MAPWEB: COOPERATION BETWEEN PLANNING AGENTS AND WEB AGENTS

David CAMACHO, José MOLINA, Daniel BORRAJO
and Ricardo ALER

1. Introduction

Nowadays, there is a vast (and growing) amount of information stored in the WEB available for any user connected to the network. This information is heterogeneous and distributed. Web information could be used by the users to solve many different problems if only they could spend enough time searching, retrieving, and analyzing the data. Internet provides a lot of WEB applications like search engines and meta-search engines that enable the users to look for the information they need. However, currently it is impractical to build a single and unified system that combines all possible information sources that could be useful to the users. Some of the reasons are summarized below:

- The number of information sources in the web grows exponentially.
- There are a lot of WEB information sources that provide similar information, each one using its own representation of the information. For instance, a traveler might want to find suitable plane companies to get to his/her destination. However, different companies will display on the web similar data but using different representations.
- Different information sources usually provide different kinds of information and it is not always easy to combine them to achieve common goals. For instance, in Non-combatant Evacuation Operations (NEO) domains, it would be useful to combine the information coming from weather forecast sites with the information obtained from Geographical Information System (GIS) servers.
- The information stored could change dynamically over time. A short-term weather forecast site is a good example.

Due to the previously elaborated problems, current WEB search engines basically rely only on purely syntactical textual information retrieval. There are only a few approaches that try to integrate a set of different and specialized sources, but unfortunately it is very difficult to develop and to maintain this kind of systems.[1] Therefore, users cannot use heterogeneous information to obtain satisfactory results in problem solving in a short time and with high quality. It is true that there are many systems that extract, filter and represent efficiently the information obtained from the WEB. However, most of these systems are focused mainly on the amount of information retrieved.[2]

Integrating heterogeneous information is one of the main goals of MAPWEB. However, having complete and high-quality information is not necessarily an end in itself. If the user wants to solve complex problems using that information, the system must include intelligent elements able to reason in complex domains. For instance, a traveler needs to be provided with good plans that combine different means of transportation in an efficient manner. Similarly, NEO and military operations need intelligent systems to move units and supplies on the terrain in a coordinated and efficient way. Artificial Intelligence (AI) provides software components that fill in that gap: planning systems that find good quality plans in complex domains, machine learning systems that learn from experience in these domains, etc. In our work, we apply such AI techniques but in a Multi-Agent System (MAS) framework (a part of what is called Distributed Artificial Intelligence (DAI)). These systems are built using a set of modular components, or *agents,* that are **specialized** in solving a particular aspect of a problem.[3] This decomposition allows each agent to use the most appropriate paradigm for solving its particular problem.[4] Every MAS uses the agent concept, which is extensively described in several publications.[5] The main characteristics of a MAS can be summarized as follows:

- Each agent has an incomplete amount of information or does not have the required abilities to solve the entire problem.
- There is no centralized control.
- Data is not centralized; therefore agents must share their data.
- System execution is asynchronous; an agent can be working and receiving queries simultaneously.
- Each agent has an internal state. It is also able to reason about the environment and possibly learn from experience in order to improve its behavior.

In our work, the agent-based framework provides certain benefits, namely:

- First, multi-agent systems are societies of (usually) heterogeneous software components, using, however, in their communication a common language.

Therefore, MAS address directly the problem of integrating heterogeneous systems, each one handling different kinds of information. This is the problem that complex web information retrieval systems have to face, as has been mentioned before.

- MAS are often used to solve AI problems, such as planning, scheduling, learning, and they have shown their value, due to the fact that:
  - ⇒ Different agents can combine their abilities in a **synergetic** manner. This has been clearly shown in the so called multi-strategy learning systems where different systems provide different characteristics useful to achieve a common goal.[6] However, this is not the only example. For instance, it has been shown that different planners work well in different domains.[7] Therefore, in some cases it would be a good idea to combine different planner agents in the same MAS system.[8]
  - ⇒ They offer modularity, flexibility, and adaptability. A MAS uses a common language and, thus, provides a means for communication between heterogeneous agents. Hence, it is easy to add new agents with new abilities, if required. These characteristics are essential in complex, large or unpredictable domains.[9]
  - ⇒ MAS are inherently parallel, in this way facilitating the efficient execution of the computationally complex problems associated with AI.

However, integrating and coordinating different agents is a complex problem in itself that has to be addressed.[10, 11] When interdependent problems arise, the agents in the system have to cooperate in order to ensure that the interdependencies are properly handled.

This paper presents a distributed multi-agent architecture – MAPWEB – that accepts queries from the users. These queries are actually the problems that have to be solved. As a result the system produces possible solution schemata by means of AI problem-solving techniques (planning and learning), which are then validated and completed using the information available on the WEB.

This paper is divided into seven sections: Section 2 presents a review of the related work in Multi-agent systems; Section 3 describes the MAPWEB architecture; Section 4 analyzes how the system interacts with the user and finds solutions; Section 5 presents an example application domain of the designed system; Section 6 summarizes the conclusions; and finally, Section 7 shows the future lines of work.

2. Related Work

There are several approaches that attempt to work with the information stored on the Web. These approaches focus mainly on the process of retrieval of (usually textual) information, but only few of them try to reason with that information. This section analyzes these systems and illustrates how they handle the stored data. We will focus on the *agent-based* systems (like *Web and Intelligent agents*) and the *multi-agent-based* systems that have been developed and deployed recently.

WEB and Internet applications can be classified in different ways. The following classification focuses mainly on how these systems use the available data:

1. **Simple Web-Applications:** Systems that search, retrieve and store information, like *searchers*, *meta-searchers* or any other popular information retrieval applications. The main goals of these systems are the search and retrieval of WEB information.
2. **Complex Web-Applications:** Systems that transform the obtained information, share with other systems its knowledge, and even could cooperate with other systems to obtain *solutions* that may be useful to the users. These kinds of systems have a wide range of characteristics that attempt to achieve more complex tasks than just retrieving information.

Figure 1 displays a possible classification of the most common WEB applications. Solid lines show that the majority of a given kind of system belongs to the designated class and discontinuous lines show that some applications could be built using only a subset of the characteristics so that intelligence and/or robustness of the systems is increased.

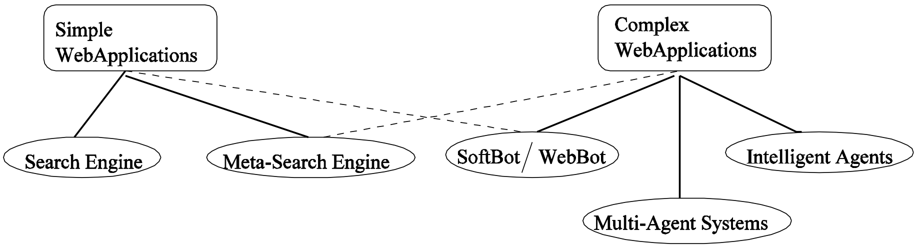


Figure 1: Generic classification of WEB systems.

2.1. *Intelligent Agents*

Intelligent Agents are software entities that assist people and act on their behalf. They make the user's life easier, save his/her time, and provide a simplified view of a complex world. Any *Intelligent Agent* tries to assist, advise or learn from past experience or from other agents' experience to anticipate the requirements of the user. In fact, the agent-based technology is a combination of various technologies including, but not limited to, neural networks, expert systems, fuzzy logic, machine learning, planning.[12]

It is a difficult task to characterize accurately what is an agent. There is extensive literature on the topic.[13,14] The following features can be used to characterize an intelligent agent:

- Agents are *pro-active* in nature (although they can and will be reactive as well).[15]
- Agents can *learn* as a result of their action – not to mention from their mistakes.[16,17]
- Agents can be *predictive* in nature.[18] [19]
- Other key attributes of the paradigm include *autonomy*, *security*, *personality*, and *mobility*.[20,] [21] However, an agent need not possess all these characteristics. The fact that an agent can move from one environment to another is not a requirement in all cases.
- Lastly, agents are *social* in nature. They can collaborate with other agents as well as delegate tasks to subordinate or "better suited for the job" agents.[22]

Different and successful intelligent agents have been developed recently. In the following, some of these agents will be briefly described:

- **Softbot.** This agent interacts with a software environment by using and interpreting the environment feedback. The softbot effectors are UNIX commands that enable the agent to change the state of the user environment.[23]
- **SodaBot** is a general-purpose software agent user environment and construction system. Its main component is the *basic software agent* that is a computational framework for building agents; it is essentially an *agent operating system*. Through the definition of a new programming language it is possible for the users to implement a wide-range of typical software agent applications, such as on-line assistants or meeting-scheduling agents.[24]
- **SIMS** and **ARIADNE.** These intelligent information agents are focused mainly on information retrieval and integration of different kind of information sources. SIMS focuses on the integration of well-structured databases,[25] while the ARIADNE project deals with accessing information from more loosely structured Web sources.[26]

2.2. WebAgents

Currently there is an enormous number of Web applications that offer different services to Internet users, such as search and meta-search engines (*Lycos*, *Altavista*, *Yahoo*), e-commerce markets, auctions, web directories, etc.[27] As we have said in Section 1, due to the current evolution of the WEB (and other on-line information sources), it has become a necessity to provide some sort of intelligent assistance to the users. WebAgents are applications that are able to consult the best Internet sites and perform agent specific tasks, such as retrieving, processing, tracking and submitting required information. WebAgents perform specific Internet tasks. From this point of view, the functionality of WebAgents is given by the agents installed on the system and their specific purpose. There is a lot of research and development on this kind of systems. Here we only briefly mention some of the developments:

- **MetaCrawler.** The METACRAWLER SOFTBOT is a parallel WEB search service that provides unified interface by which any user can query popular general-purpose WEB search engines, such as *Lycos* or *Altavista*. METACRAWLER has some characteristics that enables it to obtain results of higher quality than simply showing the output from the search service.[28]
- **Letizia** is a *user interface agent* that assists users browsing the World Wide Web. While the user operates a conventional Web browser, the agent tracks user behavior and attempts to anticipate items of interest by doing concurrent, autonomous exploration of links from the user's current position. The agent automates a browsing strategy consisting of best-first search augmented by heuristics derived from inferring user interest from its browsing behavior.[29]
- **WebWatcher** is a "tour guide" agent for the World Wide Web. Once a user enters to WebWatcher what kind of information he/she looks for, it accompanies the user from page to page. While the user browses the web, it highlights hyper-links that it believes could be of interest. Its strategy to give advice is learned from the feedback from earlier tours. Currently, WebWatcher is online only on an irregular basis.[30]
- **WebPlan.** This intelligent WEB agent has been developed at Kaiserslautern Universtity. WEBPLAN is a search assistant for domain-specific search on the Internet based on dynamic planning and plan execution techniques.[31]

2.3. Multi-Agent Systems

Due to the growing importance of agent-based technologies in the development of software systems, there are several commercial and research agent development toolkits. It is very difficult to select an appropriate toolkit, as each toolkit has been designed for a certain *architecture* or *paradigm*. We will only examine several popular toolkits and deployed MAS.

- **AgentBuilder.** This is a very popular commercial toolkit for building and testing agent-based software. Agents constructed using AgentBuilder communicate using KQML (Knowledge Query and Manipulation Language).[32] It makes it possible to develop and extend the standard KQML performatives (or messages) to include additional performatives.[33]

- **JAFMAS.** This toolkit provides a framework that helps developers to structure their ideas into specific agent applications. It directs the development from a speech-act perspective and supports multicast and directed communication, KQML or other speech-act performatives. It also performs some analysis on the multi-agent system coherency and consistency.[34]
- **JADE** (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems and applications, conforming to FIPA standard for intelligent agents.[35] JADE can be considered as an *agent middle-ware* that implements an *Agent Platform* and a development framework.[36]
- **JATLite** is a framework for creating multi-agent systems. JATLite includes a message router (*agent message router* or simply *AMR* agent) that supports message buffering, allowing agents to fail and recover. Agents can send and receive messages using KQML. Message buffering also supports a name-and-password mechanism that enables agents to move freely between hosts.[37]
- **KASBAH** is a virtual market place on the WEB where users can create autonomous agents that buy and sell goods on their behalf. Users can specify parameters to guide and constrain the agent's overall behavior. Any intelligent agent in KASBAH is an object (an instance of a class) and the market place allows the user to create buying and selling agents, which then interact in the market with other agents. The agents themselves are not very smart, although they are completely autonomous. Agents do not use AI or Machine Learning techniques. The interesting aspect of KASBAH is its multi-agency. It is a good framework for testing different important characteristics of this kind of systems, such as *negotiation*.[38]
- **MPA.** The Multiagent Planning Architecture is a framework for integrating diverse technologies into a system capable of solving complex planning problems. MPA has been designed for application to planning problems that cannot be solved by *individual systems,* but rather require the coordinated effort of a diverse set of technologies and human experts.[39]
- **CMUEXPRESS** is a MAS architecture developed at Carnegie Mellon University (CMU). Its purpose is to plan, execute plans, and monitor its performance. It has been applied to Non-combatant Evacuation Operations (NEO). In this specific case, the entire system integrates about twenty agents.

In particular, it includes MMM (a user interface developed at Stanford Research Institute- SRI), ARIADNE (described above), and the already mentioned CMUEXPRESS. The goal is to locate, pick up, and carry civilians to a safe place. The agents collaborate in the following manner. First, ARIADNE locates the civilians. Then, CMUEXPRESS provides routing plans to transport them, in addition to monitoring the on-going plan and reacting to events. CMUEXPRESS can use the tracking information provided by ARIADNE, that is obtained from an on-line web-site.[40]

- Finally, there is a hierarchical multi-agent system developed at DERA (UK) to plan military activities (i.e., moving troops on a terrain) and execute them. Its aim is to combine deliberative and reactive behavior. The agents in the society are organized in a hierarchical military manner. For instance, there is a Squadron Commander agent, a Troop Commander agent, a Tank agent, etc. This framework enables the more reactive behaviors of the agents at lower levels of the hierarchy to be guided by the more deliberative planning of the agents above them in the hierarchy. In particular, a constraint planner (deliberative) and an anytime planner (reactive) are combined within the hierarchy.[41]

3. MAPWEB: A Multi-Agent Architecture for Reasoning on the Web

As already mentioned, the main advantage of using MAS techniques is the flexibility and adaptability of the resulting system. A MAS could consist of several heterogeneous elements. These elements, or *agents*, can play different programmed roles, could execute different functions, and could modify their behavior dynamically.

MAPWEB is a MAS approach that integrates heterogeneous agents. These agents assemble a set of *"logic-layers"* between the users and the WEB. The architecture *hides* the WEB from the users. This facilitates the user in coping with the overload of information. Figure 2 illustrates the four-layered architecture of MAPWEB.

1. **Physical World:** it represents the users.
2. **Reasoning Layer:** this layer connects any physical agent (usually human) with a set of systems that allows the agents to access the desired information.
3. **Accessing Information Layer:** this layer retrieves the information from distributed sources (like the WEB) and represents it in an understandable fashion to the previous layer.
4. **Information World:** it represents all the information available on networks, computers, or any other kind of electronic support. This *"world"* is accessible only through information retrieval systems.

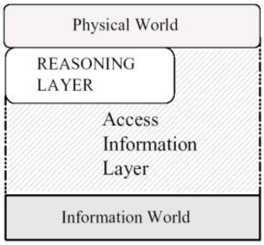


Figure 2: World/Web Layers.

How MAPWEB implements the above described multi-layer architecture, can be seen in Figure 3. The system is composed of a set of agents that can communicate, share knowledge and cooperate in order to find solutions to the problems posed by users.

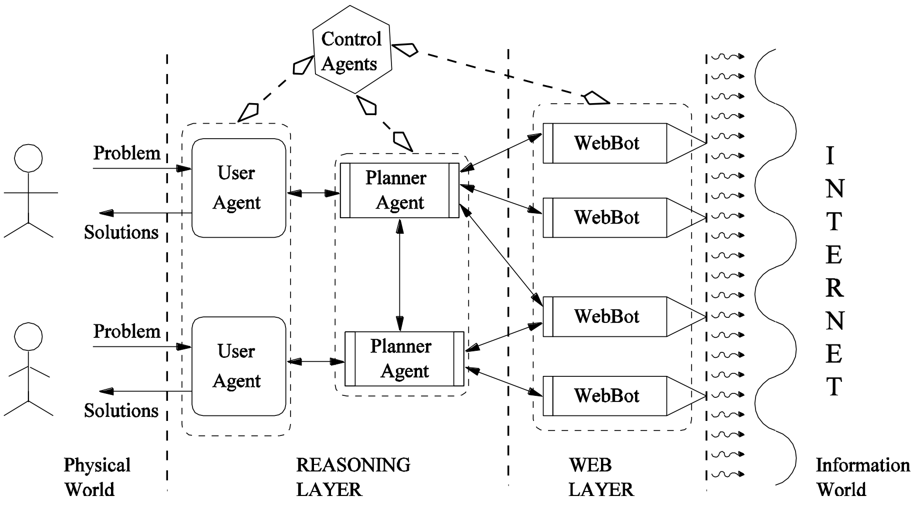


Figure 3: MAPWeb general Architecture.

This architecture has been designed to deal with some frequent problems existing on the WEB. In order to accomplish this, it is necessary to use an internal knowledge representation shared by the agents, and different reasoning techniques that enable the agents to look for new solutions. MAPWEB is a MAS approach that integrates different heterogeneous agents with diverse roles into the agent society. The types of agents used can be summarized into the following categories:

- **UserAgent:** this agent connects the *physical world* with the *reasoning layer*.

It takes user queries and displays to the user the solution(s) found by the system. UserAgents capture problem queries from the users and send them further to a reasoner-agent. PlannerAgent is the only currently developed reasoner-agent, but various kinds of reasoner-agents, such as LearningAgents, will be developed in the future. Afterwards, the reasoner-agents are responsible for finding solutions to the problem.

- **ControlAgents:** These agents belong to the *reasoning layer* and considering the organizational structure of the system, there are two different types of control-agents in MAPWEB: ***ManagerAgent*** and ***CoachAgents***. Their main roles are summarized below:
  - ⇒ **ManagerAgent:** It directs the insertion and deletion of agents from the system. This agent is responsible for building dynamic *teams* of agents specialized in different problem solving activities.
  - ⇒ **CoachAgent:** This agent controls a set of heterogeneous agents that represent a *team*, which accepts problems from any agent (software or human) in the system and attempts to solve them.

Figure 4 illustrates the relationship between these kinds of agents and the rest of the agents in MAPWEB. Agents are organized in teams, each one is managed by a coach. The whole system is leaded by a manager. Each UserAgent, PlannerAgent or WebAgent might belong to several teams if necessary for the proper work of the team.

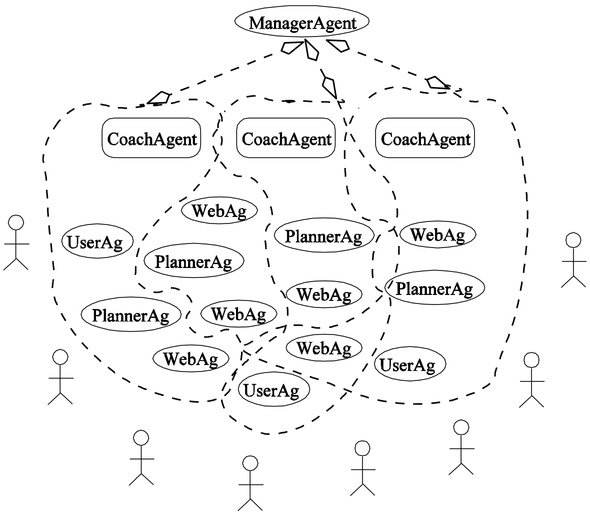


Figure 4: Manager and Coach Agents Organization.

- **PlannerAgent:** This agent (belonging to the *reasoning layer*) receives a planning problem, builds an abstract representation of it, and solves it. PlannerAgents have different abilities, such as communication and planning.
- **WebAgent:** These agents belong to the *accessing information layer* and connect the *reasoning layer* with the *information world*. Its main goal is to complete the details of the abstract plans obtained by the PlannerAgents. It receives that information from the WEB.

Some of the underlying modules (see Figure 5) of any MAPWEB-agent are:[42]

1. *Control module:* it manages all possible tasks performed by the agents. This module is basically made of an agenda, some policies, and a set of specialized skills.
2. *Knowledge module:* this module is used by the different agents to store their own knowledge.
3. *Skills module:* this module implements the specialized skills of any agent in the system.
4. *Communication module:* it implements the communication protocol with other system agents (UserAgents, PlannerAgents, CoachAgents, or WebAgents). This module is implemented using two sub-modules:
   - *Transport module:* it implements a TCP/IP network-level communication between two agents running on different computers.
   - *Language module:* it implements a standard version of KQML[43] that makes it possible to use a common language between two agents in MAPWEB.

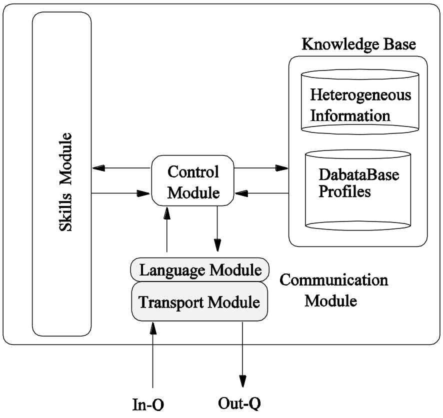


Figure 5: Skeleton-Agent in MAPWEB.

The following subsections give a more detailed description of the different agents: roles, architectures, and organization.

3.1. UserAgents

The main role of UserAgents is to connect the users with MAPWEB. Each UserAgent uses a set of Graphical User Interfaces (GUI) to communicate with the users and an implementation of the standard language KQML to communicate with other agents in the system. Figure 6 presents a modular description of the UserAgent architecture.

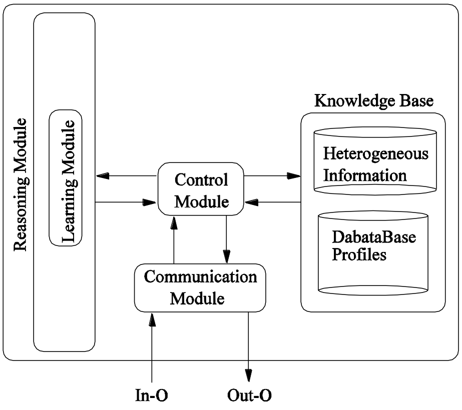


Figure 6: UserAgent Architecture.

The *Knowledge Module* is used by the UserAgent to store a set of different user profiles and successful old solutions, that can be used by UserAgent (applying its learning skills in the *Learning Module*) to analyze and customize the system.[44]

The main goals of a UserAgent are:

- To accept problems from users and to present the solutions found by MAPWEB.
- To analyze the problems and to obtain homogeneous representation for them.
- To communicate with PlannerAgents in order to request solutions.

For the accomplishment of the previously described goals, it is necessary to provide, for each particular domain, the specific set of GUIs that can represent all the necessary input/output information for communication with the external world, and to define an ontology that allows the other agents in the system to know the type of problem that has to be solved. Section 5 will present the set of GUIs for a considered domain.

3.2. PlannerAgents

Any PlannerAgent has a modular architecture where each module has its own capabilities and tasks. These are the reasoning agents in the system. Figure 7 depicts the PlannerAgent's modular representation.

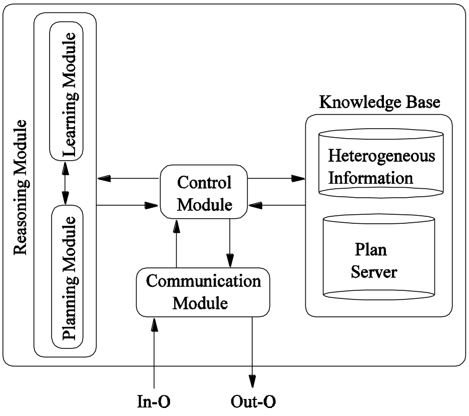


Figure 7: PlannerAgent Architecture.

Some of the most interesting characteristics of PlannerAgent are:

- *Communication module:* it implements a subset of specific *performatives* (speech-acts in KQML) used by PlannerAgents to share plans or sub-plans.
- *Knowledge module:* Stores useful information for the agents. It is composed of two main sub-modules:
  - ⇒ *Heterogeneous Information:* This sub-module stores useful data (heterogeneous information) about the application domain, planning operators, heuristics, information about other agent characteristics, statistics information, etc.
  - ⇒ *Plan server:* This module stores old plans or sub-plans that can be used in finding new solutions.[45]
- *Control module:* it is used to manage the various agent modules. Some of its main functions are:
  - ⇒ To handle abstract solutions; they should be validated using the information acquired from other agents, or from other heterogeneous information sources.
  - ⇒ To build an agenda that handles its own tasks and the questions posed by other agents.

- ⇒ To deal with all possible answers given to questions asked by other agents and/or users.
- *Reasoning module:* It is mainly comprised of two sub-modules:
  - ⇒ *Learning modules:* They can modify the system behavior if the obtained solutions are successful in solving user problems. Currently, a Case-Base Planning Module is being developed, and it is used to gain efficiency in the planning process by retrieving and adapting past stored solutions; it avoids performing the planning process.[46]
  - ⇒ *Planning module:* it performs the actions necessary to solve the user problem. Currently, the planning module uses the non-linear planner PRODIGY4.0.[47]

The PlannerAgents use a planner as main reasoning module. The agent generates an abstract representation of the problem and the specific user queries (given by the UserAgent). Then, it uses a planner to obtain a very abstract solution (or solutions) of the problem, and finally cooperates with the WebAgents to fill in the details of these abstract solutions.

3.3. ControlAgents

As previously described, there are two different types of Control agents in MAPWEB. They have identical architecture (see Figure 8) but different roles.

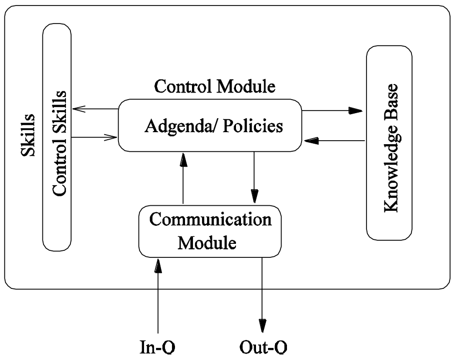


Figure 8: Generic ControlAgent Architecture.

We could outline the differences as follows:

1. ManagerAgent:
   - There is only one ManagerAgent.

- It is responsible to add and remove other agents from the system.
- It controls which agents are active in the agent society.
- It groups agents in teams.
- It determines which are the agents shared by the different teams.

2. CoachAgent:
    - It controls a team of agents, guaranteeing stability and smooth operation of the active agents.
    - It reports problems to the ManagerAgent. For instance, when a new agent is required for the team.
    - It guarantees that the agendas of the team members are coherent.

To function correctly, MAPWEB (for any possible multi-agent topology) needs at least one Manager and one Coach agents to build teams that will be able to reason about the user problems.

3.4. WebAgent

The *WebAgents*, like the other system agents, have their own modular architecture (it is shown in Figure 9). A WebAgent handles (CONTROL MODULE) the questions received from other agents (PlannerAgents), and translates these questions into queries to the WEB (INTERNET ACCESS MODULE). The answers from the WEB will be filtered and stored in a data base (DATABASE FROM WEB). This useful information will be sent later to the PlannerAgent. WebAgents know various places where to look for the requested information.

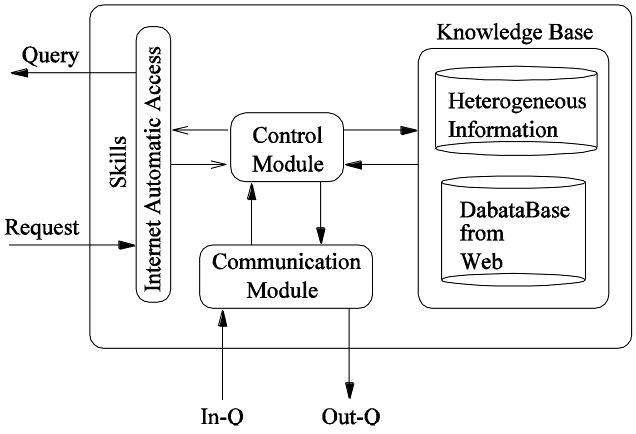


Figure 9: WebAgent Architecture.

Although MAPWEB has a very general architecture and it is possible to apply MAPWEB to different domains, the paper presents an implementation of a set of WebAgents specialized in the task of retrieving, filtering, and representing the necessary information from the WEB for a specific domain (see Section 5).

4. Problem Solving and Cooperation in MAPWEB

MAPWEB has an architecture where different agents have to cooperate in order to reach a solution. Different agents need to share their knowledge and skills to complete the abstract solutions obtained by the PlannerAgent. MAPWEB's success depends on the following factors: sharing knowledge to obtain new solutions and using different Web and reasoning skills by the MAPWEB agents to find useful solutions for the users. In what follows the format for sharing and communicating knowledge, and the generic cooperative-solving process in MAPWEB, are analyzed.

4.1. Sharing Information between Agents

Agents in MAPWEB use a common representation for the knowledge. This characteristic facilitates the process of sharing and reasoning with the knowledge. Agents use **performatives** in their communication. Any performative contains an implicit order to another agent. For communication between system agents, a subset of the KQML format is currently being used.[48] This format is shown in Table 1.

Table 1: Some performatives in MAPWEB.

Performative	**Format**
achieve	(:content (FLY Company MAD ZAZ…) :language JAVA :ontology Electronic-Tourism :in-reply-to MAPWEB :sender PAgent1 :receiver WBot1)
tell	(:content (FLY IBERIA 323 Price…) :language JAVA :ontology Electronic-Tourism :in-reply-to MAPWEB :sender WBot1 :receiver PAgent1)

This example illustrates the representation of two performatives: ACHIEVE and TELL. The first performative (ACHIEVE) is sent by a PlannerAgent (*PAgent1*) to a WebAgent

(*WBot1*) asking for WEB information the WebAgent is specialized in. The second performative (TELL) is the reply from the WebAgent to the PlannerAgent; it stores the information retrieved by the agent *WBot1*.

There exist other KQML performatives implemented by MAPWEB agents to manage the group of agents and to allow agent negotiation, such as ACCEPT, REJECT, REGISTER, UNREGISTER, DELETE, INSERT.

4.2. Cooperation in MAPWeb

This section describes how UserAgents, PlannerAgents, and WebAgents cooperate to solve problems. From a generic point of view, a problem is a pair (***initial situation***, ***final situation***). An example of a problem is the following: a person intends to fix (final situation) a broken car (initial situation). A solution to a problem is the sequence of actions to be performed so as to get from the initial to the final situation (called a `plan`). Usually, actions are defined in terms of `operators`. For instance, `screw(x)` could be an operator denoting the use of the screwdriver on any screw `x`. Therefore, a solution to the car fixing problem could be anything like the plan presented in Figure 10.

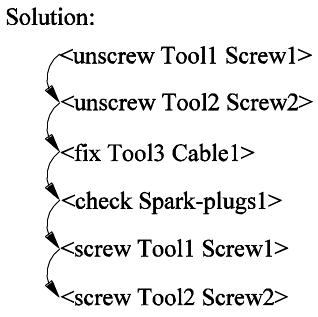


Figure 10: A Possible *Car Fixing* Plan.

A set of problems that use the same operators is called a `domain`. The goal of MAPWEB is to give solutions to problems in a domain as just defined.

The sequence MAPWEB follows to solve a problem is like this:

1. The user interacts with a UserAgent to define his/her problem. Then the UserAgent sends to a PlannerAgent an ACHIEVE performative containing the problem definition.

2. The PlannerAgent receives the problem definition and analyzes it. Usually, a user problem contains a lot of detail and that makes problem solving computationally very expensive for classical AI planning systems. For that reason, before attempting to solve it, the PlannerAgent discards some of the detail and transforms the user problem into an abstract representation. For instance, in the car fixing domain, there could be many different kinds of parts and tools to deal with them. In that case, the PlannerAgent would reduce the number of different parts and tools to a manageable quantity. Then, the user problem would be transformed into an abstract representation that uses only the reduced set of parts and tools. At this point, the PlannerAgent would use a planning system to solve the abstract problem and get several possible abstract solutions. However, the user needs all the details to be able to apply the plan. Furthermore, many of the abstract solutions might not be valid in reality since they ignore actual details. Therefore, the abstract plans have to be completed and validated. The PlannerAgent analyzes which parts of the abstract plans require completion, and asks for details the WebAgents.
3. WebAgent receives PlannerAgent's queries for details, looks for information at those web-sites the agent is specialized in, and returns the information to the PlannerAgent in a common shared format. If it cannot find the requested information, the PlannerAgent will be informed, and it will discard all the plans that include the invalid operator. For instance, different car companies could maintain web-sites with information on technical characteristics of cars, tools, and parts, which could be used by the specialized WebAgents to fill in the requested details. If the WebAgents could not find information for validating the `fixing` step, because, for instance, there are no `Tools` to handle `Cable1`, all the plans that include this step will be discarded by the PlannerAgent.
4. Finally, the PlannerAgent receives a TELL performative from several WebAgents, validates and completes the abstract plans, and returns complete plans to the UserAgent. In our example, a possible complete solution would include which actions to perform and the specific tools and parts to use. This plan could be utilized directly by the user.

5. Illustrative Application of MAPWEB

In principle, MAPWEB can be applied to many and diverse problem solving domains. In this section, we describe how MAPWEB has been applied to a particular domain – **"electronic tourism"** (or simply *e-tourism*) – and how the different agents cooperate to solve problems in this domain. Earlier versions of MAPWEB have been described by Camacho and coauthors.[49] This section will first present the e-tourism domain (i.e., how solutions are represented) and then how the different agents in MAPWEB cooperate to provide solutions to the user. Communication between the UserAgent, the PlannerAgent, and the WebAgents will be elaborated in detail.

5.1. *Electronic Tourism Domain*

An e-tourism system has to provide the following services to the user:

1. Informing how to go from the initial to the destination town using different means of transportation.
2. Lodging at destination.
3. Informing about possibilities when visiting a town (renting a car, local transport, etc.).
4. Informing how to return to the initial (or other) town.

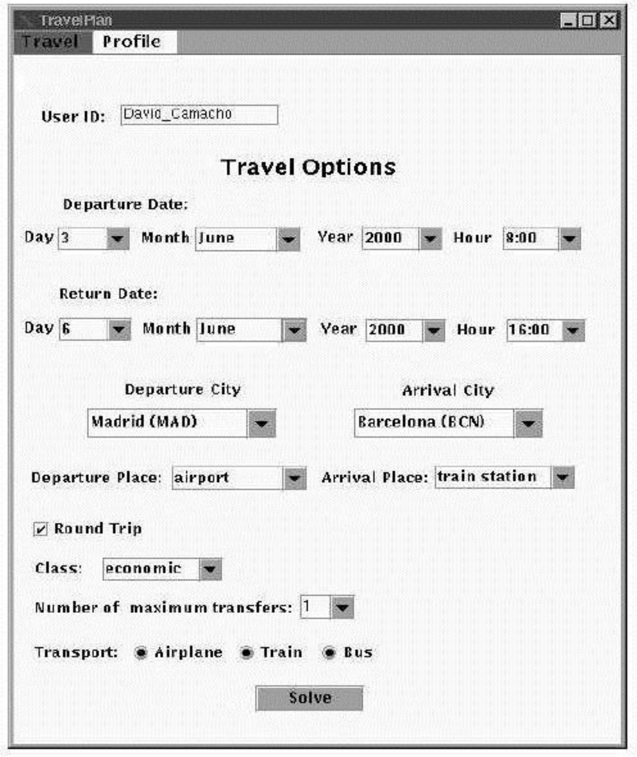
MAPWEB has the abilities enumerated above. However, in this paper, we will focus mainly on the logistics problem of providing the user with plans to move from one place to another. Moving from place to place involves long-range trips that can be accomplished via airplanes, trains, or buses. It also involves taking local transportation means (taxi, subway, bus, etc.) to move between airports, bus stations, or train stations. In order to represent and provide solutions to the user, we have defined an e-tourism domain that uses the operators illustrated in Table 2.

Table 2: E-tourism planning operators

Operator	**Arguments**
TRAVEL-BY-AIRPLANE	User-name, Company, Origin-airport, Destination-airport
TRAVEL-BY-TRAIN	User-name, Company, Origin, Destination
TRAVEL-BY-BUS	User-name, Company, Origin, Destination
MOVE-BY-LOCALBUS	Origin, Destination
MOVE-BY-TAXI	Origin, Destination
MOVE-TO	Origin, Destination
BOOK-HOTEL-ROOM	User-name, Hotel, City
…	

5.2. *UserAgent → PlannerAgent Communication*

The UserAgent provides a GUI to the user, so that s/he can describe the problem and the restrictions associated with it. Obviously, GUIs depend heavily on the problem domain: other domains would require other GUIs. Figure 11 presents the *input*-GUI to the system.



User ID: David_Camacho

Travel Options

Departure Date:

Day 3 Month June Year 2000 Hour 8:00

Return Date:

Day 6 Month June Year 2000 Hour 16:00

Departure City: Madrid (MAD)
Arrival City: Barcelona (BCN)

Departure Place: airport
Arrival Place: train station

[✓] Round Trip

Class: economic

Number of maximum transfers: 1

Transport: ● Airplane ● Train ● Bus

Figure 11: User Agent Input.

The data the user has to provide to the system is as follows:

- Departure and return dates
- Departure and arrival cities
- Starting and arrival places inside the cities (airport, train station, bus station, etc.)
- One-way or return trip
- Maximum number of transfers
- Cost (economy class, business class, first class, second class, tourist class, etc.)
- Long-range transport (airplane, train, or bus)

In the example given in Figure 11, the user plans to travel to Barcelona (Spain) from Madrid (Spain) on the 3rd of June at 8 o'clock. The return date is the 6th of June at 4 or later. The user would like to start his/her trip from an airport and wishes to end it at a train station in Barcelona. S/he wants to minimize cost and s/he does not specify the long-range transportation means. Also, s/he does not want to transfer more than once.

Once the UserAgent has received the problem, it sends an ACHIEVE performative to a PlannerAgent and waits for the solution.

5.3. PlannerAgent → WebAgents Cooperation

The PlannerAgent receives from the UserAgent a problem and proceeds with building an abstract representation that retains only the parts essential for the planning process. For instance, a typical description of the previous problem for an AI planning system would include:

- All the cities in the world
- All the airports, train stations, etc. inside those cities
- All the plane, bus, and train companies in the world
- All local transportation means (taxi, subway, etc.) in the cities

Any classical AI planning system would get bogged down if it tries to find a plan by considering all these elements. Instead, the PlannerAgent builds an abstract problem in the following way:

1. First, it defines an abstract city. This city includes all the possible local transport and only the long-range transport terminals that the user wishes to use. For instance, if the user wants to travel only by plane, the abstract city would include just airports. The goal is to reduce the number of elements in the problem, so that the planner can handle them more efficiently. In the previous example, as there are no restrictions on the long-range transport, the abstract city would have airports, bus stations, and train stations.
2. Then, this abstract city is repeated as many times as is the maximum number of transfers supplied by the user. It is important to note that the cities are abstract cities (i.e. they have no attached names; they are present in the abstract plan to represent the initial, middle, and final travel points).
3. Finally, the rest of the details provided by the user are ignored at this stage. For example, departure and arrival times, travel cost, etc. is not considered. This data will be used later to query the WebAgents and validate the abstract solutions.

As an illustration, from the problem given by the UserAgent, the PlannerAgent would construct a planning problem that includes three unnamed cities: `city0`, `city1`, and `city2`. `city0` is the departure city, `city2` is the destination, and `city1` is a (possible) transfer city. Each of the cities includes all possible local transportation means, abstract locations (`hotel1`, …) and terminals (`airport0`, `trainstation0`, ...). Finally, the planning problem would include an initial situation of the user being at `airport0` in `city0`, and the goal situation is that of the user being at `trainstation2` in `city2`.

The above described abstract problem would be given to the PlannerAgent planner (Prodigy4.0) which would obtain several possible abstract solutions. In this case, the planner would reply with the plans given in Figure 12.

```
Solution 1:
 <move-to trainstation0 bustop01>
 <move-to bustop01 airport0>
 <travel-by-airplane user1 plane0 airport0 airport1>
 <move-to airport1 bustop11>
 <move-by-localbus bustop11 bustop12>
 <move-to bustop12 trainstation1>
 <travel-by-train user1 train1 trainstat1 trainstat2>


Solution 2:
 <move-to trainstation0 bustop01>
 <move-by-localbus bustop01 bustop02>
 <move-to bustop02 airport0>
 <travel-by-airplane user1 plane0 airport0 airport2>
 <move-to airport2 bustop21>
 <move-by-localbus bustop21 bustop22>
 <move-to bustop22 trainstat2>
```

Figure 12: Two abstract solutions generated by Prodigy for the travel problem.

This is a set of abstract plans that contain no details. Some of the steps in the plan might not even exist in the real world. Therefore, these plans need to be validated and completed. This is achieved by querying the WebAgents. In this case, the following query schemas would be generated:

```
Queries:
(travel-by-airplane user plane0? Madrid city1?)
(travel-by-train user train1? city1? Barcelona)
```

The queries above have some uninstantiated variables (`plane0?`, `train1?`, and `city1?`). The variable `city1?` will be instantiated by the PlannerAgent before querying the WebAgents. The PlannerAgent will choose several actual cities by using some heuristics. For every selected city, an actual query will be generated. For instance, the first query schema would be translated into:

```
Queries:
(travel-by-airplane user plane0? Madrid Valencia)
(travel-by-airplane user plane0? Madrid Alicante)
```

These queries (and all the additional information given by the UserAgent) are sent to several WebAgents that are specialized in airplane travel, so that variable `plane0?` is instantiated as well.

A WebAgent receives a query and associated with it data and transforms it into an actual web query. The WebAgent is familiar with the structure of the data stored at the web sites it is specialized in, and it knows how to look for information in these web sources. The retrieved information is then analyzed and stored in a common template, which is subsequently sent to the PlannerAgent. In our example, the information in Table 3 would be returned to instantiate the variable `plane0?`. Actually, that variable can be instantiated in many different ways, as many as the possible flights from Madrid to Valencia.

Table 3: Retrieved WebAgent Information.

Information-Flights	flight1	flight2	flight3
air-company	Iberia	Iberia	Spanair
http-address	www.iberia.es	www.iberia.es	www.spanair.com
flight-id	323	450	null
ticket-fare	38200	21850	43700
currency	ESP	ESP	ESP
flight-duration	Null	null	null
airport-departure-city	MAD	MAD	MAD
departure-date	03-06-00	03-06-00	03-06-00
airport-arrival-city	VLC	VLC	VLC
return-date	06-06-00	06-06-00	06-06-00
class	D	D	null
number-of-passengers	1	1	1
round-trip	one-way	one-way	one-way

Finally, the PlannerAgent instantiates all the abstract plans for which it has received from the WebAgents a positive answer for each plan step. Those plans in which one or several steps received either no answer or an empty answer are rejected. Therefore, only plausible plans are sent back to the UserAgent. Every abstract plan will be instantiated into many different actual plans. Table 4 shows two of the generated solutions.

Table 4: Solutions given by MAPWeb.

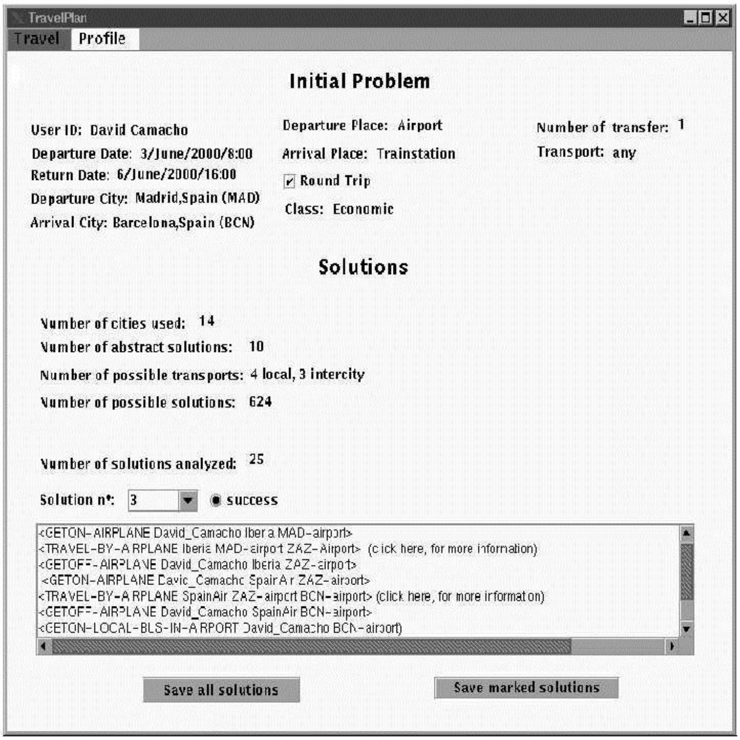
Solution1	**Solution2**
(move-to trainstation0 bustop01)	(move-to trainstation0 bustop01)
(move-to bustop01 MAD)	(move-by-localbus bustop01 bustop02)
(travel-by-airplane SMejias Iberia MAD VLC)	(move-to bustop02 MAD)
(move-to VLC bustop11)	(travel-by-airplane SMejias plane0 MAD BCN)
(move-by-localbus bustop11 bustop12)	(move-to BCN bustop21)
(move-to bustop11 VLCtrainstation1)	(move-by-localbus bustop21 bustop22)
(travel-by-train Smejias Talgo VLC BCN)	(move-to bustop22 hotel2)
(move-to BCN bustop21)	
(move-by-localbus bustop21 bustop22)	
(move-to bustop22 hotel2)	

5.4. *PlannerAgent → UserAgent Communication*

Finally, the UserAgent receives the list of actual plans and presents them to the user. Figure 13 shows the *output*-GUI where the found plans for our problem are displayed. If the user wants more information about a plan step, s/he can click on the corresponding operator and get data about departure time, location, etc.

6. Conclusions

We have presented a multi-agent approach (MAPWEB) to solve planning problems using the information available on the WEB. In particular, this paper focuses on how to solve user planning problems by means of cooperation between a PlannerAgent and several WebAgents. This cooperation amounts to dividing the planning problem into two parts: generation of abstract plans (by the PlannerAgent) and validation-completion of these plans (by the WebAgents). This is done since planning problems contain a lot of details that makes the classical AI problem solving computationally very expensive.



Initial Problem

User ID: David Camacho
Departure Date: 3/June/2000/8:00
Return Date: 6/June/2000/16:00
Departure City: Madrid,Spain (MAD)
Arrival City: Barcelona,Spain (BCN)
Departure Place: Airport
Arrival Place: Trainstation
[✓ Round Trip]
Class: Economic
Number of transfer: 1
Transport: any

Solutions

Number of cities used: 14
Number of abstract solutions: 10
Number of possible transports: 4 local, 3 intercity
Number of possible solutions: 624

Number of solutions analyzed: 25

Solution nº: 3 ● success

```
<GETON-AIRPLANE David_Camacho Iberia MAD-airport>
<TRAVEL-BY-AIRPLANE Iberia MAD-airport ZAZ-Airport>  (click here, for more information)
<GETOFF-AIRPLANE David_Camacho Iberia ZAZ-airport>
 <GETON-AIRPLANE David_Camacho SpainAir ZAZ-airport>
<TRAVEL-BY-AIRPLANE SpainAir ZAZ-airport BCN-airport> (click here, for more information)
<GETOFF-AIRPLANE David_Camacho SpainAir BCN-airport>
<GETON-LOCAL-BUS-IN-AIRPORT David_Camacho BCN-airport)
```

Figure 13: UserAgent Output.

Therefore, before attempting to solve a planning problem, the PlannerAgent discards some of the details and builds an abstract, easier to solve, version. Then, several abstract solutions are obtained. However, many of the abstract solutions might not be valid in reality due to the fact that some of the actual details are ignored. Therefore, the abstract plans have to be completed and validated.

There is another important reason to divide the planning process. Information on the WEB is heterogeneous and is provided in multiple formats. Therefore, it makes sense to have many different agents specialized in each information source or web site. Thus, WebAgents not only free PlannerAgents from the details, they also isolate them from the complexity of the information sources.

MAPWEB is not only a set of conceptual ideas. The described architecture has been implemented. Also, it has been applied to an actual domain (e-tourism) where the cooperation characteristics described above are fully exploited.

7. Future Directions

Some of the lines of future work include:

- Cooperation between several PlannerAgents. In many planning domains, a problem can be divided into a set of sub-problems. Each sub-problem could be sent to a different PlannerAgent. This would be useful for two reasons. First, problem solving can be parallelized. And, second, different kinds of sub-problems could be sent to specialized PlannerAgents that might use different planning techniques.
- Reuse of information stored in both PlannerAgents and WebAgents. Agents can learn from experience. For instance, if a PlanningAgent has previously solved a problem, it can be stored in an internal database for later use, either by the same agent or by others. In a similar manner, a WebAgent can reuse information retrieved previously to reduce the WEB access.
- Application of Case-Based Reasoning techniques,[50] so that new planning problems can be solved by adapting the plans from previously solved similar problems. This would reduce enormously the planning process which is computationally very expensive.
- Finally, in order not to overload the user with too many plans, MAPWEB should be able to rank the solutions and recommend the best ones using user profiles and by learning from user's previous behavior.

Notes:

[1] Jose Luis Ambite and Craig A. Knoblock, “Agents for information gathering,” in *IEEE Expert: Intelligent Systems and their Applications* (September/October 1997); Craig A. Knoblock and José Luis Ambite, “Agents for Information Gathering,” in *Software Agents,* ed. J. Bradshaw (Menlo Park, CA: AAAI/MIT Press, 1997).

[2] Oren Etzioni, “Moving up the information food chain,” *AI Magazine* 18, 2 (1997): 11-18.

[3] Michael N. Hunhs and Munindar P. Singh, *Readings in Agents* (San Francisco: Morgan Kaufmann, 1997).

[4] J. Bradshaw, ed., *Software Agents* (Menlo Park California: AAAI Press, 1997); Michael R. Genessereth and Steven P. Ketchpel, “Software agents,” *Communications of the ACM* 37, 7 (1994): 48-53.

[5] W. Brenner, R. Zarnekow, and H. Wittig, *Intelligent Software Agents. Foundations and Applications*, (New York, Springer-Verlag, 1998); Michael Wooldridge and Nicholas R. Jennings, “Intelligence agents: Theory and practice,” *Knowledge Engineering Review* (October 1994).

[6] Ricardo Aler, Daniel Borrajo, and Pedro Isasi, “Genetic programming and deductive-inductive learning: A multistrategy approach,” in *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, ed. Jude Shavlik (Madison, Wisconsin, July 1998), pp. 10-18.

[7] Manuela M. Veloso and Jim Blythe, “Linkability: Examining causal link commitments in partial-order planning,” in *Proceedings of the Second International Conference on AI Planning Systems* (Chicago, IL: AAAI Press, June 1994), pp. 170-175.

[8] D.E. Wilkins and D.L. Myers, “Multiagent planning architecture,” in *Proceedings on The Fourth International Conference on Artificial Intelligence Planning Systems AIPS98* (June 1998).

[9] Katya P. Sycara, “Multiagent systems,” *AI Magazine* 18, 2 (1998).

[10] Alan H. Bond and Less Gasser, *Readings in Distributed Artificial Intelligence* (San Francisco California: Morgan Kaufmann, 1988); Les Gasser, “An Overview of Distributed Artificial Intelligence,” *Distributed Artificial Intelligence: Theory and Praxis* (Kluwer Academic Publishers, 1992), pp. 9-30.

[11] Nicholas R. Jennings, *Coordination Techniques for Distributed Artificial Intelligence* (O'Hare et al., 1996), pp. 187-210.

[12] Brenner, Zarnekow, and Wittig, *Intelligent Software Agents.*

[13] Bradshaw, *Software Agents*; Michael N. Hunhs and Munindar P. Singh, *Readings in Agents.*

[14] Hyacinth S.Nwana, “Software agents: An overview,” *Knowledge Engineering Review* 11, 3 (October/November, 1996): 205-224.

[15] Wooldridge and Jennings, “Intelligence agents: Theory and practice.”

[16] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, “Webwatcher: A learning apprentice for the world wide web,” in *Working Notes of the AAAI Spring Symposium: Information Gathering from Heterogeneous, Distributed Environments* (Stanford University: AAAI Press, 1995), pp. 6-12.

17 H. Lieberman, "Letizia: An agent that assists web browsing," in *Proocedings of the International Joint Conference on Artificial Intelligence IJCAI95* (1995), pp. 924-929.

18 Armstrong, Freitag, Joachims, and Mitchell, "Webwatcher."

19 Erik Selberg and Orentz Etzioni, "The Metacrawler architecture for resource aggregation on the web," *IEEE Expert* (IEEE, January/February 1997): 8-14.

20 Brenner, Zarnekow, and Wittig, *Intelligent Software Agents*; Wooldridge and Jennings, "Intelligence agents: Theory and practice."

21 Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield, "Secret agents - a security architecture for the KQML agent communication language," in *Proceedings of the ACM CIKM Intelligent Information Agents Workshop* (New York: Association of Computing Machinery, December 1995).

22 Michael R. Genessereth and Steven P. Ketchpel, "Software agents;" Wooldridge and Jennings, "Intelligence agents: Theory and practice."

23 Orentz Etzioni, N. Lesh, and R. Segal, "Building softbots for UNIX, in *Software Agents-Papers from 1994 Spring Symposium*, Technical Report SS-94-03 (AAAI Press, 1994), pp. 9-16.

24 Michael Coen, *SodaBot: A Software Agent Environment and Construction System*, Technical Report 1493 (MIT AI Lab, 1994).

25 Yigal Arens, Craig A. Knoblock and Chun-Nan Hsu, "Cooperating agents for information retrieval," in *Proceedings of the Second International Conference on Cooperative Information Systems* (Toronto, Ontario, Canada: University of Toronto Press, 1994); Craig A. Knoblock and José Luis Ambite, *Software Agents*.

26 J.L. Ambite and C. A. Knoblock, "Planning by rewriting: Efficiently generating high-quality plans," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (1997); Craig A. Knoblock, Steven Minton, Jose Luis Ambite, and Naveen Ashish, "Modeling web sources for information integration," in *Proceedings of the Fifteenth National Conference on Artificial Intelligence* (Madison, WI, 1998).

27 Oren Etzioni, "Moving up the information food chain."

28 Erik Selberg and Orentz Etzioni, "The Metacrawler architecture for resource aggregation on the web."

29 H. Lieberman, "Letizia: An agent that assists web browsing."

30 Thorsten Joachims, Dayne Freitag, and Tom Mitchell, "A tour guide for the world wide web,"in *Proceedings of IJCAI97* (August 1997); R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, "Webwatcher: A learning apprentice for the world wide web."

31 J. Hullen, Ralph Bergmann, and F. Weberskirch, "Webplan: Dynamic planning for domain-specific search in the internet," in *Workshop Planen und Konfigurieren PuK-99* (1999).

32 Chelliah Thirunavukkarasu, Tim Finin, and James Mayfield, "Secret agents - a security architecture for the KQML agent communication language."

33 Reticular Systems, *AgentBuilder. An Integrated Toolkit for Constructing Intelligent Software Agents* (Reticular Systems Inc., February 1999).

34 Deepika Chauhan and Albert D. Baker, "Jafmas: A multiagent application development system,"in *Proceedings on The Second International Conference on Autonomous Agents Agent's 98* (Minneapolis, May 9-13, 1998).

35 <http://www.fipa.org> (27 May 2001).

36 Jeremy Pitt and Fabio Bellifemine, "A protocol-based semantics for FIPA'97 acl and its implementation in Jade," in *Proceedings of AI*IA* (1999); Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa, "Jade - a fipa-compliant agent framework," in *Proceedings of PAAM'99* (London, April 1999), pp. 97-108.

37 Charles Petrie, "Agent-based engineering, the web, and intelligence," *IEEE Expert* 11, 6 (December 1996): 24-29.

38 Anthony Chavez and Pattie Maes, "Kasbah: An agent marketplace for buying and selling goods," in *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology* (London, UK, April 1996).

39 David Wilkins and Karen Myers, "A common knowledge representation for plan generation and reactive execution," *Journal of Logic and Computation* 5, 6 (1995): 731-761; D.E. Wilkins and D.L. Myers, "Multiagent planning architecture."

40 Manuela Veloso, Tucker Balch, and Scott Lenser, "Integrating information agents, planning, and execution monitoring," in *Proceedings of Agents-2000* (June 2000).

41 J. Baxter and R. Hepplewhite, "A hierarchical distributed planning framework for simulated battlefield entities," in *Proceedings of 19th Workshop of the UK Planning and Scheduling Special Interest Group PLANSIG 2000* (December 2000).

42 Any MAPWEB-agent is built using a set of standard and reusable Java packages and classes implemented as the basis to build the different system agents.

43 Tim Finin, Jay Weber, et. al., *Draft specification of the KQML agent communication language* (1993); Tim Finin, R. Fritzson, D. Mackay, and R. McEntire, "KQML as an agent communication language,"in *Proceedings of the Third International Conference on Information and Knowledge Management CIKM94* (New York: Association of Computing Machinery, 1994), pp. 456-463.

44 This characteristic is being developed at the moment.

45 This module is being developed at the moment.

46 Manuela M. Veloso and Jaime G. Carbonell, "Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization," *Machine Learning* 10, 3 (March 1993): 249-278; J. Hullen, Ralph Bergmann, and F. Weberskirch, "Webplan: Dynamic planning for domain-specific search in the Internet."

47 M. Veloso, J. Carbonell, A. Perez, D. Borrajo, E. Fink, and J. Blythe, "Integrating planning and learning: The Prodigy architecture," *Journal of Experimental and Theoretical AI* 7 (1995): 81-120.

48 Finin and Weber, *Draft specification of the KQML agent communication language.*

49 David Camacho, Daniel Borrajo, and Jose Manuel Molina, "Travelplan: A multiagent system to solve web electronic travel problems," in *Workshop on Agent-Based Recommender Systems. Fourth International Conference on Autonomous Agents* (Barcelona, Catalonia Spain: ACM, June 2000); David Camacho, José M. Molina, and Daniel Borrajo, "A multiagent approach for electronic travel planning," in *Proceedings of the Second International Bi-Conference Workshop on Agent-Oriented Information Systems AOIS-2000* (Austin, TX, USA: AAAI-2000).

50 Agnar Aamodt and Enric Plaza, "Case based reasoning: Foundational issues, methodological variations and system approaches," *AI Communications* 7, 1(1994): 39-59; Manuela M. Veloso and Jaime G. Carbonell, "Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization."

DAVID CAMACHO is a lecturer at the Department of Computer Science, Universidad Carlos III de Madrid (UC3M). He is a member of the Complex and Adaptive Systems Group (SCALab). He holds a B.Sc. in Physics Science (1994) from the Universidad Computense de Madrid. He has researched in several areas, including planning, inductive logic programming, fuzzy logic, and multi-agent systems. He has also participated in international projects on automatic machine translation and optimization of industrial processes. *Address for correspondence:* Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain, E-mail: dcamacho@ia.uc3m.es

JOSÉ M. MOLINA received his Ph.D. in Telecommunication Engineering from the Universidad Politecnica de Madrid, in 1997. He has been a member of the System, Signal and Radio Communications group in the University Politecnica of Madrid since1992. He is an Associate Professor at the Computer Science Department, Carlos III of Madrid University. He also participates in the Complex Adaptive Systems group. He has published more than 5 journal and 50 conference papers. His current research focuses on the application of soft computing techniques (Neural Networks, Evolutionary Computation, Fuzzy Logic and Multi-agent Systems) to engineering problems, such as RADAR, robot control and vision. *Address for correspondence:* Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain, E-mail: jmolina@ia.uc3m.es

DANIEL BORRAJO has been University Professor in Computer Science at the Universidad Carlos III de Madrid (UC3M) since 1998. Previously, he was Associate Professor at UC3M and Universidad Politécnica de Madrid (UPM). He received his Ph.D. in Computer Science from UPM in 1990. He holds a B.Sc. in Computer Science (1987) from the Universidad Politécnica de Madrid. He has been vice-dean of the Computer Science degree at UC3M and, currently, he is the head of the Department of Computer Science. He has published over 50 journal and conference papers. Dr. Borrajo's main research interest is the integration of different Artificial Intelligence techniques, especially concerning machine learning. His main research interest now is focusing the learning system towards real-world planning domains and problems, considering the quality of the solutions, and the self-experimentation of the system. *Address for correspondence:* Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain, E-mail: dborrajo@ia.uc3m.es

RICARDO ALER is a lecturer at the Department of Computer Science, the Universidad Carlos III. He has researched in several areas, including automatic control knowledge learning, genetic programming, and machine learning. He has also participated in international projects on automatic machine translation and optimization of industrial processes. He holds a PhD in Computer Science from Universidad Politécnica de Madrid (Spain) and a M.Sc. in Decision Support Systems for Industry from Sunderland University (UK). He graduated in Computer Science at Universidad Politécnica de Madrid. *Address for correspondence:* Universidad Carlos III de Madrid, Avda. de la Universidad, 30, 28911 Leganés (Madrid), Spain, E-mail: aler@inf.uc3m.es