AN ARCHITECTURE FOR WEB AGENTS

Zhisheng Huang
Vrije University of Amsterdam
The Netherlands
Email: huang@cs.vu.nl

Anton Eliëns
Vrije University of Amsterdam
The Netherlands
Email: eliens@cs.vu.nl

Paul de Bra
Eindhoven University of Technology
The Netherlands
Email: debra@win.tue.nl

KEYWORDS
Intelligent agent, web agent, agent architecture

ABSTRACT

In this paper we propose an extended BDI architecture for web agents. The architecture is general, it covers 2D web agent with text based interfaces for retrieval services as well as 3D web agents like avatar-embodied guides that help visitors to navigate in virtual environments. Furthermore, we define the primitives of sensor/effector of web agents, and show how those different types of web agents can be implemented, based on the general architecture.

INTRODUCTION

Currently, there is a lot of interest and work in the area of developing agents systems and applications that make use of web technology, which ranges from intelligent information agents, interface agents, to e-commerce agents. One of the central issues is how we can develop a general architecture for agents on the Web, so that no matter what the application domains are, the infrastructure can still be shared.

(Bell 1995) proposes a general architecture for intelligent agents. The architecture is so general that no matter what the application domains are, the kernel of the agent systems is stable, and the components which have to be specified are only the sensors and effectors of agent systems.

Based on Bell's architecture, in this paper, we propose a general architecture for web agents, which covers agents that provide a text-based interface to for example information retrieval services, as well as avatar-embodied guides that help visitors to navigate in virtual environments. Furthermore, we discuss the primitives of sensor/effector of web agents, and how those different types of web agents can be implemented, based on the general architecture.

This research is part of a Dutch research project WASP, for Web Agent Support Program. As part of this research project we are developing PAMELA, a Personal Assistant for Maintaining Electronic Archives. The architecture we present in this paper is meant to serve in selecting a suitable implementation in the WASP project.

A TAXONOMY OF WEB AGENT

Many types of web agents have been proposed in recent years, which range from domain-dependent agents to function-dependent agents. The natural questions related to that phenomenon are: what are the relations among so many different types of web agents? Are they redundant, or overlapped? Is there any taxonomy to classify them? In (Huang et al. 2000), we propose a taxonomy of web agents. This section is a brief introduction to the taxonomy of web agents. We consider the following three dimensions of web agent types:

- 2D versus 3D

A 2D web agent is one which is aware of http, file, and ftp protocols, whereas a 3D web agent is one which is aware of not only those protocols, but also virtual reality protocols. Typical 2D web agents provide a text-based interface for information retrieval services. Typical 3D web agents are avatar-embodied guides that help visitors to navigate in virtual environments.

- Client versus server

As the names imply, a client web agent is on the client side, whereas server web agent is on the server side. A typical client web agent can serve as a personal information assistant. A typical server web agent can serve as the front-end of web servers to offer information more intelligently.

- Singularity versus multiplicity

As the names imply, a single web agent would not consider the interface with other web agents, whereas multiple web agents would interact on each other.

There are different types of web agents based on those three dimensions, which consists of a complexity lattice of web agent types, which is shown in Figure 1. 3D-server-multiple-agents are at the top of the complexity hierarchy, whereas 2D-client-single-agents are at the bottom. All the dimensions we consider for the taxonomy are directly web-dependent. The dimension “2D-3D” specifies the internet protocols agents have to be aware of, the dimension “client-server” is concerned with internet service modes agents have to offer, and the dimension “singularity-multiplicity” determines agent communication languages. We do not consider the functional dimension which are not directly relevant to the Web, like cooperating agents, problem solving agents, and negotiation agents, etc. We do not discuss the dimensions which are domain dependent, like expertise seeking agents, e-commerce agents, etc. However, our taxonomy does cover most types of those agents. They are either 2D agent or 3D agent, either client agent or server agent, either single agent or multiplicity agent, or some of their combinations. Different types of web agents suggest different types of interaction modes with users and web servers. See (Huang et al. 2000) for details.

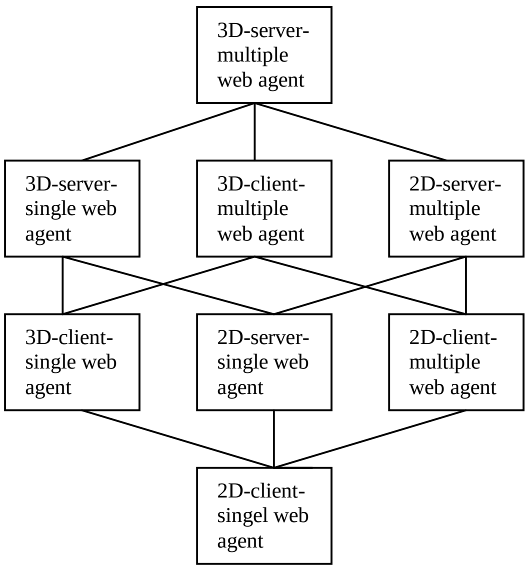
3D-server-multiple web agent
3D-server-single web agent
3D-client-multiple web agent
2D-server-multiple web agent
3D-client-single web agent
2D-server-single web agent
2D-client-multiple web agent
2D-client-singel web agent

Figure 1: Lattice of Web Agents

ARCHITECTURE FOR WEB AGENTS

(Bell 1995) proposes an architecture of intelligent agents, shown in Figure 2. According to Bell, a reasoning agent is situated in the real world and consists of two connected modules; a high-level (symbolic) reasoning system (or “mind”) and a low-level (procedural) action system (or “body”). The symbolic reasoning system is composed of a module for theoretical reasoning and a module for practical reasoning.

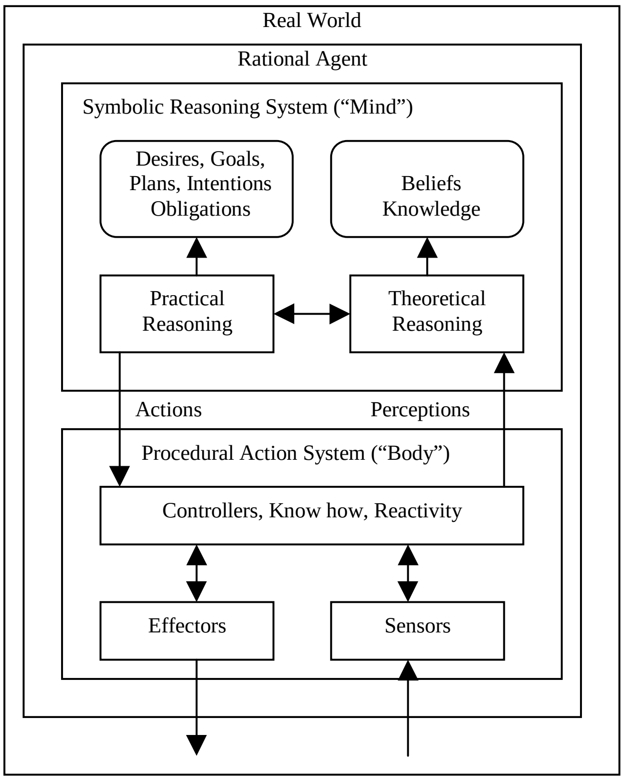
Real World
Rational Agent
Symbolic Reasoning System (“Mind”)
Desires, Goals, Plans, Intentions Obligations
Beliefs Knowledge
Practical Reasoning
Theoretical Reasoning
Actions
Perceptions
Procedural Action System (“Body”)
Controllers, Know how, Reactivity
Effectors
Sensors

Figure 2: Bell's Agent Architecture

The theoretical reasoning module represents the agent's beliefs, knowledge of, and reasoning about the world. The reasoning done by this module includes standard deductive reasoning as well as inductive, abductive, and probabilistic reasoning. It also performs database-type operations (lookup, update, addition, revision, deletion). The practical reasoning module represents the agent's reasoning about what it should do. The planning system generates plans to achieve the agent's goals, schedules the resulting actions for execution, passes them to the procedural action system, and monitors their execution by it. The procedural action system (or “body”) consists of controllers, motion systems and perception systems. This component represents the agent's physical capacities and skills (“know how”). The controllers control and monitor the sensors and the effectors and mediate between them and the reasoning systems. They receive high-level action commands from the practical reasoning system, expand these to the appropriate level of detail, pass them to the effectors systems for execution, and perform low-level monitoring on them. They also pass perceptions (symbolic descriptions of the environment based on feedback from the sensors) to the theoretical reasoning system. The “mind” and “body” function as co-routines each of which is more or less active depending on the environment, the processing (reasoning) resources available, and the tasks at hand. This allows the agent to form long-term strategic plans to execute them and to react to events.

Although a lot of agent architectures have been proposed (Jennings et al. 1998), we select Bell's agent architecture as the point of the departure for the following reasons:

- **Generality.** Bell's agent architecture is so general that no matter what the application domains are, the kernel of the agent systems is stable, and the components which have to be defined are only the sensors and effectors of agent systems. In particular, no matter whether we are interested in the development of a 2D web agent, or in that of a 3D web agent, The kernel of the web agent would be the same. The only difference between a 3D web agent and a 2D web agent are the corresponding operators of the sensors and the effectors.
- **Flexibility**. Although Bell's agent architecture is based on the popular Belief-Desire-Intention architecture, i.e., so-called BDI architecure (Rao and Georgeff 1991), it offer more elaborated conceptual models of rational agents, which cover knowledge maintenance and belief revisions in the theoretical reasoning component, as well as planning and scheduling in the practical reasoning component. Moreover, the sophisticated models do not cause big difficulties in the implementation, for the sub-components can be dropped when we consider the concrete implementation. We propose a simplified architecture for web agents in this paper.
- **Availability.** A lot of formal theories have been proposed based on BDI architecture (Jennings et al. 1998, Rao and Georgeff 1991). Bell's architecture is an extended BDI architecture. Furthermore, based on Bell's agent architecture, several theories have been developed, including a formal theory of dynamic goal hierarchies (Bell and Huang 1997), which investigate how rational agents can pursue their goals and drop their commitment rationally.

Considering the simplicity of the implementation, we propose a general architecture for web agents, which is shown in Figure 3, which is not only a simplified Bell's agent architecture but also an extended BDI architecture, for it consists of Belief-Desire-Intention components, plus sensor/effector components. We drop the component of obligation in the architecture and make no distinction between goals and intentions, and no distinction between belief and knowledge, for the simplicity of the implementation. However, those omitted components can be easy to be recovered when they are needed. Currently we use the Distributed Logic Programming language DLP (Eliëns 1992) to implement the web agent PAMELA. In the following, we define the operators of sensor/effector in logic programming languages. An operator which cannot be defined in terms of other operators is called a *primitive* of sensor/effector.

The followings are some operators of the sensors for 2D web agents. However note that the lists are not complete, and some of them are not primitives.

- *getURL(URL,Text):* get the text of the *URL*;
- *getValue(URL, Property,Value):* get the value of the property in *URL.*
- *isValid(URL):* check if the *URL* is valid;
- *isPermissible(URL,Permission*): check if the *URL* is permissive for the web agent to read/write/ execution.

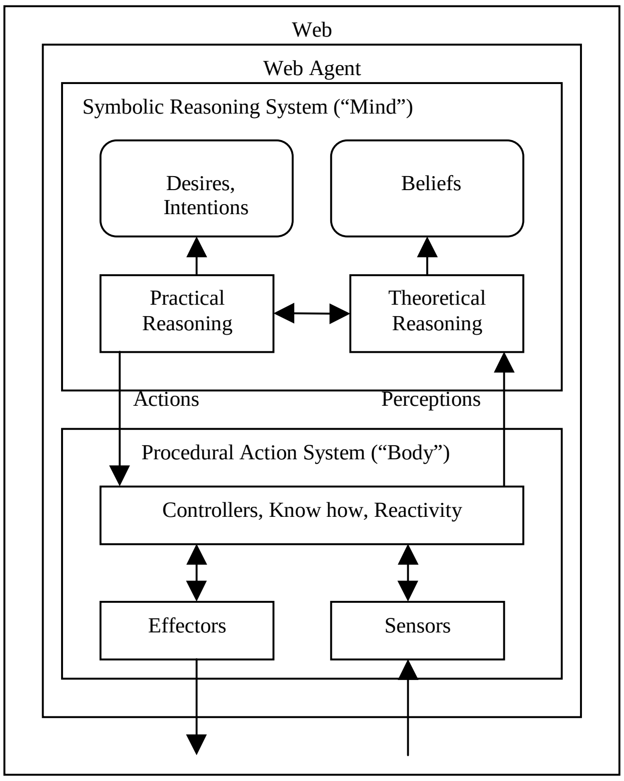

Figure 3: Architecture for Web Agents

The operators of the effectors are:

- *setText(URL, String, NewString, NewURL):* replace *String* in *URL* with *NewString*, and save it as a *NewURL* if the *newURL* is permissive for writing. If *URL* and *NewURL* are the same, then adding the link to the text without changing *URL*;
- *setLink(URL, String, Link, NewURL)*: add the hyper link to the string in *URL*, and save it as a *NewURL.*

According to our taxonomy of web agents, 3D web agents subsume 2D ones. Thus, all of the primitives of sensor/effector of 2D web agents are also valid for 3D web agents. Moreover, 3D web agents have their own primitives of the sensor/effector to process virtual environments. Virtual reality modeling language (ISO 1997) is a standard tool for virtual environments over the Web. Therefore, 3D web agents have the following primitives of the sensors. Please refer to VRML documents (ISO 1997) for the details of positions, orientations, rotations and viewpoints.

- *getViewpointPosition(Agent, X, Y, Z):* get the agent's viewpoint position;
- *getViewpointOrientation(Agent, X,Y,Z,R):* get the agent's viewpoint orientation;
- *getPosition(Object,X,Y,Z):* get object's position;
- *getRotation(Object,X,Y,Z,R):*get object's rotation.

The primitives of the effectors for 3D web agent are:

- *loadURL(URL);* load a VRML world;
- *setViewpointPosition(Agent,X,Y,Z):* set the agent's viewpoint position;
- *setViewpointOrientation(Agent, X,Y,Z,R):* set the agent's viewpoint orientation;
- *setPosition(Object,X,Y,Z):* set object's position;
- *setRotation(Object,X,Y,Z,R):* set object's rotation.

All of the primitives are valid for single web agents. For multiple web agent systems, we have the following primitives of sensor/effector, which are inspired from the agent communication language KQML. (Labrou and Finin 1994).

- *getAgentProperty(Agent, Property, Value):* get the value of the agent's property;
- *getReply(Agent, Topics, Message):* reply to the agent with the message of the topics.
- *ask(Agent, Topics):* ask the agent to reply on the topics;
- *tell(Agent, Topics, Message):* tell the agent the message on the topics.

All of web agents can be considered as client web agents. For server web agents, the primitives of sensor/effector are as follows:

- *ifRequested(Host,Requirement):* check if a service requirement is sent from a host;
- *reply(Host,Requirement,Message):* reply the host with the message on the requirement.

CASE STUDIES

In this section, we study several cases of the applications, by which we want to show how those primitives of sensor/effector can be used to define the behaviors of the web agents.

Scenario One*: (2D web agent for archive update and maintenance)* The user, Harry, is a conference organizer. He has a text file of the conference programme, in which there is a list of the speakers and the titles of their presentations. Harry wants the web agent PAMELA to help him to transfer the programme into a real html file, namely, try to link the speaker's homepage address to all of the names of the speakers, and add the links of the papers to the titles of the presentations. In particular, Harry has to advise the web agent PAMELA to do the followings:

- All of the papers now locate at the directory "C:/conference" locally;
- Check all the files in the directory, find the title of the file, and make the link to the title of the programme;
- Find the speaker's email address in the file; and then try to find the speaker's homepage address on the Web according to their email addresses. To simplify it, just try to guess the address of the homepage from the e-mail address by the following: if the email address is USER@DOMAIN, then the possible homepage address is: www.DOMAIN/~USER; if it is valid, then update it, if it is not valid, just simply use the email address as the hyperlink, namely, the link "mailto:USER@ DOMAIN".

Here is the part of the formalization of the scenario:

Beliefs:
addLinktoTitle(Programme, URL):-
getValue(URL,''title",Title),
setLink(Programme,Title, URL, Programme).

guessHomepage(EmailAddress,HomepageAddress):-
eq(EmailAddress,USER+@+Domain),
eq(HomepageAddress, "www."+Domain+"/~"+USER),
isValid(HomeAddress).

guessHomepage(EmailAddress,HomepageAddress):-
eq(EmailAddress,USER+@+Domain),
eq(GuessedHomepageAddress,"www."+Domain+"/~
"+USER), NOT isValid(GuessedHomeAddress,),
eq(HomepageAddress,"mailto:"+EmailAddress).

addHomepagetoName(Programme,Speaker):-
getEmailAddress(Speaker,EmailAddress),
guessHomepageAddress(EmailAddress,
HomepageAddress),isValid(HomepageAddress),
setLink(Programme,Speaker,HomepageAddress,
Programe).

getEmailAddress(Speaker,EmailAddresss):-
conferencepaper(URL),getValue(URL,

```
“author", Author),eq(Author,Speaker),
getValue(URL,``email",EmailAddress).
```

Intentions(Goals):
updateProgramme(“C:/conference/programme.htm").

Desires:
updateProgramme(“C:/conference/programme.htm").

It seems that we can simply use a single logic program (with extended sensor and effector primitives) to fulfill the task. A natural question is why we need the complicated BDI architecture. One of the main reasons is: if the web agent is implemented in that way, that would mean that whenever the web agent gets a goal from the user, like *updateProgramme*, she has to execute it immediately. An intelligent web agent would not simply behave like that. Just considering a situation in which you ask your secretary to task a task, she would promise you to do that, however, it is not necessary for her to fulfill the task immediately, unless the task is urgent and the deadline is coming soon. She can have her own desires and preferences. She knows how to schedule the tasks and fulfill the tasks in due time. A web agent is designed with the extended BDI architecture would behave more intelligently. The web agent would commit herself to fulfill the task, however, she may drop the commitment under some special situations, for instance, there are more important tasks for her to do, or the programme file is not available because of some unexpected reasons. Therefore, we need a more sophisticated architecture for Web agents so that the web agents have a powerful practical reasoning component for planing and scheduling the tasks, and they can decide their commitments or drop some of them under certain conditions.

Scenario Two: *(3D web agents for soccer playing)* We have implemented a soccer game, in which 3D web agents are the players. A demonstration version of the WASP soccer game is now available from the WASP project web site: www.cs.vu.nl/~huang/wasp. Based on the primitives of sensor/effector proposed above, we can define a lot of extended actions, like run-and-trace, shooting, passing, for the soccer playing agents. See (Huang et al. 2001) for details. 3D web agents in WASP soccer games show significantly intelligent behaviors.

Just see the following simple example, in which we define the action “look-at-ball” in terms of the primitives of sensor/effector proposed in this paper, by using the Distributed Logic Programming Language DLP:

```
look_at_ball(Player,Ball) :-
        getPosition(Player, X,_,Z),
        getPosition(Ball, X1,_,Z1), X \== X1,!,
        R is atan((Z1-Z)/(X-X1)) - sign(X-X1)*1.57,
        setRotation(Player,0.0, 1.0, 0.0, R).
```

CONCLUSIONS

Based on Bell's agent architecture, in this paper we have proposed an extended BDI architecture for web agents. We have shown that the architecture is general and it covers different types of web agents and their applications. We are developing PAMELA, a Personal Assistant for maintaining Electronic Archives, based on the extended BDI architecture, by using Distributed logic programming language DLP. As one issue of further work, we will investigate the complete specifications of sensor/effector.

REFERENCES

Bell, J. 1995. “A Planning Theory of Practical Rationality”. *Proceedings of AAAI'95 Fall Symposium on Rational Agency.* 1-4.

Bell, J. and Huang, Z. 1997. “Dynamic goal hierarchies”, *Intelligent Agent Systems, Theoretical and Practical Issues,* LNAI 1209. Springer. 88-103.

Bell, J. and Huang, Z. 1999. “Dynamic Belief Hierarchies”, *Formal Theories of Agents*, LNAI 1760. Springer. 20-35.

Eliëns, A. 1992. *DLP, a Language for Distributed Logic Programming.* Wiley.

Huang, Z.; Eliëns, A.; van Ballegooij, A.; and de Bra, P. 2000. “A Taxonomy of Web Agents”. *Proceedings of the 11th International Workshop on Database and Expert Systems Applications,* IEEE Computer Society. 765-769.

Huang, Z.; Eliëns, A.; and Visser, C. 2001. “Programmability of Intelligent Agent Avatars”, Research report. Department of Computer Science, Vrije University of Amsterdam.

ISO. 1997. *VRML97: The Virtual Reality Modeling Language.* ISO/IEC. 14772-1, 1997.

Jennings, N.R.; Sycara, S.; and Wooldridge, M. 1998. “A Roadmap of Agent Research and Development”, *Autonomous Agents and Multi-Agent Systems I.* Kluwer, 275-306.

Labrou, Y. and Finin, T. 1994. “A Semantics Approach for KQML – a general purpose communication language for software agents”, *Proceedings of the 3rd International Conference on Information and Knowledge Management,* ACM Press.

Rao, A. and Georgeff, M. 1991. “Modeling rational agents within a BDI-architecture”, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning,* Morgan Kaufmann Publishers. 473-484.