MAPWEB: Cooperation between Planning Agents and Web Agents

David Camacho, José M.Molina, Daniel Borrajo, Ricardo Aler
Universidad Carlos III de Madrid
Avda. de la Universidad, n.30
28911,Leganes,Spain
{dcamacho,jmolina,dborrajo}@ia.uc3m.es, aler@inf.uc3m.es

ABSTRACT

This paper presents MAPWEB (**M**ulti**A**gent **P**lanning in the **W**eb), a multiagent system for cooperative work among different intelligent software agents whose main goal is to solve user planning problems using the information stored in the World Wide Web (WEB). MAPWEB is made of a heterogeneous mixture of intelligent agents whose main characteristics are cooperation, reasoning, and knowledge sharing. The architecture of MAPWEB uses four types of agents: *UserAgents* that are the bridge between the users and the system; *ControlAgents (Manager and Coach Agents)* that are responsible to manage the rest of agents; *PlannerAgents* that are able to solve planning problems; and finally *WebAgents* whose aim is to retrieve, represent and share information obtained from the WEB. MAPWEB solves planning problems by means of cooperation between PlannerAgents and WebAgents. Instead of trying the PlannerAgent to solve the whole planning problem, the PlannerAgent focuses on a less restricted (and therefore easier to solve) problem (what we call an abstract problem) and cooperates with the WebAgents to validate and complete abstract solutions. In order for cooperation to take place, a common language and data structures have also been defined.

Categories and Subject Descriptors

H.3.5 [**Online Information Services**]: Data sharing, Web-based services; I.2 [**Artificial Intelligence**]; I.2.6 [**Learning**]: Knowledge acquisition; I.2.8 [**Problem Solving**]: Planning; I.2.11 [**Distributed Artificial Intelligence**]: Intelligent agents, Multi-Agent Systems, Web agents

Keywords

Information System, Agent Architecture, Multi-Agent Systems, Web Agents, Intelligent Agents, Planning.

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 allow the users to look for the information they need. However, currently it is impractical to build a single and unified system that combines all of the possible information sources that could be useful to the users. Some of these problems 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 for the information. For instance, a traveller 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 NEO domains (Non-combatant Evacuation Operations), it would be useful to combine information coming from weather forecast sites and information obtained from GIS (Geographical Information System) servers.
- The information stored could change dynamicaly over time. A short term weather forecast site is a good example.

Due to the previous 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 maintain and to develop this kind of systems [3, 28]. 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 those systems are focused mainly on the amount of information to be retrieved [17].

Integrating heterogeneous information is one of the goals of MAPWEB. However, having complete and good 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 about complex domains. For instance, a traveller needs to be supplied good plans that combine different means of transport in an efficient way. Similarly, NEO and military operations need intelligent systems to move units and supplies on the terrain in a coordinated and efficient manner. Artificial Intelligence provides software components that fill in that gap: planning systems to find good quality plans in complex domains, machine learning systems to learn from experience in those domains, etc. In our work, we use such AI techniques but in a Multi-Agent System (MAS) framework (a part of what is called Distributed Artificial Intelligence or DAI). These systems are built using a set of modular components, or *agents* [24] that are **specialized** at solving a particular problem aspect. This decomposition allows each agent to use the most appropriate paradigm for solving its particular problem [9, 22]. Any MAS uses the agent concept, which is extensively described in [10, 43]. The properties of a MAS can be summarizez as follows:

- Each agent has an incomplete amount of information or does not have the required abilities to solve the whole problem.
- There is no centralized control.
- Data is not centralized, therefore agents must share their data.
- System execution is asynchronous, any agent can be working while it receives queries anytime.
- Each agent has an internal state. It is also able to reason about the environment and possibly learn to improve its behaviour.

Any MAS should be able to include new different software applications (or complex systems) and use them like a new agent in the system. Unfortunately, including a new software application in the system does not guarantee that it will integrate correctly within the system and that it will achieve the expected results. Therefore, it is necessary to provide methodologies to integrate distributed control algorithms and problem solving techniques into the system.

The agent framework provides several advantages for our purpose, because:

- First, multiagent systems are societies of (usually) heterogeneous software components, but communicating with each other in a common language. Therefore, MAS addresses directly the problem of integrating heterogeneous systems, each one handling different kinds of information, which is the problem complex web information retrieval systems have to face, as we mentioned before.
- MAS are often used to solve AI problems, like planning, scheduling, learning, etc and they have shown their worth because:
  - Different agents can combine their abilities in a **synergic** way. This has been clearly shown in the so called multistrategy learning systems, where different systems provide different characteristics useful for a common goal [2]. But this is not the only example. For instance, it has been shown that different planners work well in different domains [39]. Therefore in some cases it would be a good idea to combine different planner agents in the same MAS system [41].
  - They offer modularity, flexibility, and adaptability. A MAS uses a common language to communicate 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 [35].
  - MAS are inherently parallel, therefore permitting to execute more efficiently the computationally complex problems associated with AI.

However, integrating and coordinating different agents is a complex problem in itself that must be addressed [8, 21, 26]. When interdependent problems arise, the agents in the system must cooperate with each another to ensure that interdependencies are properly managed.

This paper presents a distributed multiagent architecture (MAPWEB) that accepts queries from the user. Such queries are actually problems to solve. The system then produces possible schematic solutions by means of AI problem solving techniques (planning and learning), which are then validated and completed by using the information available in the WEB.

This paper is divided into seven sections: Section 2 presents a revision 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 looks for solutions; Section 5 presents an example over the WEB using the designed system in a particular application domain; Section 6 summarizes the conclusions of the paper; and finally, Section 7 shows the future lines of work.

2. RELATED WORK

There are several approaches that try to work in different ways with the information stored in the Web. These approaches focus mainly on the retrieval (usally textual) information, but only few of them try to reason with that information. This section analyzes these systems and show how they handle the stored data. Next subsections describe the main domain characteristics that have been used for the deployed WEB applications, and the main characteristics of these applications. We will focus on the *agent-based* systems (like *Web and Intelligent agents*) and the *multiagent-based* systems that have been developed and deployed during the last years.

WEB and Internet applications can be classified in different ways. The next 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 information obtained, share with other systems its own knwoledge, and even could cooperate with others to obtain *solutions* that will be useful to the users. These kinds of systems have a wide range of characteristics that try to achieve more complex tasks than just retrieving information.

Figure 1 displays a posible classification of the most common WEB applications. Solid lines show that most of these kinds of systems should belong to this class and the discontinous lines show that some applications could be built by using a subset of characteristics so that intelligence and/or robustness of the system is increased.

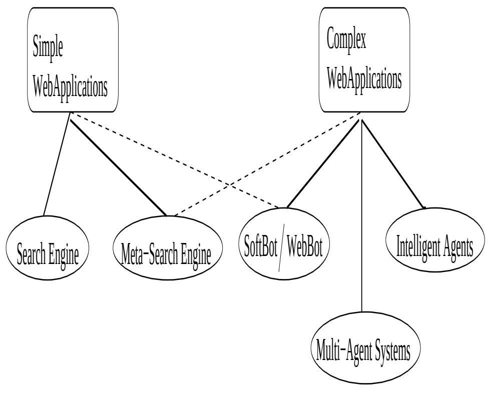


Figure 1: Generic classification of WEB systems.

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 experiences or from other agents experiences to anticipate the necessities from the user. In fact, agent technology is a combination of many technologies [10] including but not limited to, neural networks, expert systems, fuzzy logic, machine learning, planning, etc ...

It is difficult to characterize accurately what is an agent. There is a wide literature about this [9, 10, 24, 34]. The following points can be used to characterize an intelligent agent:

- Agents are *proactive* in nature (although they can and will be reactive as well) [43].
- Agents can *learn* as a result of their actions - not to mention their mistakes [5, 30].
- Agents can be *predictive* in nature [5, 30, 33].
- Other key attributes into the paradigm include *autonomy*, *security*, *personality*, and *mobility* [10, 36, 43]. However, an agent needs not have all of 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 agents or "better suited for the job" agents [22, 43].

Different and successful intelligent agents have been developed last years. Next, some of these agents are briefly described:

- `Softbot`. This agent interacts with a software environment by using and interpreting the environment feedback. The softbot efectors are UNIX commands that allows to the agent change the user's environment state [18].
- `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 which 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, like on-line assitants or meeting scheduling agents [15].
- `SIMS` and `Ariadne`. These intelligent information agents are focused mainly on information retrieval and intregrating different kinds of information sources. SIMS focuses on the integration of well-structured databases [16, 28], while the ARIADNE project deals with accessing information from more loosely structured Web sources [4, 29].

2.2 WebAgents

Currently there is an enormous number of Web applications that offer different services, like search and meta-search engines (*Lycos, Altavista, Yahoo, etc ...*), e-commerce markets, auctions, web directories, etc ... to Internet users [17]. As we said in Section 1, due to the current evolution of the WEB (and other on-line information sources), it is almost required some sort of intelligent assistance. 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 informations. Any WebAgent performs 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 developement about these kinds of systems. Here we only mention some of them:

- `MetaCrawler`. The METACRAWLER SOFTBOT is a parallel WEB search service that provides a single interface with which any user can query popular general-purpose WEB search engines, such as LYCOS or ALTAVISTA. METACRAWLER has some characteristics that allows it to obtain resuls of higher quality than simply showing the output from each search service [33].

- `Letizia` is a *user interface agent* that assists a user browsing the World Wide Web. As 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 a best-first search augmented by heuristics inferring user interest from browsing behavior [30].

- `WebWatcher` is a "tour guide" agent for the world wide web. Once any user asks to WebWatcher what kind of information is seeking, it accompanies the user from page to page. While the user browses the web, it highlights hyperlinks that it believes could be of interest. Its strategy for giving advice is learned from feedback from earlier tours. Currently WebWatcher is online only on an irregular basis [5, 27].

- `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 [23].

2.3 Multi-Agent Systems

Due to the growing agent-technologies importance 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 *p*aradigm. We will only examine several popular toolkits and deployed MAS.

- `AgentBuilder`. This is a very popular commercial toolkit to build and test agent-based software. Agents constructed using AgentBuilder communicate using KQML [20]. It allows to develop and extend the standard KQML performatives (or messages) to include any additional performatives [25].

- `JAFMAS`. This toolkit provides a framework to help developers to structure their ideas into specific agent applications. It directs development from a speech-act perspective and supports multicast and directed communication, KQML or other speech-act performatives. It also performs some analysis of multi-agent system coherency and consistency [13].

- `JADE` (Java Agent Development Framework) is a software development framework aimed at developing multi-agent systems and applications, conforming to FIPA [1] standard for intelligent agents. `JADE` can be considered an *agent middle-ware* that implements an *Agent Platform* and a development framework [7, 32].

[1] *http://www.fipa.org*

- `JATLite` is a framework for creating multi-agent systems. `JATLite` includes a message router (*agent message router* or simply the *AMR* agent) that supports message buffering, allowing agents to fail and recover. Agents can send and receive messages using KQML, an early ACL standard, although other languages such as FIPA's ACL can also be used. Message buffering also supports a name-and-password mechanism that lets agents move freely between hosts [31].

- `Kasbah` is a virtual market place on the WEB where users can create autonomous agents to buy and sell goods on their behalf. Users can specify parameters to guide and constrain the agent overall behaviour. Any intelligent agent in `Kasbah` is an object (an instance of a class) and the market place allows 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 main interest in `Kasbah` is its multi-agent aspect. It is a good framework to test different important characteristics in this kind of systems like *negotiation* [14].

- `MPA`. The Multiagent Planning Architecture (`MPA`) 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 efforts of a diverse set of technologies and human experts [41, 42].

- `CMUexpress` is a MAS architecture developed at CMU whose purpose is to plan, execute plans, and monitor its performance. It has been applied to Non-combatant Evacuation Operations (NEO). In this particular case, the whole system integrates about 20 agents. In particular, it includes MMM (a user interface developed at SRI), Ariadne (an information agent developed at ISI), 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 way. First, Ariadne locates the civilians. Then, CMUExpress provides routing plans to transport them, besides 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 website [37].

- Finally, there is a hierarchical multiagent 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 behaviour. Agents in the society are organised in a hierarchical military manner. For instance, there is a Squadron Commander agent, a Troop Commander agent, a Tank agent, etc. This framework allows the more reactive behaviours of agents at the lower level of the hierarchy bo be guided by more deliberative planning from agents above them in the hierarchy. In particular, a constraint planner (deliberative) and an anytime planner (reactive) are combined within the hierarchy [6].

3. MAPWEB: A MULTIAGENT ARCHITECTURE FOR REASONING IN THE WEB

As we said before, the main advantage for using MAS techniques is the flexibility and adaptability of the resulting system. A MAS could be made of several and heterogeneous elements. These elements or *agents*, can play different programmed roles, could execute different functions, and could modify their behaviour dynamically.

MAPWEB is a MAS approach that integrates heterogeneus agents. These agents build a different set of *"logic-layers"* between the users and the WEB. The architecture *hides* the WEB to the users. Thus, the user avoids coping with the overload of information. Figure 2 shows the four-layered architecture of MAPWEB.

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

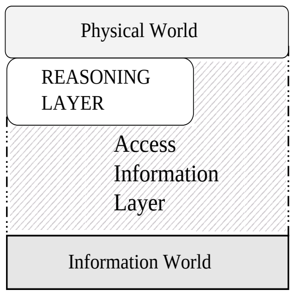


Figure 2: World/Web Layers.

The way MAPWEB implements the previous multi-layer architecture, can be seen in Figure 3. The system is built by a set of agents that can communicate, share knowledge and cooperate among them to search for solutions to problems posed by users.

This architecture has been designed to cope with some frequent problems existing in the WEB. To do this, it is necessary to use an internal knowledge representation sharable by the agents, and use different reasoning techniques that allow the agents to search for new solutions. MAPWEB is a MAS approach that integrates different heterogeneous agents with different roles into the agent-society. They can be summarized as:

- **UserAgent**: this agent connects the *physical world* and the *reasoning layer*. It pays attention to user queries and displays to them the solution or solutions found by the system. UserAgents get problem queries from the users and give them to a reasoner-agent. PlannerAgent is at the moment the only developed reasoner-agent developed, but different kinds of reasoner-agents, like LearningAgents will be in the future. Subsequently the reasoner-agents requests for solutions to those problem.

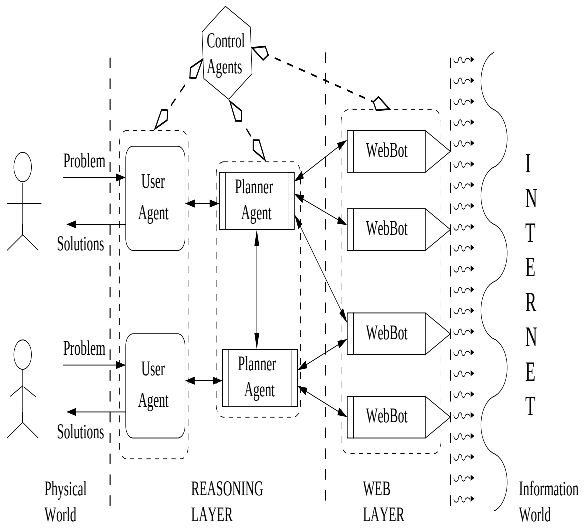


Figure 3: MAPWEB general Architecture.

- **ControlAgents**: These agents belong to the *reasoning layer* and due to 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**: Handles the insertion and deletion of agents in the system. This agent is the responsible for building dynamical *teams* of agents, each one of them specialized in problem solving tasks.
  - **CoachAgent**: This agent manages a set of heterogeneous agents that represent a *team* of agents that accept problems from any agent (software or human) in the system and try to solve them.

  Figure 4 shows the interrelation among these kinds of agents and the rest of the system agents in MAPWEB. Agents are organised in teams, each one is managed by a coach. The whole system is managed by a manager. Any UserAgent, PlannerAgent or WebAgent could belong to several teams if it could be necessary to the correct work of the team.

- **PlannerAgent**: This agent (belonging to the *reasoning layer*) receives a planning problem, builds an abstract representation of it, and solves it. PlannerAgents have different skills like communication and planning.

- **WebAgent**: These agents belong to the *access information layer* and connect the *reasoning layer* with the *information world*. Its main goal is to fill in the details

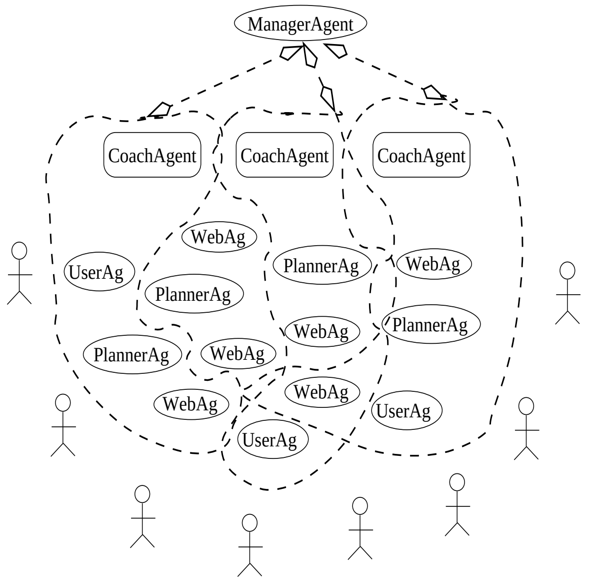


Figure 4: Manager and Coach Agents Organization.

of the abstract plans obtained by the PlannerAgents. It obtains that information from the WEB.

Some of the basic characteristics (see Figure 5) in any MAPWEB-agent are:[2]

1. *Control module*: it manages all the possible tasks performed by the agents. This module is basically made of an agenda, some policies, and a set of especialized skills.

2. *Knowledge module*: this module is used by the different agents to store their own kwnowledge.

3. *Skills module*: this module implements the specialized skills of any agent in the system.

4. *Communication module*: Implements the communication protocol with other system agents (UserAgents, PlannerAgents, CoachAgents, or WebAgents). This module is implemented using two sub-modules:

   - *Transport module*: implements a TCP/IP network level communication between two agents running in different computers.
   - *Language module*: implements an standard version of KQML [19, 20] that allows to share a common language between two agents in MAPWEB.

Next subsections give a more detailed description about the different agents: roles, architectures, and organizations.

[2]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.

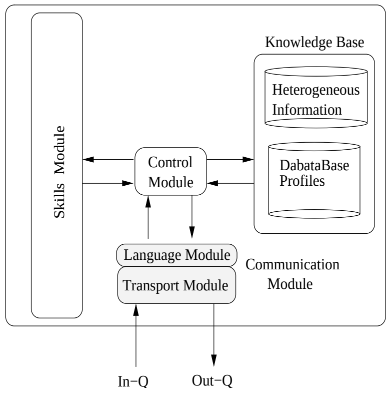


Figure 5: Skeleton-Agent in MAPWEB.

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 comunicate with the users and an implementation of the standard language KQML to communicate with other agents in the system. In Figure 6 a modular description of UserAgent architecture is shown.

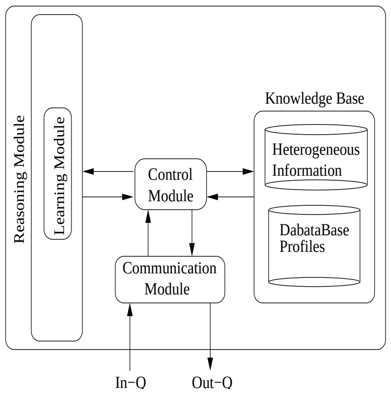


Figure 6: UserAgent Architecture.

The *Knowledge Module* is used by the UserAgent to store a set of different users profiles and successful old solutions, that should be used by UserAgent (applying its learning skills using its (*Learning Module*) to analyze and customize the system.[3]

The main goals for a UserAgent are:

- To accept problems from users, and to show the solutions found by MAPWEB.
- To analyze the problems and to obtain an homogeneous representation of them.
- To communicate with PlannerAgents to ask for solutions.

In order to fulfill these previous goals, it is necessary to provide for each particular domain the specific set of GUI that

[3]This characteristic is being developed at the moment.

can represent all the necessary input/output information to comunicate with the external world, and to define an ontology that allows the other agents in the system to know the type of the problem that must be solved. In Section 5, the set of GUIS for the analyzed domain are shown.

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 shows a modular description.

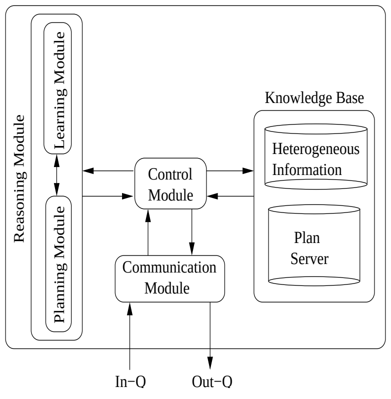


Figure 7: PlannerAgent Architecture.

Some of its most interesting characteristics are:

- *Communication module*: it implements a subset of specific *performatives* (speech-acts in KQML) to share plans or subplans between PlannerAgents.
- *Knowledge module*: Stores all the useful information for the agents. It is composed by two main submodules:
  - *Heterogeneous Information*: This stores the useful data (heterogeneous information) about the application domain, planning operators, heuristics, information about other agents characteristics and statistics information, etc ...
  - *Plan server*: This module stores old plans or subplans to find out a new solution.[4]
- *Control module*: to manage the different agent modules. The following are some of its main functions:
  - 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 allows to handle the questions asked by other agents and its own tasks.
  - To handle all possible answers given to questions asked by other agents/users.
- *Reasoning module*: PlannerAgents have mainly two submodules:
  - *Learning modules*: They can modify the system behaviour if the obtained solutions are successful to the user queries. Currently a Case-Base Planning Module is being developed, and it is used to gain efficiency in planning processes by retrieving and adapting old stored solutions in the own agent to avoid the planning process itself [23, 40].
  - *Planning module*: Works to solve the user problem. Currently the planning module uses a non linear planner named PRODIGY4.0 [38].

[4]This module is being developed at the moment.

The PlannerAgents use a planner as the main reasoning module. The agent generates an abstract representation of the problem and the specific users 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 gaps of these abstract solutions.

3.3 ControlAgents

As we said in the previous section, there are two different types of Control agents in MAPWEB. They have the same architecture (see Figure 8) with different roles.

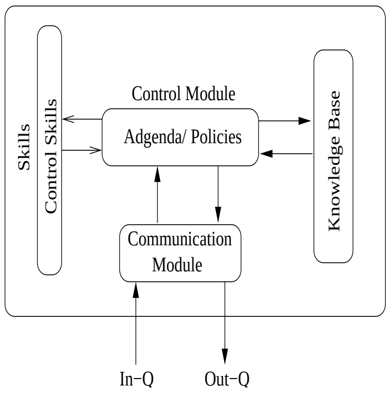


Figure 8: Generic ControlAgent Architecture.

We could summarise the differences as follows:

1. **ManagerAgent**:
   - There is only one of them.
   - It is responsible to add and remove other agents from the system.
   - It manages which agents are active in the agent society.
   - It groups teams of agents.
   - It determines which agents are shared between different teams.
2. **CoachAgent**
   - It controls a team of agents, guaranteeing stability and a smooth working of the active agents.
   - It reports problems to the ManagerAgent. For instance, when a new agent is required for the team.

- It guarantees that agendas of the agents in the team are coherent.

To work correctly in MAPWEB (for any possible Multi-agent topology) it is necessary to use at least one Manager and one Coach to build teams of agents that will be able to reason over the user problems.

3.4 WebAgent

The *WebAgents*, like other system agents, have their own modular architecture (this architecture is shown in Figure 9). A WebAgent handles (CONTROL MODULE) questions from other agents (PlannerAgents), and translates these questions into queries to access the WEB (INTERNET ACCESS MODULE). 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 several places to look for the requested information.

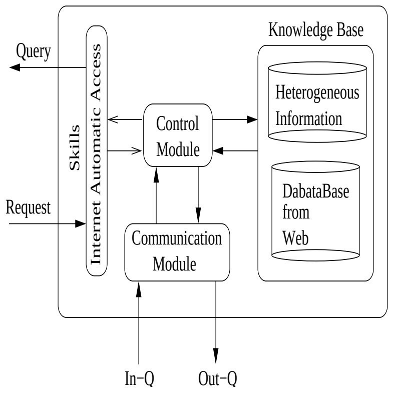


Figure 9: WebAgent Architecture.

Although MAPWEB has a general architecture and it is possible to apply it to different domains, this 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 particular domain (see section 5).

4. PROBLEM SOLVING AND COOPERATION IN MAPWEB

MAPWEB has an architecture where different agents need to cooperate to reach solutions. Different agents need to share knowledge and skills to complete the abstract solutions obtained by the PlannerAgent. MAPWEB success needs from both characteristics: sharing knowledge to obtain new solutions and to use the different Web and reasoning skills among the different MAPWEB agents to find useful solutions for the users. In the next sections the format for sharing and communicating knowledge, and the generic cooperative solving process in MAPWEB, are analyzed.

4.1 Shared Information Among Agents

Agents in MAPWEB use a common representation for the knowledge. This characteristic allows to simplify the processes of sharing and reasoning with the knowledge. The communication among agents uses **performatives**. Any performative stores an implicit order to other agent. To communicate two system agents, a subset of the KQML format (Knowledge Query and Manipulation Language) [19] is currently being used. In Table 1 this format is shown. This example shows the representation for two performatives: ACHIEVE and TELL. The first performative (ACHIEVE) is sent by a PlannerAgent (*PAgent1*) to a WebAgent (*WBot1*) asking for WEB information that the WebAgent are specialized in. The second performative (TELL) is the request from the WebAgent to the PlannerAgent, this request stores the retrieved information obtained by the *WBot1*.

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

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)

Table 1: Some performatives in MAPWEB.

4.2 Cooperation in MAPWeb

This section presents the way 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 problem is that of a person wishing to fix (final situation) a broken car (initial situation). A solution to a problem is the sequence of actions to get from the initial situation to the final one (called a `plan`). Usually, actions are defined in terms of `operators`. For instance, `screw(x)` could be an operator to use the screwdriver on any screw `x`. Therefore, a solution to the car fixing problem could be something like the plan showed in Figure 10.

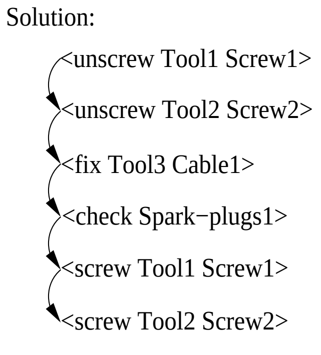


Figure 10: Possible *Fixing car* Plan.

A set of problems that uses the same operators is called a `domain`. The goal of MAPWeb is to give solutions to problems in a domain as defined above.

The sequence MAPWeb follows to solve a problem is as follows:

1. The user interacts with the UserAgent to define his/her problem. Then the UserAgent sends an ACHIEVE performative to a PlannerAgent containing the problem definition.
2. The PlannerAgent receives the user problem and analyzes it. Usually, a user problem contains a lot of details that makes problem solving for classical AI planning systems computationally very expensive. So, before attempting to solve it, the PlannerAgent discards some of the details and transforms the user problem into an abstract representation. For instance, in the car fixing domain, there could be many different kinds of pieces and tools to handle those pieces. In that case, the PlannerAgent would reduce the number of kinds of pieces and tools to a manageable size. Then, the user problem would be transformed into an abstract representation that uses only the reduced set of pieces and tools. At this point, the PlannerAgent would use a planning system to solve the abstract problem and get several abstract possible solutions. However, the user requires all the details to be able to apply the plan. Furthermore, many of the abstract solutions might not be valid in reality because it ignores actual details. Therefore, the abstract plans have to be completed and validated. The PlannerAgent analyzes which parts of the abstract plans require to be completed, and asks for details to the WebAgents.
3. WebAgents receive PlannerAgent queries for details, look for them in those websites the agent is specialized in, and returns the information to the PlannerAgent in a shared format. If it cannot find such information, the PlannerAgent will be told so, and it will discard all the plans that include this unvalid operator. For instance, different car companies could have websites informing about technical characteristics of cars, tools, and pieces, which could be used by 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 contain 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 solutions would include which actions to perform and the specific tools and pieces to use. This plan could be utilised directly by the user.

5. MAPWEB APPLICATION EXAMPLE

In principle, MAPWEB can be applied to many different 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 poblems in this domain. Some ideas about earlier versions of MAPWEB have been described in [11, 12]. This section will first describe the e-tourism domain (i.e. how solutions are represented) and then we describe how the different agents in MAPWEB cooperate to provide solutions to the user. Communications between the UserAgent, the PlannerAgent, and the WebAgents will be described in detail.

5.1 Electronic Tourism Domain

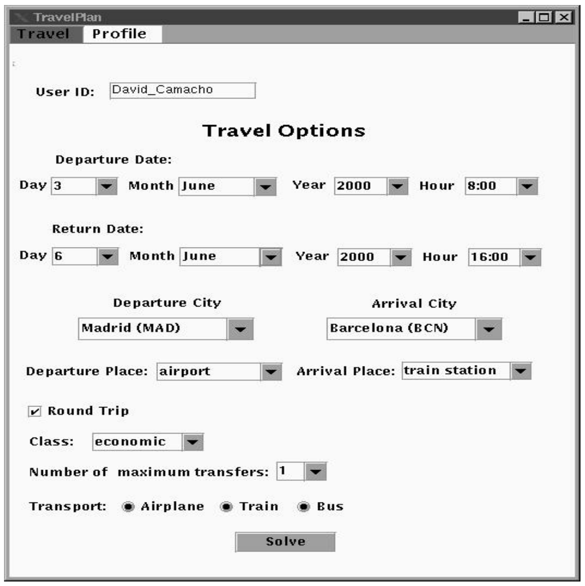
An e-tourism system must provide the user services such as:

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

MAPWEB has the abilities enumerated above. However, in this paper, we will focus on the logistics problem of providing the user with plans to move from one place to another place. Moving from place to place involves long range travels that can be achieved by means of airplanes, trains, or buses. It also involves taking local transport 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 enumerated in Table 2.

5.2 UserAgent $\longrightarrow$ 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 on the problem domain: other domains would require other GUIs. Figure 11 shows 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 supply to the system is as follows:

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
...	

Table 2: E-tourism planning operators

- Departure and return dates
- Departure and arrival cities
- Origin and arrival places inside the cities (which airport, train station, bus station, etc ...)
- One way or return trip
- Maximum number of transfers
- Economic cost (luxury, business, first class, tourist class, etc ...)
- Long range kind of transport (airplane, train, or bus)

In the example described in Figure 11, the user wants to travel to Barcelona (Spain) from Madrid (Spain) on June the 3rd at 8:00. The return date is June the 6th at 4:00 or later. The user wants to start his travel from an airport and wishes to end his trip at a train station in Barcelona. S/he wants to minimise cost and s/he does not mind what kind of long range transport is used. Also, s/he does not want to transfer more than once.

Once the UserAgent has received the user 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 to build an abstract representation which retains only those parts that are 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 transports (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 the elements mentioned. Instead, the PlannerAgent builds an abstract problem in the following way:

1. First, it defines an abstract city. This city includes all possible local transports, but only the long range transport terminals that the user wishes to use are included. For instance, if the user wants to travel on plane only, 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 about the long range transports, the abstract city has airports, bus stations, and train stations.
2. Then, this abstract city is repeated as many times as the maximum number of transfers supplied by the user. It is important to remark that the cities are abstract cities.
3. Finally, the rest of details provided by the user are ignored at this stage. For instance, departure and arrival times, travel cost, etc ... are not considered. This data will be used later to query the WebAgents and validate the abstract solutions.

For instance, 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 one of the cities includes all possible local transports, abstract locations (`hotel1`, ...) and terminals (`airport0`, `trainstation0`, ...). Finally, the planning problem would include an initial situation of the user being in `airport0` at `city0`, and a goal of the user being in `trainstation2` at `city2`.

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

```
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>
```

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

Table 3: WebAgent Information Retrieved

```
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>

...
```

This is a set of abstract plans that contain no details and some of which might not even be possible to executed. Therefore, those 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?`). `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)
```

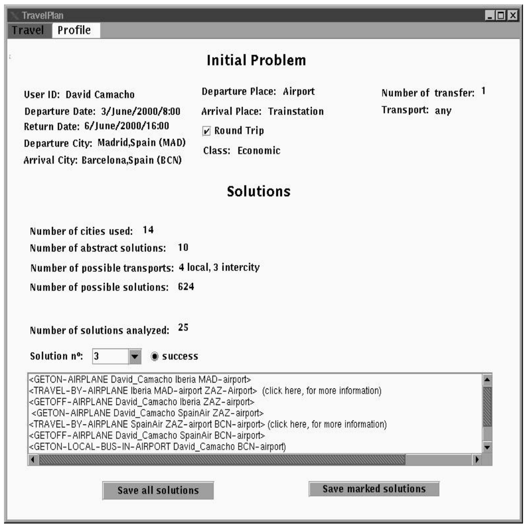
Those queries (an all the additional information given by the UserAgent) are sent to several WebAgents that know about airplane travel, so that variable `plane0?` is instantiated as well.

A WebAgent receives a query and associated data and transforms it into the actual web query. The WebAgent knows the structure of the data stored in the web sites it is specialized in, and knows how to look for information in those web sources. The information retrieved is then analyzed and stored in a common template, which is subsequently sent to the PlannerAgent. For instance, in our example, the following information (see Table 3) would be returned to instantiate variable `plane0?`. Actually, that variable can be instantiated in many ways, as many as possible flights there are from Madrid to Valencia. Table 3 shows some retrieved flight information by WebAgents.

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

5.4 PlannerAgent $\longrightarrow$ UserAgent Communication

Finally, the list of actual plans is received by the UserAgent and displayed to the user. Figure 12 shows the *output*-GUI where the found plans for our problem are displayed. A couple of solutions are shown in Table 4. 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 ...



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 12: UserAgent Output.

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)	

Table 4: Solutions given by MAPWEB **.**

6. CONCLUSIONS

We have presented a multiagent approach (MAPWEB) to solve planning problems by using information distributed in 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 those plans (by the WebAgents). This is done because planning problems contain a lot of details that makes classical AI problem solving computationally very expensive.

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

There is another important reason to divide the planning process into two cooperating agents. Information in the WEB is hetereogeneous 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 complexities of the information sources.

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

7. FUTURE LINES OF WORK

- Cooperation among several PlannerAgents. In many planning domains, a problem can be divided into a set of subproblems. Each subproblem could be sent to different PlannerAgents. This would be useful for two reasons. First, problem solving is parallelized. And second, different kinds of subproblems could be sent to specialized PlannerAgents, that could 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 previosly to reduce WEB access.
- Application of Case Based Reasoning techniques [1, 40], so that new planning problems can be solved by adapting previously solved plans which were similar. 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 order solutions and recommend the best ones by using user profiles and by learning from user previous behaviour.

8. REFERENCES

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological varations, and system approaches. *AICom-Artificial Intelligence Communications. IOS Press*, 7:39–59.

[2] R. Aler, D. Borrajo, and P. Isasi. Genetic programming and deductive-inductive learning:A multistrategy approach. In J. Shavlik, editor, *Proceedings of the Fifteenth International Conference on Machine Learning, ICML'98*, pages 10–18, Madison, Wisconsin, July 1998.

[3] J. L. Ambite and C. A. Knoblock. Agents for information gathering. In *IEEE Expert: Intelligent Systems and their Applications*, September/October 1997.

[4] J. L. Ambite and C. A. Knoblock. Planning by rewriting: Efficiently generating high-quality plans. In *In proceedings of the Fourteenth National Conference on Artificial Intelligence*, 1997.

[5] 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*, pages 6–12, Stanford University, 1995. AAAI Press.

[6] J. Baxter and R. Hepplewhite. A hierarchical distributed planning framework for simulated battlefield entities. In *In Proceedings of 19th*

Workshop of the UK Planning and Scheduling Special Interest Group (PLANSIG 2000), December 2000.

[7] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa-compliant agent framework. In *Proceedings of PAAM'99, London*, pages 97–108, April 1999.

[8] A. H. Bond and L. Gasser. *Readings in Distributed Artificial Intelligence*. San Francisco California, Morgan Kaufmann, 1988.

[9] J. Bradshaw. *Software Agents*. Menlo Park California, AAAI:Press, 1997.

[10] W. Brenner, R. Zarnekow, and H. Wittig. *Intelligent Software Agents. Foundations and Applications.* Springer-Verlag. ISBN: 3-540-63411-8, New York, 1998.

[11] D. Camacho, D. Borrajo, and J. M. 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), June 2000. ACM.

[12] D. Camacho, J. M. Molina, and D. 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), July 2000. AAAI-2000.

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

[14] A. Chavez and P. Maes. Kasbah: An agent marketplace for buying and selling goods. In *Proceedings of the First International Conference on the Practical Appication of Intelligent Agents and Multi-Agent Technology, London, UK*, April 1996.

[15] M. Coen. *SodaBot: A Software Agent Environment and Construction System.* MIT AI Lab Technical Report 1493, June, 1994.

[16] Y. A. Craig A. Knoblock and C.-N. 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.

[17] O. Etzioni. Moving up the infomation food chain. *AI Magazine*, 18(2):11–18, summer 1997.

[18] O. Etzioni, N. Lesh, and R. Segal. Building softbots for unix. In *Software Agents-Papers from 1994 Spring Symposium (Technical Report SS-94-03)*, pages pp. 9–16. AAAI Press, 1994.

[19] T. Finin and J. W. et. al. *Draft specification of the KQML agent communication language.* Jun 15 1993.

[20] T. Finin, R. Fritzson, D. Mackay, and R. McEntire. Kqml as an agent communication language. In *In Proceedings of the Third International Conference on Information and Knowledge Management (CIKM94)*, pages 456–463. New York: Association of Computing Machinery, 1994.

[21] L. Gasser. *Distributed Artificial Intelligence: Theory and Praxis*, chapter An Overview of DAI, pages 9–30. Kluwer Academic, 1992.

[22] M. R. Genessereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[23] J. Hullen, R. Bergmann, and F. Weberskirch. Webplan: Dynamic planning for domain-specific search in the internet. In *Workshop Planen und Konfigurieren (PuK-99)*, 1999.

[24] M. N. Hunhs and M. P. Singh. *Readings in Agents.* San Francisco California, Morgan Kaufmann, 1997.

[25] R. S. Inc. *AgentBuilder. An Integrated Toolkit for Constructiong Intelligent Software Agents.* ed. by Reticular Systems Inc., February,1999.

[26] N. R. Jennings. *Coordination Techinques for Distributed Artificial Intelligence*, pages 187–210. O'Hare et al., 1996.

[27] T. Joachims, D. Freitag, and T. Mitchell. A tour guide for the world wide web. In *Proceedings of IJCAI97*, August 1997 (longer version internal CMU technical report September 1996).

[28] C. A. Knoblock and J. L. Ambite. *Software Agents*, chapter Agents for Information Gathering. J. Bradshaw ed., AAAI/MIT Press, Menlo Park, CA, 1997.

[29] C. A. Knoblock, S. Minton, J. L. Ambite, and N. Ashish. Modeling web sources for information integration. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence, Madison, WI*, 1998.

[30] H. Lieberman. Letizia: An agent that assists web browsing. In *Proocedings of the International Joint Conference on Artificial Intelligence (IJCAI95)*, pages 924–929, 1995.

[31] C. Petrie. Agent-based engineering, the web, and intelligence. *IEEE Expert*, 11(6):24–29, December 1996.

[32] J. Pitt and F. Bellifemine. A protocol-based semantics for fipa '97 acl and its implementation in jade. In *Proceedings of AI*IA*, 1999.

[33] E. Selberg and O. Etzioni. The metacrawler architecture for resource aggregation on the web. In *IEEE Expert*, pages pp. 8–14. IEEE, January/February 1997.

[34] H. S.Nwana. Software agents: An overview. *Knowledge Engineering Review*, 11(3):205–224, October/November 1996.

[35] K. P. Sycara. Multiagent systems. *AI Magazine*, 18(2), 1998.

[36] C. Thirunavukkarasu, T. Finin, and J. Mayfield. Secret agents– a security architecture for the kqml agent communication language. In *In Proceedings of the ACM CIKM Intelligent Information Agents Workshop*. New York: Association of Computing Machinery, December 1995.

[37] M. Veloso, T. Balch, and S. Lenser. Integrating information agents, planning, and execution monitoring. In *In Proceedings of Agents-2000*, June 2000.

[38] 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*, 7:81–120, 1995.

[39] M. M. Veloso and J. Blythe. Linkability: Examining causal link commitments in partial-order planning. In *Proceedings of the Second International Conference on AI Planning Systems*, pages 170–175, Chicago, IL, June 1994. AAAI Press, CA.

[40] M. M. Veloso and J. G. Carbonell. Derivational analogy in PRODIGY: Automating case acquisition, storage, and utilization. *Machine Learning*, 10(3):249–278, Mar. 1993.

[41] 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.

[42] D. E. Wilkins and K. L. Myers. A common knowledge representation for plan generation and reactive execution. *Journal of Logic and Computation*, 5(6):731–761, 1995.

[43] M. Wooldridge and N. R. Jennings. Intelligence agents: Theory and practice. *Knowledge Engineering Review*, October 1994.