An agent-based architecture for web-assistants

A. García-Serrano, F. Bueno, J. Correas, J. Z. Hernández

Departamento de Inteligencia Artificial
Facultad de informática
28660 Boadilla del Monte, Madrid
34 91 336 7441
{agarcia, bueno, jcorreas, phernan}@dia.fi.upm.es

Abstract. This paper presents a proposal for the modelling of the different cognitive components taking part in an advanced Human Computer Interaction for an advice-giving assistant. The paper focus is the web-assistant prototyped within the project ADVICE in the e-commerce scenario. In particular, introduces a model for the deployment of this type of intelligent agents with multi-user capabilities. The current working version of the prototype is running on a Windows NT platform. In addition, the components of the prototype are easily portable to different platforms since they are implemented in Java (JDK 1.3) and Ciao Prolog both portable languages. After the presentation of the prototype some conclusions of this work are included, as well as a proposal for future work in the VIP_ADVISOR project, regarding the design and deployment of intelligent agents.[1]

Keywords and Conference Topics

Intelligent Agents, Human-Computer Interaction, Agent-based architecture.

Contact Person

Ana Garcia-Serrano

Departamento de Inteligencia Artificial
Universidad Politécnica de Madrid
Campus de Montegancedo s/n
28660 Boadilla del Monte (Madrid), España
email: agarcia@dia.fi.upm.es
Tlf.; 91 336 7441
Fax.: 91 352 4819

Section: Paper Track

[1] This work was supported by the ADVICE project (IST-1999-11305) and the ongoing project VIP_ADVISOR (IST-2001-32440)

An agent-based architecture for web-assistants

Abstract. This paper presents a proposal for the modelling of the different cognitive components taking part in an advanced Human Computer Interaction for an advice-giving assistant. The paper focus is the web-assistant prototyped within the project ADVICE in the e-commerce scenario. In particular, introduces a model for the deployment of this type of intelligent agents with multi-user capabilities. The current working version of the prototype is running on a Windows NT platform. In addition, the components of the prototype are easily portable to different platforms since they are implemented in Java (JDK 1.3) and Ciao Prolog both portable languages. After the presentation of the prototype some conclusions of this work are included, as well as a proposal for future work in the VIP_ADVISOR project, regarding the design and deployment of intelligent agents.[2]

Keywords and Conference Topics Intelligent Agents, Human-Computer Interaction, Agent-based architecture

1. ADVICE-GIVING APPLICATIONS

The development of an Advice-giving Assistant shares the complexity of the Human Computer Interaction and the Multi-agent systems fields. This complexity mainly resides in the management of the different information models engaged in an advanced communicative process.

In order to reach a coherent management of the complete interactive process (i.e. the dialogue), three major subtasks are identified: the interface interpretation and presentation tasks, the dialogue management, and the intelligent content generation. In our proposal the whole interactive process is performed by an agent-based system:

– The interpretation of the user and generation of the system utterances are the *interface agent* main tasks.
– The identification of the coherence of the discourse pieces is the responsibility of the *interaction agent*.
– The selection of the system utterances according with the circumstances (i.e. user requirements, state of the dialogue, and others) is performed by the intelligent *agent*, that includes a knowledge-based system incorporating the domain knowledge.

[2] This work was supported by the ADVICE project (IST-1999-11305) and the ongoing project VIP_ADVISOR (IST-2001-32440)

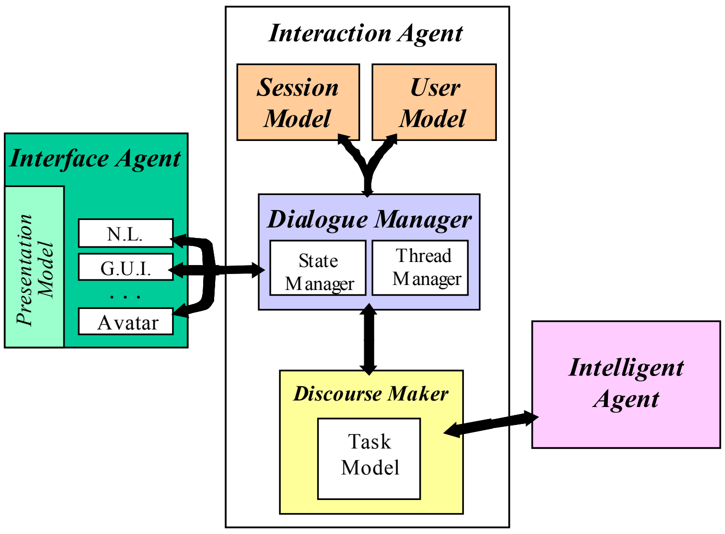

Figure 1: Cognitive Architecture

The ADVICE deployed system is an advice-giving application for E-commerce, supporting a customer adapted intelligent assistance in the basis of:

1. A fluid communication with the user, supported by different media (natural language, GUI and 3D character) in order to achieve an advanced multimedia user interface.
2. An interaction approach supported by a dialogue model with a twofold dimension: informational and intentional. The informational approach establishes that the coherence of the discourse follows from semantic relationships between the information conveyed by successive utterances. The intentional approach claims that the coherence of discourse derives from the communicative intention of both speakers.
3. A customised configuration of the offer according to the user needs and the evolution of the dialogue.

The architecture of the system includes an Interface Agent to manage the multimedia presentation, the Interaction Agent to support the advanced user-system interaction and the Intelligent Agent incorporating a knowledge-based model for an e-shop. The main processing steps of the whole application are presented in the sequel.

Example: The user input is the utterance
“I think I need a pendulum saw”
(or a click on the pendulum saw image on the web-page).

The next processes are performed:

Interface agent: Identification of a stream of speech acts
Null, inform(data,identity,product,saw),
inform(data,feature,product,type=pendulum)
Interaction agent: Identification of a new state into the current dialogue (plan based)
Solve (the previous state was request(product))
Interaction agent: Identification of the topic
Product type = pendulum (to be added to the session model)

After the user interaction (by clicking or writing a sentence) the interface agent output is a stream of speech acts representing the semantic and intentional contents of the user interaction. This stream is received by the interaction agent, that updates its different information models. These information models are the session, the dialogue and the user models. From this, the interaction agent identifies from the current state of the dialogue, by a plan-based process, the kind of the next interaction step to be performed in order to produce an answer to the user according to the evolution of the dialogue. The detail of this process is out of the scope of this paper (for more information see [2,3]). It happens that sometimes the answer can only be produced by requesting the intelligent agent for a (new) offer tree that guides the selection of the next interaction step.

When the interaction agent sends a request to the intelligent agent, its answer is obtained by a knowledge-based reasoning process. The answer is either a new offer tree with the products that fulfill the current user requirements or the prune of the previous offer. An offer tree has either a solution or a set of alternatives from which a selection is made to generate a proposal for the next step in the dialogue. The solution or the selected alternative is sent to the interaction agent, that updates the information models and produces a complete answer (a new stream of speech acts) to be sent to the interface agent.

The interface agent finally either generates a new question to the user (pro-activity), when an alternative was the proposal made by the intelligent agent, or produces an explanation showing some information related to the solution. In the first case, when the required information from the user is received and returned to the interaction agent, the prune of the offer tree according to the user answer is performed.

2. INTER-AGENT COMMUNICATION

The agent communication is supported by XML-based contents of the messages they interchange. The semantic structures or streams of speech acts that represent the user and agents contents of the messages they interchange, are defined regarding the application.

This is a closed-environment application, so no one except interface and interaction agents will communicate with each other. Because of this, we decided not to use FIPA compliant messages in the ADVICE project. The agents simply send FIPA-ACL inform-like messages, and in the content of those messages the XML speech acts related to the current interaction step are included. The 14 speech acts were obtained from the analysis of 20 real dialogues between a customer and the seller of a real shop, with an average of 10 interactions per dialogue. This is not far from other works in the area of dialogue systems (see for example related work included in [3,4]).

Interface agent → Interaction agent

```
Performative: inform
Content: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE speech SYSTEM ".\\speech.dtd">
<speech><id><sessionid>uno</sessionid></id>
<speechact><informativeact><informdata><informfeature subject="product" con-
tent="subtype=plunge">
</informfeature></informdata></informativeact></speechact>
<speechact><informativeact><informdata><informidentity subject="product" con-
tent="saw">
</informidentity></informdata></informativeact></speechact>
 </speech>
```

Interaction agent → Interface agent

```
Performative: inform
Content: <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE speech SYSTEM ".\\speech.dtd">
<speech><id><sessionid>uno</sessionid></id>
<speechact><nullact type="dp"></nullact></speechact>
<speechact><requestact><choice matter="feature" subject="product"
content="Cutting depth=66/55"></choice></requestact></speechact>
</speech>
```

For an extension of this approach in an open environment, this messages could be adapted to a FIPA-ACL structure (we will go further into this issue in the last section).

3. AGENTS DEPLOYMENT IN CIAO-PROLOG

The system presented above targets any number of users. However, because of the heavy-weighted processes involved, it is not realistic to simply launch new processes for the whole application for each new user that connects to the system. An agent-based approach to this problem helps maintaining the solution simple and effective.
There are several problems in offering multi-user capabilities with this system, mainly due to the inner aspects of the different agents. It also depends on the communication features between them. The use of different programming languages may, in principle, also pose certain technical problems, which have been, however, easily solved, thanks to the use of standard user-transparent communication machinery.

In the next a description of the Ciao language and of its benefits for the deployment of the system is included. After this, a detailed explanation of the different options taken to the different stages of the final deployment of the system is presented.

Ciao is a multi-paradigm language based on logic programming. It supports Prolog, but also offers Object Oriented capabilities, functional programming style, parallel, concurrent, and distributed programming facilities. In particular, relevant to the system presented herein, Ciao includes support for Web related functionality and agent programming.

Basic support for agents in Ciao is elaborated from the object oriented extension, which in turn is an elaboration of the module system. This provides the basic constructs for capturing "individuality" in the sense of agents. Moreover, at the level of these individual entities, concurrency and distribution comes through via the concept of active modules/objects.

A module/object is active when it can run as a separate process. This concept provides for "autonomy" at the execution level. The active module service of Ciao allows starting and/or connecting (remote) processes which "serve" a module/object. The code served can be conceptually part of the application program, or it can be viewed alternatively as a program component: a completely autonomous, independent functionality, which is given by its interface. Ciao also includes HTTP connectivity, an interface to Java, and a basic HTML/XML parser.

The main problem in the deployment of the complete application after the isolated development of the interface, interaction and intelligent agents arose in the Interaction Agent. This agent keeps a memory and the state of the interaction with the client. This memory is also unique and independent for each user, and even different for the same client in different sessions. Things are different for the Natural Language (NL) components of the Interface Agent (in this paper we focus in the details of the interface agent related only with the NL components, as this is the work performed in the ADVICE project by the authors of this paper). This one keeps no state, and it works just serving petitions for interfacing with the user. Therefore, it is only necessary to manage the petitions as soon as possible. A similar thing happens for the Intelligent Agent, which simply acts as a server of petitions related to the application domain.
Our approach has been a two-sided double communication solution, based on the different functionalities of the agents in each side, and the communication features offered by the programming languages involved.

Two different techniques have been used to solve the multi-user problem for this system:

- **Queue of servers**. Conceptually, a single interface agent (NL components) might serve all the petitions from a queue (much the same as the other media drivers -- the avatar, the GUI, etc.-- are unique). However, this could delay the response time if there were too many petitions at the same time. This has been solved by creating a spool of interface agents, serving petitions as needed. At the starting time of the system, several interface agents are created. The number of agents in the spool depends on the congestion of the system, that is to say, the average of expected concurrent sessions.

- **Replication or instantiation**. For each client, an instance of the Interaction Agent is created, and destroyed when the interaction is over. This is needed because the interaction agent must keep the state, the context, and the history of the interaction for each user, so it needs an independent process for each one.
  The instance of the Interaction agent is double. This is because of the distributed nature of the application (the user interface might be hosted by a different computer than the dialogue manager or even the intelligent agent), and also because of the different programming languages used. The double nature of the interaction agent instance is depicted in Figure 2.
  The so called DPC (Dialogue Processing Components, that stands for the interaction and intelligent agents as a unit in the deployed system) keeps the history of the dialogue with the user. It is written in Ciao, and communicates with a Java object that acts as a front-end to it. This object is a mere channel for communicating with the Interface Agent, which handles the different media interfaces. One instance of this agent is also created for each client, because it keeps the response IP, the session ID and some other information about the client.
  On the other side of the DPC, the Intelligent Agent is unique, serving petitions from all active DPCs. In any case, the solution based on the spool of servers might also be applicable here.

The main problem of this approach is the communication. In a single user scenario, each agent could have a defined address, and basic TCP/IP (i.e., sockets) could be used. But in this context, the agents are created dynamically, and the IP address of each agent is not known *a priori*. This problem has been avoided by using standard, programmer-transparent communication protocols such as JAVA RMI and a well designed interface for Ciao/JAVA communication. The operation of the system is as follows:

1. At system start-up, two processes are launched: the SessionManager and the InteractionManager. These two processes bind themselves into the RMI Registry, thus becoming publicly accessible. They are registered with a fixed name: *rmi://host:1099/SessionManager* and *rmi://host:1099/InteractionManager*. These objects are the responsible of creating and registering the new instances of the agents created for each client.
2. When a new client enters into the system, an new instance of the Interface Agent is created with a call to the method SessionManager::getInterfaceAgent. The first commitment of this object is to manage the creation of a new instance of the DPC for the session that is being started.
3. For this purpose, the Interface Agent, once the reference of the InteractionManager is obtained, makes a call to the method InteractionManager::getDPC, which returns the reference of the remote object that has been created, a new DPC agent.
4. When a new DPC is created, the system creates an object belonging to the class InteractionAgent. The commitment of this object is to launch the corresponding Ciao process, and it has to communicate with this process. It delivers directly the

incoming XML message to the Ciao process, and it returns the output of this process as the output of its method invocation.

5. Once all the processes are created, the system is ready to serve user's petitions. When these interactions come from the Natural Language input form, the Interface Agent sends the utterance to the spool of NL Agents. This petition is served by one of those agents, no matter which one. In a similar way, the Natural Language output of the system is managed by one of the free NL Agents in the spool.
6. When the client shuts down the session, the DPC and the Interface Agent of its own are killed. The system calls the methods InteractionManager::shutdownDPC and InterfaceManager::shutdownSession. The first one also kills the Ciao process for the DPC.

In this way, the multi-user operation is rather simple (see figure 2). For each new client, the system carries out the same steps: it creates a new instance both of the Interface Agent and of the DPC. Each different pair of instances is bound to each different user, and manages just only its interaction.

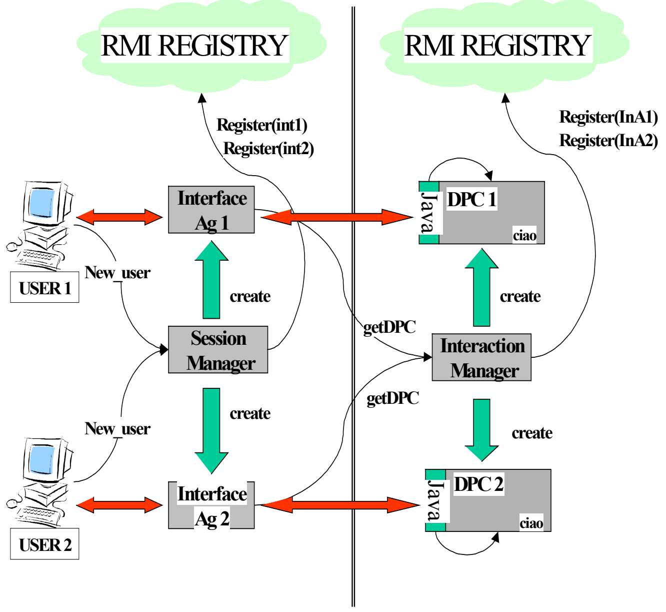

Fig. 2 Multi-user Extension.

With this philosophy the ADVICE system could manage 11 concurrent users. This was tested in a Pentium III computer, with 256 Mb of RAM memory. A more powerful machine could be used to manage more users, if required.

CONCLUSIONS

It was presented a cognitive architecture for supporting an advanced HCI model for the design of a virtual assistant in an e-commerce scenario. The architecture components divides the system into three major parts: interface, interaction, and intelligent cognitive components. A first version of the cognitive architecture presented has been implemented in a virtual assistant for e-commerce during the ADVICE project (www.advice.iao.fhg.de/). The deployed prototype was implemented in Ciao Prolog (www.clip.dia.fi.upm.es/Software) and JAVA and supports multi-user access.

For the VIP-Advisor project (IST-2001-32440), the idea is to use the extension of our approach to be ready for offering the knowledge-based and advanced interaction functionality to future personal agents. The interaction agent developed with our approach and connected with company devoted systems that contain expertise in the company domain (the substitute of the intelligent approach in our current application) will perform some kind of coherent dialogue with the personal agents according with the requirements and goals of the users that they represent.

In the future work we are going to include in the approach the complete set of FIPA compliant communicative acts and extend the communication capabilities of Ciao in order to support several other possibilities for intelligent agent architectures in open environments. The approach here will build on standard or broadly accepted interfaces. One step towards this is the JADE/Ciao support.

ACKNOWLEDGMENTS

Our thanks to the members of the CLIP and ISYS research groups for their work in the development of CIAO and ADVICE project.

REFERENCES

1. Ciao Prolog 1.6-p3 (http://www.clip.dia.fi.upm.es/Software)
2. A. García-Serrano, D. Teruel, "A personalized customer service by means of configurable offers and dialogue-based interaction" AMEC Workshop at Amsterdam, December 2001.
4. A. García-Serrano, J. Calle, and J.Z. Hernández. "Dialogue Management for an Advice Giving Virtual Assistant". Workshop on Knowledge and Reasoning in Practical Dialogue Systems, IJCAI 01. Seattle (USA).
5. A. García-Serrano, J.Z. Hernández, and P. Martínez. "Intelligent Assistance to E-commerce". eBusiness and eWork 2001 Conference. Venice (Italy), October 2001.
6. L. Rodrigo, A. García-Serrano, P. Martínez, 2001. Gestión Flexible de Diálogos en el proyecto Advice. In Procesamiento del Lenguaje Natural, n 27, pp. 319-320. SEPLN, sept. 2001