treatment of such illnesses have changed quickly in the last years. Drugs are quite specific depending on the kind of stroke, and physicians require of supporting tools that help them to identify the clinical category of strokes.

In order to facilitate the exchange on experiences the case-based agent was integrated in a multi-agent system. The multi-agent system is organized according to the different hospitals in a given zone (see Figure 3). Currently, four hospitals are represented by a different case-based agent (CB-agent) in the architecture. Then each CB-agent asses the diagnosis process according to the criteria of the physicians in a given hospital, and cooperates with other CB-agents when the assessment provided within an hospital is not significant.

At the agent level, each agent relies on a Case-Based Reasoning (CBR) approach. CBR is applied with two different goals: when trying to find a diagnosis, and when trying to provide cases to another agent. In the first case, the four steps of the CBR cycle according to [**?**] have been extended to include using cases provided by other agents (see Figure 4). In the second case, a single retrieval step is enough. Learning, in any of the situations, is performed by accumulating new cases, because accumulating experiences is a key issue in medical practice.

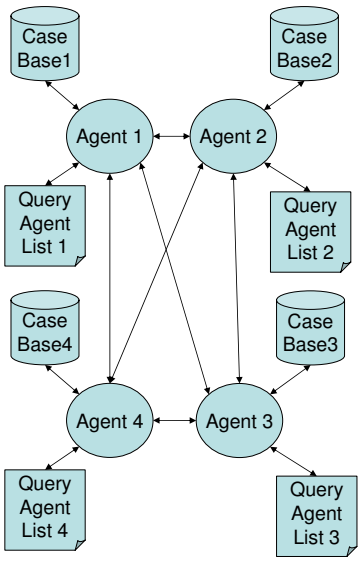

FIGURE 3. The multi-agent system in which agents cooperate when solving a problem

One particularity of the approach is the case structure, since cases are trees. A case was formally defined as follows:

$$C_i =< D_i, S_i > \tag{5.1}$$

where D_i is the problem description, and S_i the problem solution.

The notation used to represent the attributes is a set of nested attribute-value pairs. That is, D_i is defined as follows:

$$D_i = \{(x_j, v_j)\} \tag{5.2}$$

where $x_j \in X$, X is the set of attributes used to describe the problem, and v_j is the value of x_j which can be a list $\{(x_k, v_k)\}$ or a single value.

The problem solution of the case consists in the clinical category. Formally,

$$S_i = (clinicalCategory, v_j) \tag{5.3}$$

where v_j is one of the five possible values of clinicalCategory (*atherothrombothic, cardioembolic, small vessel disease , other, undefined*).

Regarding the CBR cycle (see figure 4), different methods have been defined for retrieval, reuse, revise and retain.

First, in the retrieve phase, given a current (new) case to be diagnosed, the most similar cases are retrieved from the case-base. This phase of the cycle consists on the following steps:

1. Matching the current case against all cases in the memory

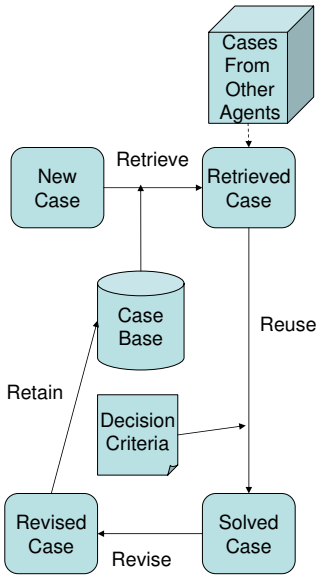

FIGURE 4. The different phases of the CBR cycle

2. If the most similar cases have a similarity degree less than a given threshold θ, then:
   (a) Let be $\theta' = \theta$
   (b) While there are agents to ask and $\theta' \geq \theta$ do,
      (i) Ask the next agent for relevant cases
      (ii) θ = highest similarity degree of the provided cases
3. Selecting the k most similar cases

Note, then, that the classical view of the CBR retrieve has been extended in order to include collaboration with other agents in the environment.

The similarity metric used in the matching procedures is defined based on the similarity between two trees, according to the case structure. It is defined as a weighted average as follows:

$$sim(D_{new}, D_{mem}) = \frac{\sum_{i \in D_{new}} \omega_i sim_i(v_i^{new}, v_i^{mem})}{\sum_{i \in D_{new}} \omega_i} \tag{5.4}$$

where ω_i is the weight expressing the relevance of the i attribute, and v_i^{new} and v_i^{mem} are the values of the i attribute in the new and memory case correspondingly, and $sim_i(v_i^{new}, v_i^{mem})$ the similarity between these values. Note that if attribute i is not present in the memory case, this function is assumed to be 0.

Given an attribute x_i, the similarity of two of their values is computed as follows:

$$sim_i(v_j, v_k) = \begin{cases} 0 & \text{if } v_j \text{ is a single value and } v_k \text{ is a tree structure} \\ 0 & \text{if } v_j \text{ is a tree structure and } v_k \text{ is a single value} \\ 1 - \delta(v_j, v_k) & \text{if } v_j \text{ and } v_k \text{ are single values} \\ sim(v_j, v_k) & \text{otherwise} \end{cases} \quad (5.5)$$

where δ is the Hamming distance of two values, and $sim(v_j, v_k)$ is the similarity between two trees. On one hand, δ is computed as follows:

$$\delta(v_j, v_k) = \begin{cases} 1, & \text{if } v_j \neq v_k; \\ 0, & \text{otherwise.} \end{cases} \quad (5.6)$$

On the other hand, note that using $sim(v_j, v_k)$ inside the definition of $sim_i(v_j, v_k)$ makes the similarity measure recursive.

Once all the similarities of the current case with the memory cases have been computed, the selection step consists in choosing the most similar cases. In the CB-agents, a k-neighbor approach was followed (so the k-most similar cases were retrieved). Particularly, with k=5 the results were good enough.

Note that always some cases are gathered either from the memory or from other agents. In the best case, the retrieved cases are over the θ threshold .

In the reuse phase the k-most similar cases are used to elaborate the solution to the current case. In the revise phase, the case-based agent requires some feedback from the user in order to know whenever it has been successful or not. Finally, all cases are retained in the memory because accumulating experiences is a key issue in medical practice.

5.2. Recommending from experience

GenialChef [**?**] is a recommender system based on case-based learning and trust. From the former, the agent is able to improve its recommendations to a user. From the latter, the agent also improves its recommendations and gives innovative proposals to the user thanks to the information provided by other agents interactions.

5.2.1. Case-base learning. The case-base system consists of a set of previous items explicitly and/or implicitly assessed by the user. The initial case base is empty, and the CBR agent uses a training set to learn an initial profile of the user preferences from a set of training cases. The profile is improved by accumulating experiences.

One of the most relevant issues of the approach in GenialChef was the incorporation of a drift attribute in order to adapt the case-base to the user interest over time. The drift attribute belongs to the case description and its function is to age the cases in the case base. For this purpose, a reinforcement mechanism was used to update the drift values.

As described in [**?**], the drift attribute works as follows:

- The drift attribute value is confined to the [0-1] interval.

- New items are inserted in the case base with the maximum drift value when the user shows some interest about them.
- The value of the drift attribute is decreased over time, emulating the gradual process of people losing interest in something. The decreasing function is a simple one where the drift attribute δ_q of a case q is decreased by multiplying the last drift value by a factor β of between 0 and 1 (see Equation 5.7).

$$\delta_q = \delta_q * \beta \tag{5.7}$$

  The system decreases drift attributes each time a new item is incorporated into the case base. When a case reaches a drift value under a certain threshold (ξ), it is discarded.
- The value of the drift attribute is increased (rewarded) if the retrieved case results in a successful recommendation. The rewarding function is as simple as the decreasing one. The drift attribute δ_q of a case q is increased by dividing the last drift value by a factor λ of between 0 and 1 (see Equation 5.8).

$$\delta_q = \delta_q / \lambda \tag{5.8}$$

Both, the decreasing and increasing operations define a reinforcement mechanism.

5.2.2. Trust learning. The incorporation of trust in GenialChef enables the definition of what has been call the opinion-based filtering method of recommended systems [**?**]. Each agent has a list of friend agents in the neighborhood in which it relies: $C_i = \{(a_{i_1}, t_{i,i_1}), (a_{i_2}, t_{i,i_2}), \ldots, (a_{i_n}, t_{i,i_k})\}$, where a_{i_j} is an agent identifier and t_{i,i_j} is a number between [0,1] that represents the truth value the agent a_i has on agent a_{i_j}. The opinion-based filtering method consist on querying this agents in a lack of information situation (see more details on [**?**]).

Initially the contact list is empty. Thus, agents contact other agents in the world and learn the initial trust using a *playing agents* procedure following [**?**].

Trust values are updated according to a reinforcement mechanism, so the recommender agent learns about the benefits of the interaction performed with other agents in the neighborhood. Thus, for every agent a_{e_i} in the contact list of the agent a_q, his/her trust value $t_{q,i}$ is updated as follows:

$$t_{q,i} = \varphi * t_{q,i} + (1 - \varphi) * r_{real} \tag{5.9}$$

where r_{real} is the real interest of the suer on the product (see details in its computation on [**?**]) and φ is a parameter of the system that manages the evolution dynamics of trust.

Other approaches to trust are described in chapter **??**.

5.3. Discussion

This section describes several practical systems of learning capabilities in agents. Particularly, the case-based paradigm is shown to be a valid paradigm to learn from experience in different domains and situations. On one hand, isolate learning

allows the improvement of the agents decisions (how to improve diagnosis, how to improve recommendations) . On the other hand, learning by the interaction with other agents allows to improve agents collaboration (when to rely on other agent's outcomes, trust).

One of the major issues regarding leaning, however, is the cold-start problem, that is, the set up of the learning mechanism employed by agents. In the GenialChef system described, this problem is bypassed by the use of a training set which is used for both, training the internal problem solving capability of agents, and for training the interaction model of the neighbour agents (trust).

Other problems as the tradeoff between diversity and coverage should also be taken into account. That is, not all agents experiences should be accumulated, and only the ones that provide some diversity should be retained. The system studied in this section proposes a drift attribute; but some other maintenance mechanism should be explored. Finally, the different parameters that configure a learning mechanism, as learning threshold, learning rates, are also a matter of discussion in the design of learning capabilities.

6. Conclusions

In this chapter, several cognitive abilities that feature agents and multi-agent system have been described and illustrated by means of several practical applications. Particularly, the following abilities have been described and illustrated: planning, memory, decision making and learning. First, planning as a kind of problem solving, has been analysed in the context of recommending tours to citizens. Second, complex memory structures have been described in the deployment of Belief-Desire-Intention agent architectures that have been proved useful for dealing with carbon dioxide exchange monitoring, recommender systems, and alzheimer patients monitoring. Third, decision making methodologies have been used to asses the adaptation of the user preferences, as shown the practical applications on the healthcare domain. And fourth, case based learning and trust learning have been explained as part of the learning abilities in agents in a medical diagnosis problem and a restaurant recommender domain correspondingly.

Some intelligent agents analysed combines different cognitive capabilities, while other approaches integrates different abilities at the multi-agent level. Whether to integrate several abilities at the agent level or to combine them in a multi-agent architecture is a difficult design decision. At the agent level, for example, there are different agents in the HeCase system able to make decisions, as well as learn from the environment and build a user model. Another interesting example are the CBP-DBI agents analysed in section sec:memory which uses case-based reasoning to deal with memory management, and combines planning and adaptation techniques to achieve agents' goals. Most of the practical cognitive abilities at the agent level seen along this chapter rely on case-base reasoning approaches. Such

paradigm seems a valid practical approach to integrate problem solving, memory, and learning.

On the other hand, at the multi-agent level, other multi-agent system that integrates cognitive abilities have been described. For example, the SAMP system is composed by a set of agents, each of them exhibiting a particular cognitive ability, mainly case-base reasoning and planning. A particular case is the multi-agent system described in section 5.1 in which all the agents have the same cognitive capabilities, but they differ in their past experiences. Note, however, that agent interaction at the agent level is being used from a complementary point of view, assuming that all the agents are collaborating for achieving a common goal. In a future, it is important to consider other scenarios, specially competitive scenarios, in which cognitive abilities would play an important role. Such capabilities, combined with the new paradigms coming from the research on agents and multi-agents systems, namely, negotiation, argumentation, etc., are in the right way of creating artifacts that enhance generality and flexibility of intelligent systems.

Thus, agents and multi-agents systems have found the way of integrating different subfields of intelligence, planning with learning, decision-making with learning, and others. However, along the chapter, several problems have been highlighted in the design of cognitive capabilities, mainly the parameters required by the cognitive mechanisms that influence the goodness of the final results. For example, in the decision making methods, the set of linguistic terms, the set of criteria, or the aggregation operators have been identified as critical elements on the definition of a MCDA method. Other examples are the learning rates and thresholds of case-base reasoning. The definition of this kind of parameters depend at some extend to the skill of the intelligent agent designer and it is a difficult issue to automate.

Acknowledgment

Many thanks to the Agenticies.es for supporting the collaboration of the authors. This work has been partially funded by the Spanish MCyT under project TIC2002-04146-C05-05, MCYT TIC2003-07369-C02-02, JCYL-2002-05 project SA104A05, the Spanish MEC under project TIN2004-063540-C02-02, and DURSI AGAUR under grant SGR 00296 (AEDS).

Beatriz López
University of Girona,
Campus Montilivi, edifice P4,
17071 Girona,
Spain
e-mail: blopez@eia.udg.es

Susana Fernández
University Carlos III of Madrid,
Avd. Universidad 30, edificio Sabatini,
28021 Leganés, Madrid,
Spain
e-mail: sfarregu@inf.uc3m.es

Javier Bajo
Pontifical University of Salamanca,
Calle Compania 5,
37002 Salamanca,
Spain
e-mail: jbajope@upsa.es

Juan M. Corchado
University of Salamanca,
Plaza de la Merced s/n,
37008 Salamanca,
Spain
e-mail: corchado@usal.es

Raquel Fuentetaja
University Carlos III of Madrid,
Avd. Universidad 30, edificio Sabatini,
28021 Leganés, Madrid,
Spain
e-mail: rfuentet@inf.uc3m.es

Manuel Gonzalez
University Carlos III of Madrid,
Avenida de la Universidad 30, edificio Sabatini,
28911 Leganes (Madrid),
Spain
e-mail: mgbedia@inf.uc3m.es

David Isern
University Rovira i Virgili
Advanced Intelligent Technologies for Knowledge Management Group (ITAKA-URV)
ETSE, Avda. Paisos Catalans, 26
43007 Tarragona (Catalonia)
Spain
e-mail: david.isern@urv.cat

Sergio Jiménez
University Carlos III of Madrid,
Avd. Universidad 30, edificio Sabatini,
28021 Leganés, Madrid,
Spain
e-mail: sjimenez@inf.uc3m.es

Aïda Valls
University Rovira i Virgili
Advanced Intelligent Technologies for Knowledge Management Group (ITAKA-URV)
ETSE, Avda. Paisos Catalans, 26
43007 Tarragona
Spain
e-mail: aida.valls@urv.cat