Software demonstrations

Tuesday, May 4 1999 10:00-12:30

Software demonstrations

Wednesday, May 5 1999 10:00-12:30

Arguing and Cooperating Agents

Michael Schroeder
City University, London
msch@soi.city.ac.uk, www.soi.city.ac.uk/ msch

Arguing and Cooperating Agents

Negotiation is fundamental to autonomous agents. In a negotiation process, a group of agents communicates with one another to come up with a mutually acceptable agreement. In many application, such a process may be the exchange of prices between a buyer and seller according to a particular protocol; in others, negotiation involves a more complicated process of argumentation to determine and change the beliefs of agents. The ACA-framework (Arguing and Cooperating Agents (Schroeder 1999)) deals with multi-agent argumentation. There are two fundamental types of interaction for multiple agents: they cooperate and argue. An agent, which does not know anything about a certain literal, cooperates with others, which help out and possibly provide the knowledge. As for argumentation, an agent believes in something and argues with other agents to determine whether this belief is valid or has to be revised. When arguing we can distinguish skeptical and credulous agents. The former is more critical towards its own beliefs than the latter. Technically, the difference amounts to the acceptance of different types of arguments: an undercut denotes an attack to a premisse of a conclusion and a rebut an attack to the conclusion itself. Skeptical agents accept undercuts and rebuts to their own arguments, whereas credulous agents accept only undercuts. In the argumentation protocol, the agents use the speech acts (Searle 1969) propose, oppose, and agree and for cooperation ask and reply.

To reduce the number of messages exchanged an agent will ask only its cooperation partners for help and only those whose domain of expertise covers the issue in question. Similarly, an agent proposes its beliefs only to its argumentation partners with the corresponding domain of expertise. All in all, an agents is defined by

1. an avatar to represent the agent
2. a set of arguments
3. a set of predicate names defining the agent's domain of expertise
4. a flag indication whether the agent is credulous or skeptical
5. a set of cooperation partners
6. a set of argumentation partners

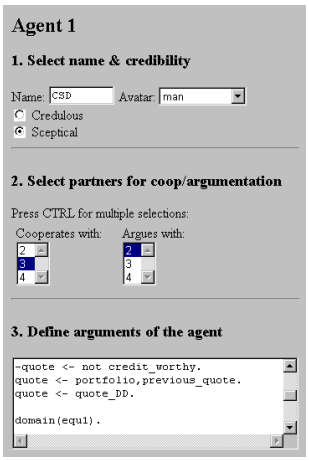


Figure 1: An HTML form to define agents.

Figure 1 shows a form to specify such an agent. The user can query an agent about its beliefs. The query results in a conversation among the agents to establish the answer to the query. For visualisation, the agents are represented by avatars and the conversation among them is animated by the messages slowly flying as text strings from sender to receiver. Figure 2 shows a screen shot of such a conversation for a BT business process explained below (Schroeder 1999).

Example

The example is derived from the ADEPT project (Jennings *et al.* 1996), which developed negotiating agents for business process management. One such process deals with the provision of customer quotes (Sierra *et al.* 1997) for

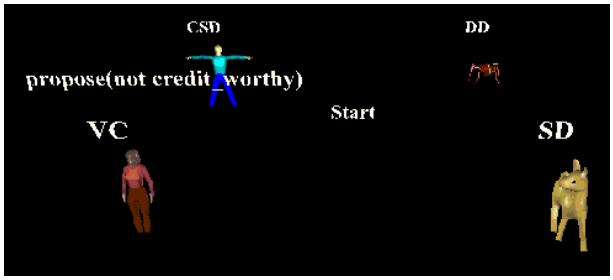


Figure 2: A screenshot of the animated argumentation trace generated.

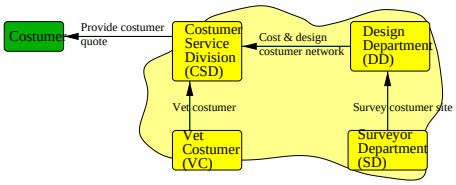


Figure 3: BT's business process to provide customer quotes.

networks adapted to the customers' needs (see Figure 3). Four agents are involved in this process: the customer service division (CSD), which makes the initial contact with the customer and delivers the quote eventually, the vet customer agent (VC), which determines whether the customer is credit-worthy, the design department (DD), which does the design and costing of the requested network if it is not a portfolio item, and the surveyor department (SD), which may has to survey the customer site for the design department.

The process works as follows. Initially, a customer issues a request. The CSD gathers some data for this request such as the requirements, the equipment already installed at the customer site, and how important the client is. Before any action is taken, the CSD asks the VC to vet the customer. If the customer is not credit-worthy the process terminates and no quote is issued to the customer. If it is credit-worthy, the CSD checks whether the required network is a portfolio item so that a previous quote exists. If positive, this quote is send to the customer, otherwise the design department is contacted. The DD develops its design and costing based on the information of given equipment held by the CSD. In many cases, however, this information may be out of date or not available at all, so that the site has to be surveyed. In this case, the DD contacts the surveyors to do a survey. After the survey is done, the DD can design and cost the network and the CSD can finally provide the customer quote.

The above scenario involves two fundamental types of interactions: argumentation and cooperation. If the DD wants to do its task it needs information held by the CSD. Therefore they cooperate. The CSD should not quote if the customer is not credit-worthy which it should assume by default. But the VC may counter-argue and give evidence for the credit-worthiness of the customer. Therefore VC and CSD argue. When arguing it is advisable to distinguish the credulousness of agents. The CSD and VC should be skeptical when arguing about giving a quote, while the other two are credulous. It is important to note that not all agents communicate with each other which would lead to a tremendous amount of messages exchanged, but each agent maintains a list of agents that it cooperates with and that it argues with. Besides knowing these partners, the agents know the partners domains so that they bother their partners only with requests relevant to them. Before we formally model and implement these aspects of the above business process we have to develop the theory underlying our argumentation framework.

Modelling CSD's arguments

Consider the customer service division CSD. It knows about the customer's equipment and its requirements and in the example we assume that the customer has requirement 2 and 3 and equipment 2 and 3. Furthermore, CSD knows that the customer is important. Besides these facts about a particular customer, CSD has some general rules such as requirements 1, 2, and 3 together make up a portfolio and can be quoted if a previous quote exists. Otherwise, the design department has to prepare the quote. CSD does not provide a quote if the client is not credit-worthy, which is assumed by default.

$requ2.$	$portfolio \leftarrow requ1, requ2, requ3.$
$requ3.$	$\neg quote \leftarrow not\ credit\_worthy.$
$equ3.$	$quote \leftarrow portfolio, previous\_quote.$
$equ2.$	$quote \leftarrow quote\_DD.$
$important.$	

CSD's domain Dom_1 covers all predicates occuring in P_1 and CSD will argue about credit-worthiness with the vet customer agent VC so that $Arg_1 = \{2\}$ and is skeptical in this respect so that $F_1 = s$. It cooperates on quotation with the design department and hence $Coop_1 = \{3\}$. So, $Ag_1 = \langle P_1, F_1, Arg_1, Coop_1, Dom_1 \rangle$.

References

Jennings, N. R. et al. 1996. Agent-based business process management. *Intl. J. of Cooperative Information Systems* 5(2&3):105–130.

Schroeder, M. 1999. An efficient argumentation framework for negotiating autonomous agents. Submitted.

Searle, J. 1969. *Speech Acts*. Cambridge (UK): Cambridge University Press.

Sierra, C.; Jennings, N.; Noriega, P.; and Parsons, S. 1997. A framework for argumentation-based negotiation. In *Proc. of ATAL-97*, 167–182. Springer-Verlag.

An Adaptive Interactive Agent for Route Advice

Seth Rogers Claude-Nicolas Fiechter Pat Langley
DaimlerChrysler Research and Technology North America
1510 Page Mill Road
Palo Alto, CA 94306
650-845-2533
rogers@rtna.daimlerchrysler.com

Current route advice systems present a single route to the user based on static evaluation criteria, with little or no recourse if the user finds this solution unsatisfactory. In this demonstration, we present a more flexible approach, the Adaptive Route Advisor. Our agent behaves more like a human travel agent, using a preference model, when known, and working with him or her to find a satisfactory route. The route advisor predicts what route a user will prefer based on a model of user preferences, and, if the predicted route is unsatisfactory, the advisor generates additional routes based on interaction with the user. The route that the user eventually selects serves as feedback to improve the preference model.

The major capabilities the Adaptive Route Advisor demonstrates are interactivity and automatic personalization. The system always provides the user with multiple route options and the opportunity to generate more. When the user selects a route, the system presumes that the user finds this route superior to the other options according to his internal preferences, and takes this as training data. We employ a perceptron algorithm that trains on differences between examples to generate a model of the user preferences. The next time this user requests a route, one of the routes the system suggests is the one that scores best according to the user preference model. The system also automatically records the user's familiar routes using Global Positioning System tracking, and includes a preference for or against familiar routes in the user model.

The route advice agent is designed for use in a car with a wireless Internet

connection. The interface is a lightweight client, and the remote servers perform compute-intensive and memory-intensive tasks. In the demo, participants see a detailed roadmap of the Seattle area. Participants or the operator enter the starting address, ending address, and a user identifier. Some user identifiers will be "pre-loaded" with plausible preference models and some familiar routes in the Seattle area, as recorded by a Global Positioning System unit we will place in our rental car. The Adaptive Route Advisor finds the preference model for the user (or creates a default model if none exists), and generates two routes between the endpoints: one optimized for that user model, and one exploratory route. The route advisor summarizes the route options in terms of estimated travel time, total distance and distance along highways, thruways, and local roads, total number of intersections and number of intersections with traffic lights and stop signs, number of left and right turns, and distance along "familiar" roads.

The user examines a route in more detail by clicking on its summary. The interface displays turn-by-turn directions and highlights the selected route on the map. If the selected route is satisfactory to the user, he or she clicks the accept button, and the system updates its user preference model. If not, the user can perform a number of "tweaks" on the route to improve some attributes of the route, such as number of turns, at the expense of others, such as time. The user continues tweaking the route options until he or she is satisfied with one of them. The user may also generate a completely different route if none of the options are nearly satisfactory. Once the user accepts a route, the system saves his preferences for the duration of the conference.

Although interaction is in the user's best interest if he or she wants a satisfactory route, the agent does not require it, and ideally interaction will become less necessary as the agent better approximates the user's cost function. This low interaction requirement is crucial for in-car decision making where the driver's attention is necessarily focused elsewhere.

This system is an example of an adaptive user interface agent that automatically and unobtrusively acquires value judgments by observing the user's actions in a domain, and utilize interaction as an additional source of value judgments. The agent generates a solution using its current user model, receives feedback from the user if its model is inaccurate, and corrects its model in areas relevant to the problem being solved.

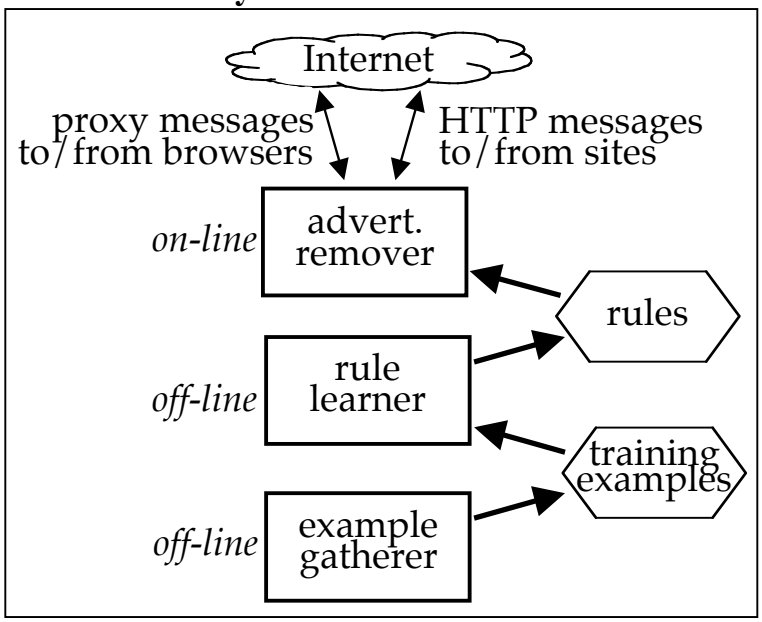
Demonstration
Learning to remove Internet advertisements

Nicholas Kushmerick
Department of Computer Science, University College Dublin
nick@ucd.ie

Abstract. *AdEater* is a fully implemented browsing assistant that automatically removes advertisement images from Internet pages. Unlike related systems that rely on hand-crafted rules, *AdEater* takes an inductive learning approach, automatically generating rules from training examples. Our experiments demonstrate that our approach is practical: the off-line training phase takes less than six minutes; on-line classification takes about 70 msec; and classification accuracy exceeds 97% given a modest set of training data. *To use AdEater, set your browser proxy to* ***cygnus.ucd.ie:8888****; see* ***www.cs.ucd.ie/staff/nick/research/ae*** *for details.*

Before ... ***... after.***

System Architecture



http://www.provider.com/index.html

```
                    ...
<A href="http://www.corp.com/sales.html">
  Our sponsor: <IMG src="http://www.corp.com/ads/thead.gif"
                    alt="click here" height="40" width="200"></A>
                    ...
<A href="contact.html">
  Contact us: <IMG src="/images/contact.gif"
                   alt="contact info" height="50" width="40"></A>
                    ...
<A href="http://www.mega.com/marketing.html">
  Funded by: <IMG src="http://www.mega.com/adverts/adimg.jpg"
                   alt="free stuff"></A>
                    ...
```

A, B, C

A	B	C	Feature	
40	50	?	**height**	
200	40	?	**width**	
5.0	0.8	?	**aspect ratio**	
0	0	1	**local?**	
1	0	0	"our"	**caption** features
1	0	0	"sponsor"	
1	0	0	"our+sponsor"	
0	1	0	"contact"	
0	1	0	"us"	
0	1	0	"contact+us"	
0	0	1	"funded"	
0	0	1	"by"	
0	0	1	"funded+by"	
1	0	0	"free"	
1	0	0	"stuff"	
1	0	0	"free+stuff"	
0	1	0	"contact"	**alt** features
0	1	0	"info"	
0	1	0	"contact+info"	
0	0	1	"click"	
0	0	1	"here"	
0	0	1	"click+here"	
1	1	1	"www.provider.com"	
1	1	1	"index"	U_{base} features
1	1	1	"index+html"	
1	0	0	"www.corp.com"	
1	0	0	"sales"	
1	0	0	"sales+html"	
0	1	0	"contact"	
0	1	0	"contact+html"	U_{target} features
0	0	1	"www.mega.com"	
0	0	1	"marketing"	
0	0	1	"marketing+html"	
1	0	0	"www.corp.com"	
1	0	0	"ads"	
1	0	0	"ads+thead"	
1	0	0	"thead"	
1	0	0	"thead+gif"	
0	1	0	"images+contact"	
0	1	0	"images"	
0	1	0	"contact"	U_{img} features
0	1	0	"contact+gif"	
0	0	1	"www.mega.com"	
0	0	1	"adverts"	
0	0	1	"adimg"	
0	0	1	"adverts+adimg"	
0	0	1	"adimg+jpg"	
AD	**$\overline{AD}$**	**AD**	**Classification**	

If **aspect ratio** > 4.5833, **alt** doesn't contain '*to*' but does contain '*click+here*', and U_{dest} doesn't contain '*http+www*', then instance is an **advertisement**.

Adele: A Web-Based Pedagogical Agent

W. Lewis Johnson, Erin Shaw, and Rajaram Ganeshan

Center for Advanced Research in Technology for Education (CARTE)
USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 902-6695 USA
Email: {johnson, shaw, rajaram}@isi.edu; Web: http://www.isi.edu/isd/carte/

1. INTRODUCTION

This demonstration will introduce Adele (Agent for Distributed Learning Environments), an animated pedagogical agent designed to complement Web-based courseware. Adele appears as an animated figure that monitors the student's activities, provides hints, critiques inappropriate student actions, tests the students' mastery of the course material, and evaluates the students' performance. She thus is a kind of intelligent tutoring system (Wenger 1987). At the same time, Adele is a kind of autonomous agent, and shares important characteristics with autonomous agents in other domains. She operates in a complex environment, including complex simulations and one or more students. She is an interface agent, capable of communication with students using a combination of verbal and nonverbal gestures and body language.

The Web is a rich environment for instructional agents, since large amounts of instructional material is becoming available in the Web environment. Agents are needed to help students find relevant course materials. They also can help overcome some of the limitations of Web-based instruction by making it more interactive and engaging. At the same time, Web-based delivery imposes a number of design constraints. Fewer interaction modalities are available than for immersive environment agents such as Steve (Johnson et al 1998, Rickel and Johnson 1999). The agents need to be lightweight, so that they can run on client machines. They need to interoperate with Web servers and browsers.

2. ARCHITECTURE

Adele's system consists of four main components: the pedagogical agent, the simulation, the client-server, and the server store. The pedagogical agent consists further of two sub-components, the animated persona and the reasoning engine. A fifth component, the session manager, is employed when the system is run in multiple-user mode. The central server is used to maintain a database of student progress and when appropriate, to provide synchronization for collaborative exercises being carried out by multiple students on multiple computers.

The reasoning engine performs all monitoring and decision making. Its decisions are based on a student model, a case task plan, and an initial state, which are downloaded from the server when a case is chosen, and on the agent's current mental state, which is updated as a student works through a case. Upon completion, a record of the student's actions is saved to the server where it is used to assess the level of the student's expertise and determine how Adele will interact with the student in future cases.

The animated persona is simply a Java applet that can be used alone with a Web page-based JavaScript interface or incorporated into larger applications, such as simulation-based exercises. The persona is capable of expressing a range of gestures, such as pointing, nodding, changing focus of gaze, and speaking.

The simulation can be authored using the language or authoring tool of one's choice. For our clinical health science applications, we developed our own simulation in Java. For a trauma care application we used Emultek's RAPID simulation authoring tool. Adele expects the simulation to notify Adele of user events and simulation state changes, so that she and interpret and respond to them appropriately.

The integrated system is downloaded to and run on the client's side for execution efficiency. This is in contrast to the architecture of most other Web-based Intelligent Tutoring Systems where the intelligent tutor sits on the server side, resulting in increased latency in tutor response to student actions (e.g., Brusilovsky et. al 1997). Reducing latency is especially critical when one considers *animating* an agent's response to a student's action, in order to achieve the perception of awareness in a shared workspace.

3. KEY CAPABILITIES

3.1 Situation and Student Monitoring

Adele must keep track both of the actions that the student performs and the events that occur in the simulation, in order to provide appropriate advice and feedback. Student monitoring makes reference to a task plan, which is an abstract description of an expected solution, in a hierarchical nonlinear plan format. For example, in clinical medicine the task plan represents a standard clinical procedure, specialized to the particular case being studied.

Some Adele exercises, such as those for trauma care, involve dynamic simulations in which students may need to reprioritize their actions from moment to moment. Adele adopts the situation space approach (Marsella and Schmidt 1990, Marsella and Johnson 1998) to model changing situations. Situation spaces include situations, which are state patterns matched against the simulation state, and transitions between situations. Task subplans are associated with particular situations, and are dynamically added to the task plan.

Although the task plan approach is general enough to apply to a range of subject matter areas, it is limited in its ability to support guidance and feedback. In clinical decision making, it is important for students to weigh the evidence that they encounter and at the same time follow clinical procedures. Task plans are good at modeling clinical procedures, but are less suitable for modeling analysis of evidence. For this purpose we have built an extension to Adele's student monitoring capability to model diagnostic reasoning, for use in courses where it is relevant. As the student uncovers evidence, the

estimate of likelihood of possible hypotheses is adjusted as necessary.

3.2 Opportunistic instruction

As the student progresses through the exercise, situations might arise that are opportunities for instruction. For example, when the patient tells the student that her condition has been slowly developing, it might be useful to make sure that student can knows what kinds of diseases develop rapidly and what kinds do not. Adele monitors such opportunities as situations, similar to the situations used for tracking student progress. But instead of specifying student behavior, they specify Adele's pedagogical behavior. For example, they can cause Adele to present a quiz about slow-onset diseases when the student receives information about disease onset from the patient.

References to Web-based instruction are also opportunities for instruction. Adele takes advantage of these opportunities in two ways. Adele's interface has a Show button that becomes active in situations where relevant reference materials exist. Clicking on the button takes the student to a particular page associated with the situation. Additionally, the student's actions can provide context that is used to provide context-specific access to reference materials. The Reference page contains a set of references that changes dynamically as the student works through the case.

3.3 Student Assessment

When a task is finished, Adele's assessment module analyzes the student's record and provides domain-appropriate feedback. For example, in a clinical domain, Adele provides three types of post-task assessment: 1) An evaluation of the diagnosis; 2) an evaluation of the diagnostic costs incurred, and 3) an evaluation of the steps taken. Scoring of the student's record varies from course to course, dependent on the preferences of the individual instructor. Once the case is complete, it is uploaded to the instructor for further review.

4. IMPLEMENTATION

To facilitate Web-based delivery, Adele is implemented in Java, making it possible to download Adele-enhanced course modules over the Web. This approach offers long-term advantages, although in the near term, incompatibilities between Java virtual machines make portability somewhat difficult. High quality text-to-speech synthesis is platform-dependent, so speech packages are wrapped in Java and invoked locally.

Adele's animations are produced from two-dimensional drawings, making it possible to run Adele on a variety of desktops and operate in heterogeneous educational environments. We placed emphasis on improving Adele's appearance of reactivity, rather than emphasize photorealism. For example, when the student clicks on a button in the simulation window, Adele turns her head to look toward where the student clicked.

5. DEMONSTRATION HIGHLIGHTS

Multiple Adele-assisted learning modules will be demonstrated, depending upon the interests of the attendees. One learning module will be taken from a clinical medicine course. For example, a simulated patient comes to the clinic complaining of a lump on her chest. The student must ask the patient questions in order to obtain a patient history, examine the patient, order relevant tests, and come up with the appropriate diagnosis (in this case cancer). Along the way Adele quizzes the student to make sure that he or she understands the distinguishing characteristics of different diseases, and is able to interpret the disease findings correctly. A geriatric dentistry case will be available for demonstration as well, in which the student must both assess the patient's situation and prescribe treatments. Finally, a dynamic case simulation such as trauma care may be shown.

6. ACKNOWLEDGEMENTS

CARTE staff members Kate Labore and Dr. Jeff Rickel, 'A' Team members Ami Adler, Andrew Marshal, Anna Romero, and medical researcher Michael Hassler all contributed to the work presented here. Invaluable assistance was provided by our colleagues in the USC School of Medicine and School of Dentistry, particularly Drs. Riccardo Hahn, Clive Taylor, Beverly Wood, and Roseann Mulligan. This work was supported by an internal research and development grant from the USC Information Sciences Institute; course development was supported by grants from the USC Health Sciences Consortium and the USC School of Medicine.

7. REFERENCES

[1] Brusilovsky, P., Schwartz, E., and Weber, G., ELM-ART: An intelligent tutoring system on world wide web. Frasson, C., Gauthier, G. and Lesgold, A. (Eds.), Proc. of the Third Int'l Conf. on Intelligent Tutoring Systems, pp. 261-269. Springer Verlag, 1996.

[2] Johnson, W.L. and Rickel, J., Steve: An animated pedagogical agent for procedural training in virtual environments. SIGART Bulletin 8, pp. 16-21, 1998.

[3] Marsella, S.C. and Schmidt, C.F., Reactive planning using a situation space, Proc. of the 1990 AAAI Spring Symposium Workshop on Planning, 1990.

[4] Marsella, S.C. and Johnson, W.L., An instructor's assistant for team-training in dynamic multi-agent virtual worlds. Goettl, B.P., Halff, H.M, Redfield C.L., and Shute, V.J. (Eds.), *Proc. of the Fourth Int'l Conf. on Intelligent Tutoring Systems* pp.464-573. Springer-Verlag, 1998.

[5] Rickel, J. and Johnson, W.L., Animated agents for procedural training in virtual reality: perception, cognition, and motor control. To appear in *Applied Artificial Intelligence Journal*, 1999.

[6] Shaw, E., Ganeshan, R., Johnson, W.L., and Millar, D., Building a case for agent-assisted learning as a catalyst for curriculum reform in medical education. To appear in Proceedings of AI-Ed '99, IOS Press, Amsterdam.

[7] Wenger, E., Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge. Los Altos, CA: Morgan Kaufmann Publishers, Inc., 1987.

Agent aided aircraft maintenance*

Onn Shehory, Katia Sycara, Gita Sukthankar and Vick Mukherjee
School of Computer Science
Carnegie-Mellon University
{onn,katia,gitars,vick}@cs.cmu.edu

1 Introduction

Aircraft maintenance is performed by mechanics who are required, by regulation, to consult expert engineers for repair instructions and approval. In addition to their own experience, these engineers rely on external information sources, which are often inadequately indexed and geographically dispersed. The timely retrieval of this distributed information is vital to the engineers' ability to recommend repair procedures in response to the mechanics' requests. This problem domain is well suited for a multi-agent system: it consists of distributed multi-modal information which is needed by multiple users with diverse preferences. Using the RETSINA multi-agent architecture [2], we demonstrate an implementation of such a system. Such an implementation reinforces the importance of multi-agent systems, and in particular the usefulness of the RETSINA infrastructure as a basis for the construction of such systems.

The MAS we developed [1] provides information retrieval and analysis in support of decision making for aircraft maintenance and repair in the U.S. Air Force. Although the solution was developed for a specific type of aircraft, the agents and the interactions among them were designed to apply to a range of similar maintenance scenarios.

2 The problem

Standard aircraft maintenance in the U.S. Air Force involves the following: when inspecting an aircraft, a mechanic who indicates a possible discrepancy consults an engineer to decide on the required repair. The engineer, in turn, may need to consult external sources of information. These include manuals, historical maintenance data and other, remotely-located experts. Hard-copy repair manuals are used, thus searching for relevant information may be both time consuming and incomplete. Historical data (records of previous similar repairs) is scarcely used, since it is stored in paper format with no search mechanisms, usually only kept for short periods of time, and may be distributed along remotely located service centers. Expert engineers may be located remotely, and their advice is available by voice or fax messages. All of these factors contribute to a slow, inefficient inspection and maintenance process.

*Funding for this work has been provided by the ONR grant #N00014-96-1-1222.

The repair process consists of the following steps:

- An aircraft arrives at a maintenance service center for regularly scheduled maintenance, which must be completed within a limited time period.
- Mechanics inspect the aircraft and locate discrepancies. For each discrepancy a mechanic finds, he/she: (1) consults manuals and other, more experienced mechanics; (2) fills in a 202a form (a standard Air Force form for aircraft discrepancies); (3) sends 202a to an engineer for advice and authorization to perform repair;
- An engineer, upon receipt of a 202a form: (1) determines repair procedure for the discrepancy described in the 202a form; (2) fills in a 202b form (a standard Air Force form for repair instructions); (3) files 202a and 202b forms for future use.
- Upon receipt of a 202b form from an engineer, the mechanic performs the repair as instructed.

3 The system design

Information sources The sources of information relevant to the inspection and maintenance processes are: (1) the form filled in by the mechanic for the current fault encountered during inspection (current 202a form); (2) collections of historical 202 (a and b) forms from previous aircraft inspection, consultation and maintenance procedures; (3) technical manuals; (4) expert engineers.

Agent types We developed an agent system that provides information gathering, filtering and fusion in support of maintenance decisions. The system has been implemented and is tested with real data from a U.S. Air-base. The system is comprised of three types of agents, as follows:

- A *form agent*. Its role is to analyze the current 202a form it receives from a mechanic, characterize the problem presented in the form, and request information which is relevant for the solution of this problem. Upon the receipt of such information it merges, filters and presents it in a meaningful and comprehensible manner to the engineer.
- A *history agent*. Upon request from another agent (probably a form agent), it searches for historical forms which are relevant to the problem presented in the request. The relative relevance of the forms is computed

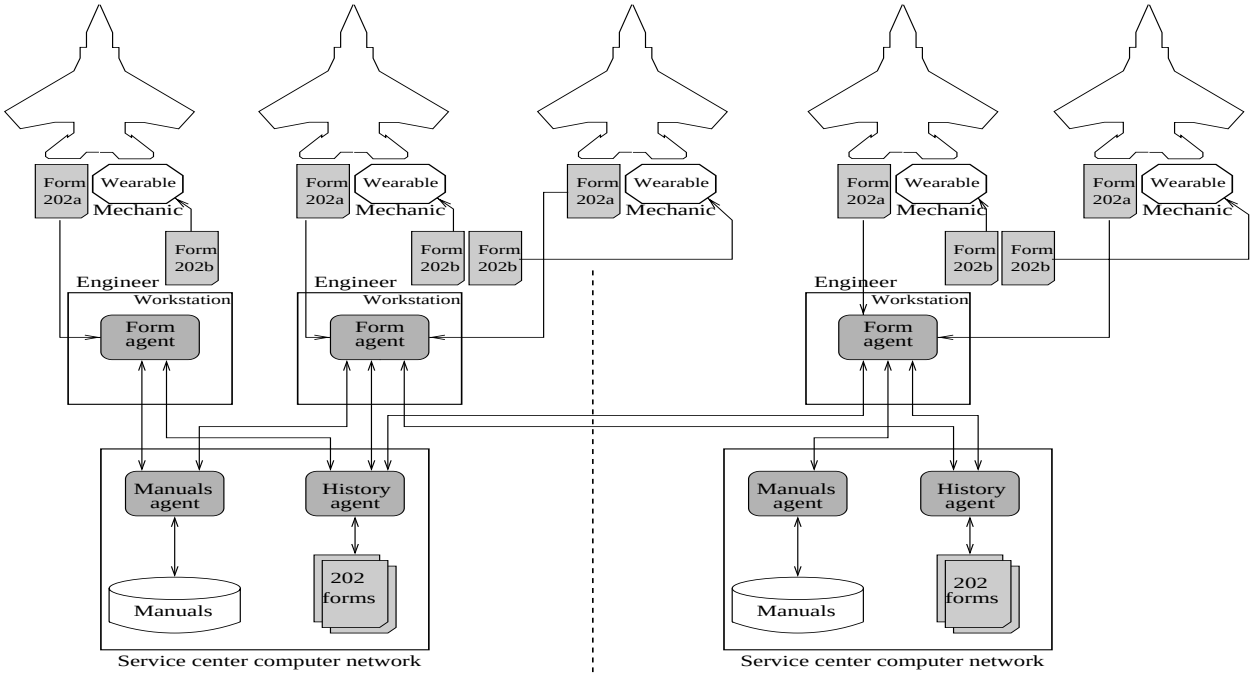


Figure 1: The multi-agent system organization.

and forms that pass some relevancy threshold are sent as an answer to the requester.

- A *manuals agent*. Upon request from another agent, it locates, in the manuals, data relevant to the problem presented in the request.

4 Data processing and flow

In our system, each specialized type of agent may have several instances. Below, we describe the processing and flow of information in the computerized system. The process begins with a mechanic inspecting the plane. The mechanic uses a wearable computer with a touch-screen, microphone and a digital camera. When a discrepancy is found, the mechanic fills in an electronic 202a form, and when necessary and practicable, adds voice notes and digital photographs. The 202a form with the attachments is sent to an engineer. At this point, the mechanic waits for a reply from the engineer.

The engineer, with the support of a form agent on his/her workstation, extracts keywords from the 202a form. Using these keywords, the form agent automatically requests relevant historical forms from history agents and relevant manual pages from a manuals agent. These requests may also be activated, controlled and edited by the user (engineer). At this point, the form agent waits for the requested information to arrive, in reply to its requests.

History and manuals agents are located on central computer networks of service centers, on which the archival information they need to access is located as well. Upon receiving a request for information, history agents perform a search on the historical 202 forms archive, and conduct a relevancy analysis. They reply with a list of relevant forms, the reason for their selection and the level of relevancy. A manuals agent performs a simple search in an indexed manuals database and replies with the results of this search.

Upon receiving replies from history and manuals agents, the form agent displays them to the engineer. Using this information the engineer can decide upon the appropriate repair procedure, fill in an electronic 202b form, attach to it graphical description grabbed from manuals and historical forms and send it to the mechanics wearable computer.

The information flow and processing end when the mechanic receives the 202b form on the wearable computer. The details in the 202b form and the approval of a repair procedure allow the mechanic to execute the actual repair.

5 Multi-agent organization

A graphical illustration of the multi-agent organization is presented in Figure 1. As depicted there, multiple mechanics each use a wearable computer in the inspection process to compose a 202a form. These forms are sent to form agents. There may be multiple form agents and each form agent may handle several 202a forms. A form agent may request information relevant to the forms it handles from multiple history agents. This is necessary since historical archives of 202 forms may be distributed over multiple service centers. Manuals agents and history agents may receive information requests from multiple form agents.

References

[1] O. Shehory, K. Sycara, G. Sukthankar, and V. Mukherjee. Agent aided aircraft maintenance. In *Proceeding of Agents-99*, Seattle, 1999.

[2] K. Sycara, K. Decker, A. Panu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert – Inteligent Systems and Their Applications*, 11(6):36–45, 1996.

Asynchronous Teams of Agents
for
Optimization and Decision-Support

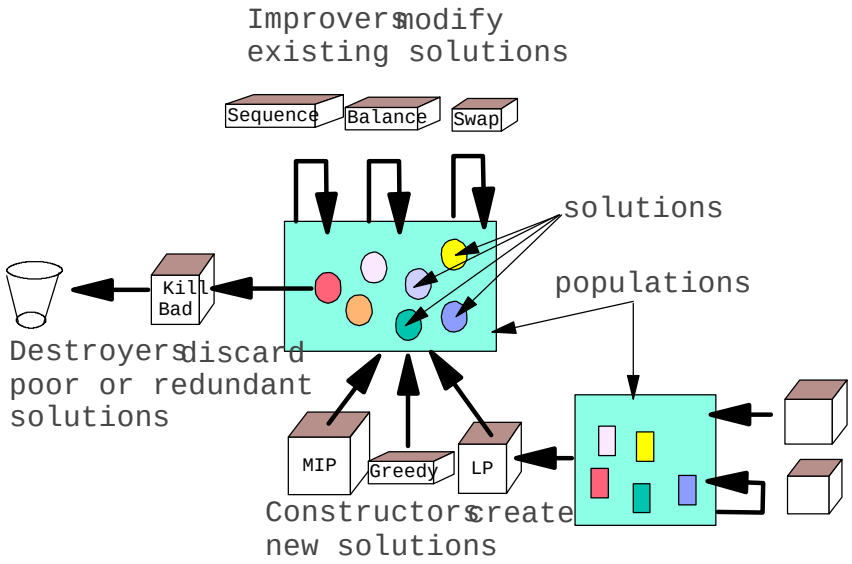
Richard Goodwin, Sesh Murthy, Rama Akkiraju, Fred Wu

IBM T. J. Watson Research Center

Yorktown Heights, NY

An asynchronous team of agents is a collection of software agents that cooperate to solve a problem by dynamically evolving a population of solutions (Talukdar et al, 1983). Agents cooperate by sharing access to populations of candidate solutions. Each agent works to create, modify or remove solutions from a population. The quality of the solutions gradually evolves over time as improved solutions are added and poor solutions are removed. Cooperation between agents emerges as one agent works on the solutions produced by another.

The figure below presents an overview of the A-Team architecture. Each agent, shown as a block, encapsulates a particular algorithm. This algorithm may consist of a call to an external system. Within an A-Team, agents are autonomous and asynchronous. Each agent encapsulates a particular problem-solving method along with the methods to decide when to work, what to work on and how often to work. These decisions are evaluation driven. An agent decides when to work and what to work on by looking at the evaluations of the solutions in the population. Agents are free to use any method for deciding when to work and what to work on, but intelligent strategies look at the state of the solutions in the population and take into account the agent's ability to improve them.

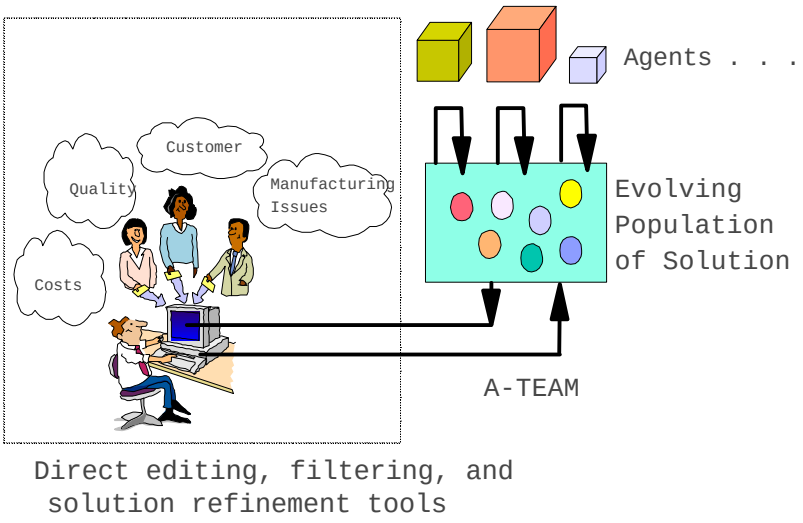


A-Teams consist of populations of solutions and agents that create, improve and destroy solutions.

Agents come in three flavors: constructors, improvers and destructors. *Constructors* create initial solutions and add them to the population. *Improvers* select one or more existing solutions from the population and produce new solutions that are added to the population. Technically, improvers are modifiers and may not actually make measurable improvements in the solutions they modify. They may, in fact, make random modifications that lead to worse solutions. Such modifications may be useful in that they serve to explore the solution space and, in so doing, may lead down a path to a better solution. Typically, however, improvers encapsulate domain specific methods designed specifically to effect quick directed improvement. Finally, d*estroyers* keep the size of the population of solutions in check. Their main function is to delete clearly sub-optimal or redundant solutions, while keeping promising solutions. This prevents improver agents from wasting effort by working on solutions that are going nowhere.

In our work, we have used A-Teams to solve multi-objective optimization problems such as production and transportation scheduling for the manufacturing domain. Each element of the solution population created by the A-Team is a complete solution to the problem such as a production schedule for a mill or a transportation schedule for a distribution center. The representation of the problem and the solution are problem specific but uniform throughout. For example, a machine scheduling A-Team might use a representation of the problem as a list of orders and machines and the solution as a sequenced assignment of orders to machines.

In order to promote cooperation between agents and human experts, our software allows the human to operate as an agent, creating, improving and destroying solutions in the population. This interaction is accomplished through our Java based user interface. This allows the decision maker and human expert to refine the solutions taking into account a number of competing prospectives.



Decision-maker as an agent in the A-Team

Demonstrations

To demonstrate the A-Team architecture, we have created three example domains, described below. Each example makes use of our agent library in C++ and our interface library in Java. Optimization algorithms for each example and solution editors were specifically created for each example.

Traveling Salesman Problem: This is a version of the classic traveling salesman problem where multiple local improvement algorithms are used to create nearly optimal tours. One twist is that we use both a population of complete solutions and a population of promising partial solutions.

Paper Trim Problem: This example is taken from paper manufacturing where large reels of paper (200 inches wide) must be cut to fill customer orders (roll widths vary from 15 to 100 inches). It is a variation on the on-dimensional cutting stock problem, with multiple objectives including minimizing loss, minimizing setups and maximizing on-time production.

Multiple Knapsack Problem: The objective of the problem is to pack items into containers in such a way that we maximize the weight of items packed into the containers while minimizing the volume of containers used. This is a variation on the classic knapsack problem and is relevant for industrial problems, such as loading items into trucks and rail cars.

References

Talukdar S.N., Pyo S.S., and Mehrotra R.; 1983. *Distributed Processors for Numerically Intense Problems.* Final Report for EPRI Project. **RP** 1983-1764-3

Murthy S., Akkiraju A., Goodwin R., Keskinocak P., Rachlin J., Wu F., Yeh J., Fuhrer R., Kumaram S., Aggarwal A., Sturzembecker M., Jayaraman R., Daigle R; 1999. *Cooperative Multiobjective Decision Support for the Paper Industry* .To appear in Interfaces.

The Bond Agent Framework - the technical content of the demo

January 31, 1999

The Bond distributed object system provides a message oriented middleware environment for developing distributed applications. Bond uses KQML as a metalanguage for inter-object communication. The message space of Bond is divided into *subprotocols*, task-oriented, closed set of messages. Examples of subprotocols are the property access, persistent storage access or security subprotocols. Bond objects can be extended with new subprotocols by the means of *probes*. Probes implement the functionality of a specific subprotocol and are attached as dynamic properties to Bond objects. A special class, called *preemptive probes* process an incoming message before it is delivered to the object, and act as filters for security, accounting, or logging purposes. Probes implement a way of *aspect oriented programming*.

Bond executables usually run as threads in the runtime environment provided by a *resident*. The resident provides the messaging thread and the local directory service for the running executables. Although Bond programs may run in stand-alone mode, their natural environment is a *Bond domain*. A domain contains a number of core servers such as the directory server, persistent storage server, authentication server and the monitoring agent.

The agent framework of the Bond system simplifies the task of developing agents by allowing the programmer to concentrate on the specific strategies of a new agent. Bond agents have the intrinsic capability to be controlled remotely and to cooperate with each other. The task of an application programmer is limited to specify the agenda, the finite state machine of the agent, and the strategies associated with each state.

The structure of the Bond agents presented in Figure 1. The components of a Bond agent are:

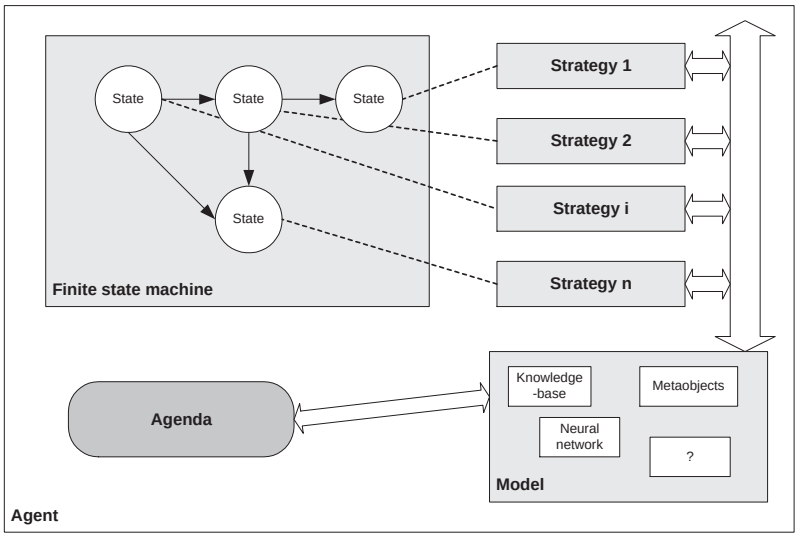


Figure 1: The anatomy of a Bond agent

- The **model of the world** is a container object which contains the information the agent has about its environment. This information is stored in the form of dynamic properties

of the model object. There is no restriction of the format of this information: it can be a knowledge base or ontology composed of logical facts and predicates, a pre-trained neural network, a collection of meta-objects or different forms of handles of external objects (file handles, sockets, etc).

- The **agenda** of the agent, which defines the goal of the agent. The agenda is in itself an object, which implements a boolean function on the model and a distance function on the model. The boolean function shows if the agent accomplished its goal or not. The agenda acts as a termination condition for the agents, except for the agents marked as having a *continuous agenda* where their goal is to maintain the agenda as being satisfied. The distance function may be used by the strategies to choose their actions.

- The **finite state machine** of the agent. The current state is a model variable named `STATE-1`. Each state has an assigned strategy which defines the behavior of the agent in that state. An agent can change its state by performing *transitions*. Transitions are triggered by internal or external *events*. External events are messages sent by other agents or programs. The set of external messages which trigger transitions in the finite state machine of the agent defines the *control subprotocol* of the agent.

- Each state on an agent has a **strategy** defining the behavior of the agent in that state. Each strategy performs actions in an infinite cycle until the agenda is accomplished or the state is changed. Actions are considered atomic from the agent's point of view, external or internal events interrupt the agent only between actions. Each action is defined exclusively by the agenda of the agent and the current model. A strategy can terminate by triggering a transition by generating an internal event. After the transition the agent moves in a new state where a different strategy defines the behavior.

The Bond agent framework can be programmed at two levels. At the expert level, the developer can define its own new strategies and agendas by programming them directly in Java. At the *blueprint* level, the user can create new agents using the blueprint language of the Bond agent framework. The blueprint is not a full featured programming language: the various aspects of agent strategies have to be programmed in Java. However, a database of ready-made strategies allows the most common aspects of agents to be assembled from components of this database, without the need of programming. The blueprint provides the assembly instructions used by the `bondAgentFactory` object to assemble the agent during runtime. The blueprint of an agent implicitly defines a *control subprotocol*, that can be used by an external object to control the agent.

Dynamic modification of agents ("agent surgery"). Bond agents can be modified dynamically during runtime. This is performed by the `AgentFactory` object using "surgical" blueprint scripts. New states, new transitions may be added, existing states or transitions deleted or new strategies added to existing states.

One of the many fascinating applications is the possibility that a coordinator agent changes it's federation of agents as a reply to new instructions from a human. The surgery of agents is a relatively inexpensive operation compared to starting new agents, and the agents are keeping their existing model, which may contain information difficult to restore in a new agent.

Migration. The behavior of a Bond agent is uniquely determined by the model of the world, which is a Bond object. The intrinsic property of Bond objects that they can be migrated between residents, imply that Bond agents can be migrated by simply interrupting the execution, transferring the model and recreating the agent from the same blueprint. The agent factories on the source and destination residents have an important role in the migration, performing the actual transfer of data. Nevertheless, it is possible that the model contains information which will become invalid after a migration - for example the handle of an open file. The agent factory can perform a check if the agent is migratable in the current status.

Links The Bond system is currently under implementation at the Bond Lab at Computer Science Department of Purdue University. More information and the second alpha version (as of January 1999) is available at `http://bond.cs.purdue.edu`

14

BTFS: The Border Trade Facilitation system[1]

Laurence R. Phillips (lrphill@sandia.gov)
Advanced Information Systems Lab
MS 0455
Sandia National Laboratories
Albuquerque, NM 87185

System Description:

We will demonstrate the Border Trade Facilitation System (BTFS), an agent-based bilingual ecommerce system built to expedite the regulation, control, and execution of commercial trans-border shipments during the delivery phase. The system was built to serve *maquila* industries at the US/Mexican border. The BTFS uses foundation technology developed here at Sandia Laboratories' Advanced Information Systems Lab (AISL), including a distributed object substrate, a general-purpose agent development framework, dynamically generated agent-human interaction via the World-Wide Web, and a collaborative agent architecture. This technology is also the substrate for the Multi-Agent Simulation Management System (MASMAS) proposed for demonstration at this conference. The BTFS executes authenticated transactions among agents performing open trading over the Internet. With the BTFS in place, one could conduct secure international transactions from any site with an Internet connection and a web browser. The BTFS is currently being evaluated for commercialization.

In 1997 the AISL completed a prototype of the Border Trade Facilitation System (BTFS), a collaborative information processing environment that operates on the Internet and World-Wide Web. The BTFS comprises multiple autonomous software agents that assist human actors in conducting international shipping transactions by creating, documenting, monitoring, and coordinating shipment transactions in information space.

The BTFS prototype demonstrates a multi-agent approach to coordinating a complex, knowledge-intensive shipping process. We have demonstrated the following agent behaviors: elicitation, mediation between ontologies, negotiation, delegation, monitoring, goal satisfaction, and conduct of an authenticated negotiation protocol for commercial contracts. A typical trans-border documentation package includes one to two dozen Spanish and English forms. The BTFS allows a registered user to fill out the core documentation set and execute the border crossing paperwork.

The essential concept of the BTFS is that the physical trans-border shipment of goods and the required accompanying certification are entirely represented as a set of events in information space, the state of which both controls and certifies events in physical space. The BTFS information system contains a real-time transaction-centric model of the physical border-crossing process. The BTFS design is based on

† This work was performed at Sandia National Laboratories, which is supported by the U.S. Department of Energy under contract DE-AC04-94AL85000

three general concepts: (1) creation of a distributed object programming environment with an underlying secure network infrastructure; (2) a distributed object representation of a shipping transaction; and (3) insertion of knowledgable software agents at critical points in the information flow.

The BTFS is supported by the AISL's distributed object programming system DCLOS (Distributed CLOS) that provides a seamless design methodology for networked object environments. DCLOS is essential to networking agents in a collaborative environment. DCLOS also supports a shared fragmented workpiece object. The information needed to effect a single shipment is captured in a complex distributed information structure with compositional semantics called the Maquila Enterprise Transaction (MET). The components of a given MET are distributed among the agencies involved in a particular shipment; no one agent or agency has access to all components. The MET is shared via proxy; when a given agent needs MET information, it is handed a MET proxy. Access is permitted based on task requirements and controlled by electronic signature. BTFS agents interact with the border-crossing process by collecting and organizing information and posting it in the MET. Control of the distributed computation is decentralized and opportunistic. Each agent computes new information components based on its internal knowledge base and the state of the MET . Changes in the components trigger computations in a manner reminiscent of blackboard systems

The framework comprises two associated abstract classes: *agent* and *agency*. An *agency* identifies an independent locus of processes, activities, and knowledge typically associated with some natural partitioning of the application domain. Agencies are collectives of agents that have ongoing high-level goals stated in business terms. In particular, the BTFS is a distributed set of agencies specialized on the commercial functions of the various stakeholders in the border-crossing process. The underlying assumption is that the application is naturally modeled as a group of interacting agencies, certainly true for the BTFS.

An *electronic commerce agency* (ECA) is a specialized subclass of the agencies class that implements architectural features specific to ecommerce applications. An ECA has the additional attributes of *transactions* and *organizations*. The transactions attribute holds a collection of open and closed transaction objects. The organizations attribute holds a collection of public proxy objects pointing to agencies that represent trading partners.

The BTFS agent society comprises several federated ECAs analogous to the interested business entities. Each ECA is populated by a heterogeneous collective of speciated agents, each of which is able to perform a fragment of the information tasks needed to effect trans-border shipment. Their exact duties are based on the idiosyncratic business rules of the actual businesses involved, so an operational ECA must be tailored and situated for each business. Constructing the ECA and the agents that make it up consists in specializing agents from a set of standard agent classes constructed for commerce. ECA classes are also pre-defined for the various required roles: originator, receiver, transport provider, and import/export broker.

A Collaborative Spoken-Language Desktop Agent Implemented with COLLAGEN

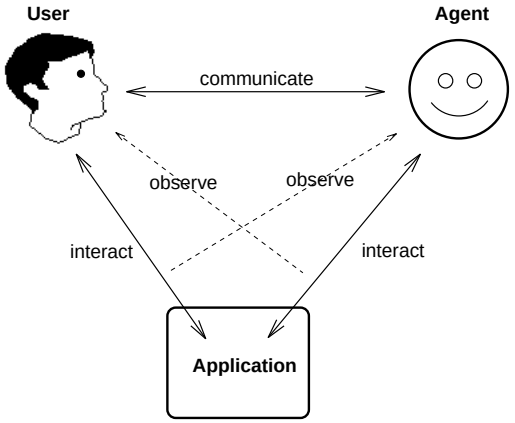
Charles Rich
MERL–A Mitsubishi Electric Research Lab
201 Broadway
Cambridge, MA 02139
rich@merl.com

Candace L. Sidner
Lotus Development Corporation
55 Cambridge Parkway
Cambridge, MA 02139
csidner@lotus.com

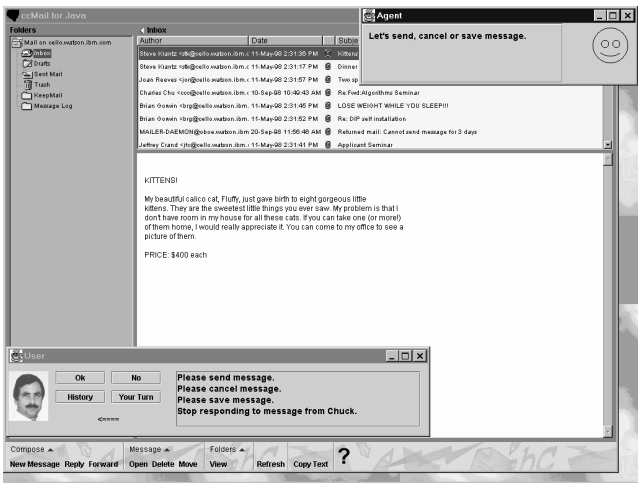
The underlying premise of the Collagen™(for *Coll*aborative *agen*t) project is that software agents, when they interact with people, should be governed by the same principles that govern human-to-human collaboration. To determine the principles governing human collaboration, we have relied on research in computational linguistics on collaborative discourse, specifically within the SharedPlan framework of Grosz & Sidner [1, 2, 4]. This work has provided us with a computationally-specified theory that has been empirically validated across a range of human tasks. We have implemented the algorithms and information structures of this theory in the form of a Java middleware component, a *collaboration manager* called Collagen, which software developers can use to implement a collaborative interface agent for any Java application.

In the *collaborative interface agent* paradigm, illustrated abstractly at the bottom left, a software agent is able to both communicate with and observe the actions of a user on a shared application interface, and vice versa. The software agent in this paradigm takes an active role in joint problem solving, including advising the user when he gets stuck, suggesting what to do next when he gets lost, and taking care of low-level details after a high-level decision is made.

The screenshot at the bottom right shows how the collaborative interface agent paradigm is concretely realized on a user's display. The large window in the background is the shared application, in this case, the Lotus eSuite™ email program. The two smaller overlapping windows in the corners of the screen are the agent's and user's *home windows*, through which they communicate with each other.



Collaborative interface agent paradigm.



Graphical interface for Collagen email agent.

A key benefit of using Collagen to build an interface agent is that the collaboration manager automatically constructs a structured history of the user's and agent's activities. This *segmented interaction history* is hierarchically organized according to the goal structure of the application tasks. Among other things, this history can help re-orient the user when he gets confused or after an extended absence. It also supports high-level, task-oriented transformations, such as returning to an earlier goal.

Collagen also includes *plan recognition* capabilities specially adapted to the collaborative interface agent paradigm [3]. The inclusion of plan recognition significantly reduces the amount of communication required of the user, since the agent can infer the intent of many user actions.

To apply Collagen to a particular application, the application developer must provide an abstract model of the tasks for which the application software will be used. This knowledge is formalized in a *recipe library*, which is then automatically compiled for use by the interface agent. This approach also allows us to easily vary an agent's level of initiative from very passive to very active, using the same task model.

We have developed prototype interface agents using Collagen for several applications, including air travel planning (see [5]), desktop activities, resource allocation, and industrial control. At Agents'99, we will demonstrate our collaborative interface agent for common PC desktop activities using email and calendar applications.

The desktop agent is the first Collagen-based agent we have built that supports spoken-language interaction. Our other agents avoided the need for natural language understanding by presenting the user with a dynamically-changing menu of expected utterances, which was generated from the current discourse state according to the predictions of the the SharedPlan theory. The desktop agent, however, incorporates a speech and natural language understanding system developed by IBM Research, allowing users to collaborate either entirely in speech or with a mixture of speech and graphical actions

References

(References [3] and [5] are available at http://www.merl.com/projects/collagen)

[1] B. J. Grosz and C. L. Sidner. Attention, intentions, and the structure of discourse. *Computational Linguistics*, 12(3):175–204, 1986.

[2] B. J. Grosz and C. L. Sidner. Plans for discourse. In P. R. Cohen, J. L. Morgan, and M. E. Pollack, editors, *Intentions and Communication*, pages 417–444. MIT Press, Cambridge, MA, 1990.

[3] N. Lesh, C. Rich, and C. Sidner. Using plan recognition in human-computer collaboration. In *Proc. 7th Int. Conf. on User Modelling*, Banff, Canada, June 1999.

[4] K. E. Lochbaum. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4), December 1998.

[5] C. Rich and C. Sidner. COLLAGEN: A collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction*, 8(3/4):315–350, 1998.

DESIRE Software Environment for Compositional Development of Multi-Agent Systems

Frances M.T. Brazier, Frank Cornelissen, Catholijn M. Jonker,
Lourens van der Mey, Jan Treur

Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
URL: http://www.cs.vu.nl /{frances,frankc, jonker,lourens,treur}. Email: {frances,frankc,jonker,lourens,treur}@cs.vu.nl

The compositional development method DESIRE (DEsign and Specification of Interacting REasoning components) for multi-agent systems supports system designers during the entire design process: from knowledge analysis through to automated prototype multi-agent system generation. The basic principles behind the DESIRE compositional development method are:

- different levels of design (conceptual design, detailed design and operational design)
- documentation of problem description (including the requirements) and design rationale
- compositionality (based on both process abstraction and knowledge abstraction levels)
- reusability (generic models of agents and tasks)
- formal semantics (based on temporal models)
- evaluation (a compositional verification method)

These are principles generally acknowledged to be of importance in both software engineering and knowledge engineering.

The conceptual design includes conceptual models for each individual agent and the interaction between agents. The detailed design of a system, based on the conceptual design, specifies all static and dynamic aspects of a system's knowledge and behaviour. Prototype implementations, the operational design, are automatically generated from the detailed design.

In DESIRE all three levels of design are supported by a (graphical) software environment, which includes libraries of both generic models and instantiated components. Generic agent models and generic task models help in structuring the process of system design.

The compositional development method DESIRE is supported by a software environment that includes tools to support system development during all phases of design. *Graphical design tools* support specification of conceptual and detailed design of processes and knowledge at different abstraction levels. A detailed design is a solid basis to develop an operational implementation in any environment. An *implementation generator* supports prototype generation of both partially and fully specified models. The code generated by the implementation generator can be executed in an *execution environment*. This execution environment can be centralized, or distributed: it can execute all agents on one server, or execute different agents on different servers.

Currently the method and environment are used in a number of research, industrial and educational environments, providing input for the evolutionary design of the method and software environment itself. The demo includes examples of design specification, as illustrated by Figures 1 and 2 below.

Key references

Brazier, F.M.T. , Dunin-Keplicz, B., Jennings, N.R. and Treur, J. (1995). Formal specification of Multi-Agent Systems: a real-world case. In: V. Lesser (Ed.), Proc. of the First International Conference on Multi-Agent Systems, ICMAS'95, MIT Press, Cambridge, MA, pp. 25-32. Extended version in: Int. Journal of Cooperative Information Systems, M. Huhns, M. Singh, (Eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.

Brazier, F.M.T., Jonker, C.M., and Treur, J., Principles of Compositional Multi-agent System Development. In: J. Cuena (ed.), Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98, 1998, pp. 347-360.

Hardware The software environment runs under Solaris, Linux, Windows 95, and Windows NT.

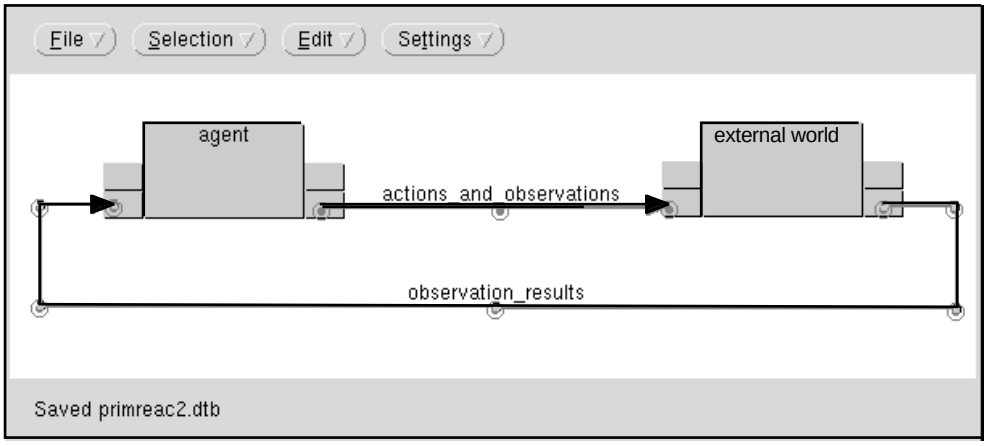


Figure 1 Graphical design tool for process composition

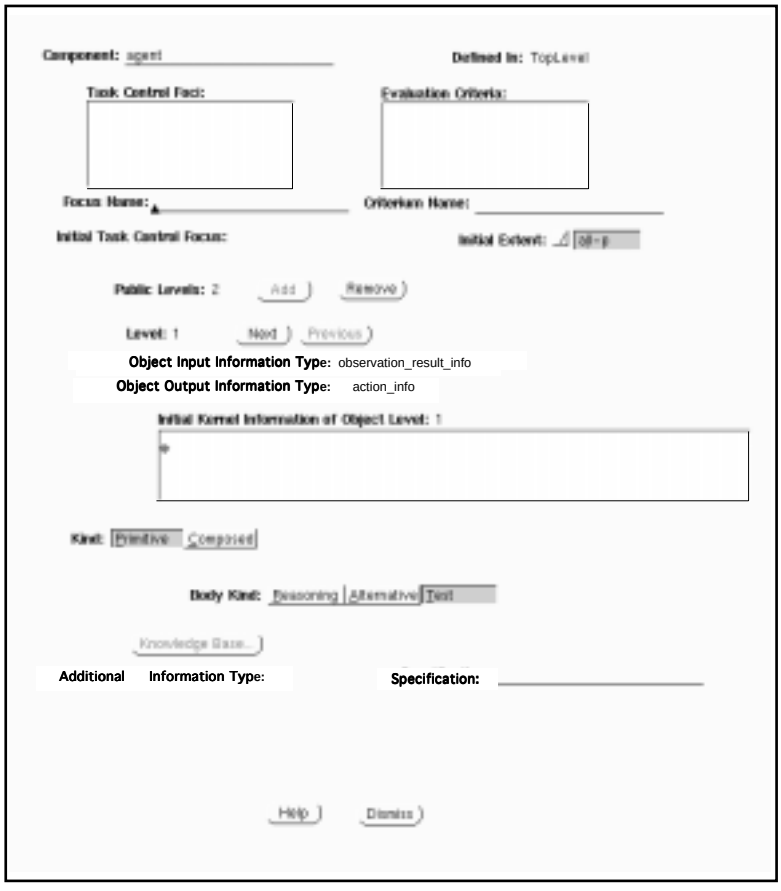
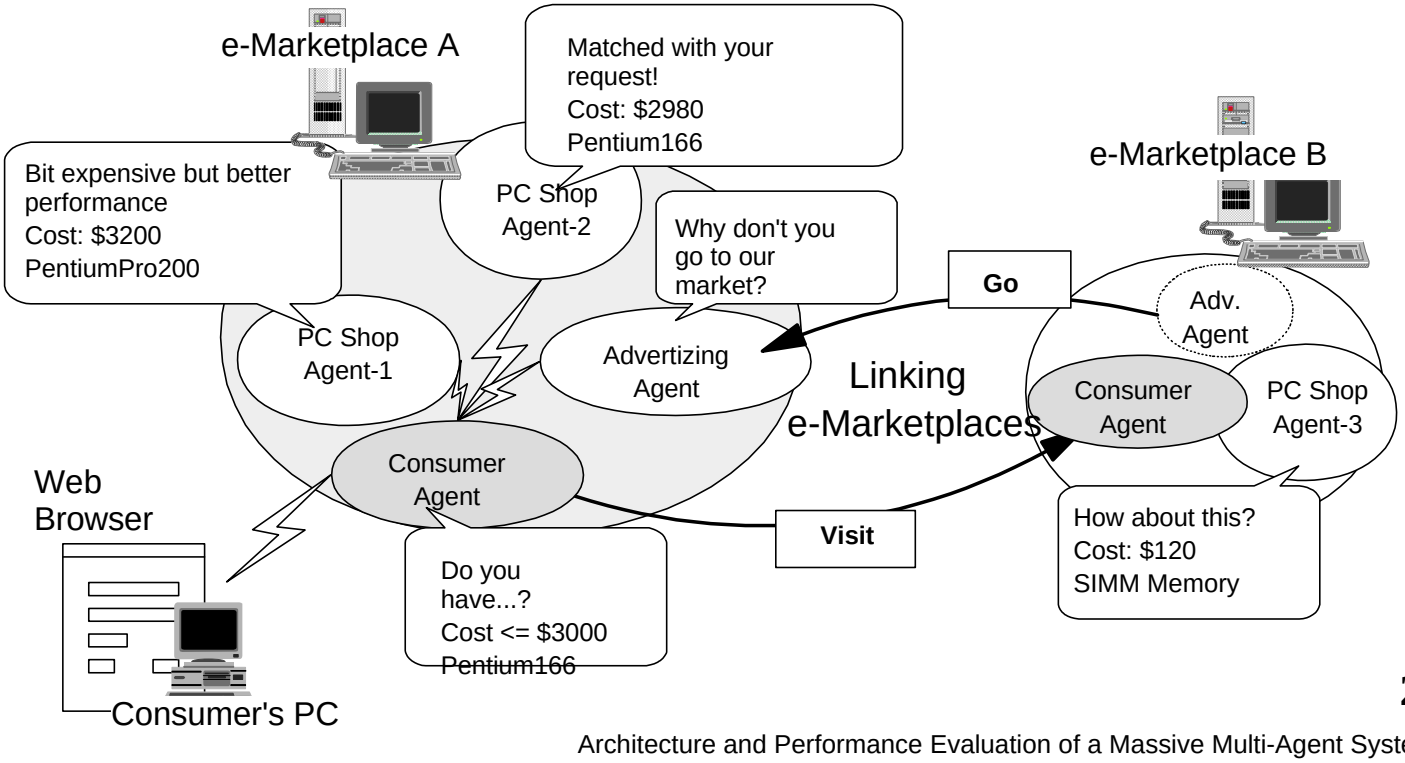


Figure 2 Component editing window

Agent-based Electronic Mall e-Marketplace

e-Marketplace

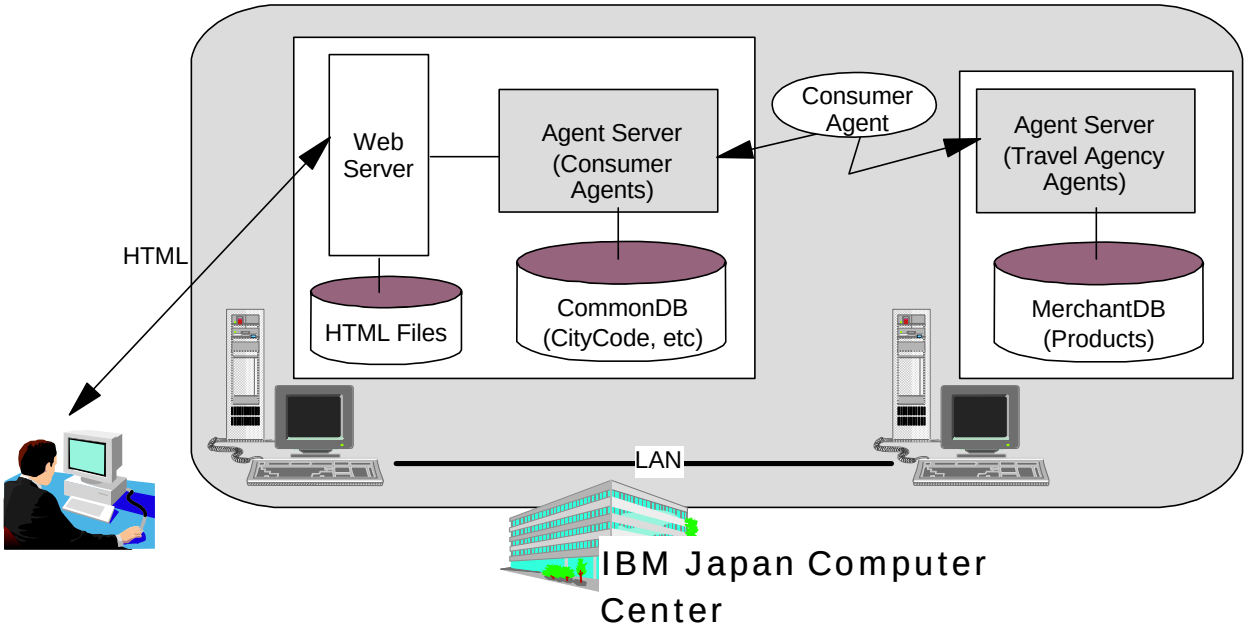
- Agent execution environment, within which consumer and merchant agents are situated
  - Agents interact with each other
    - Consumer agents search products
    - Merchant agents provide their products
  - Each agent has its own policy
    - Merchant agents may provide products that do not meet consumer requirements, and products of different types
    - Consumer agents may filter out products and merchant agents
    - Market Ad. agents may advertize their marketplace address
- Multiple e-Marketplaces comprises a virtual community
  - Agents roam around e-Marketplaces to meet others



"TabiCan" , a commercial site for travel information using e-Marketplace

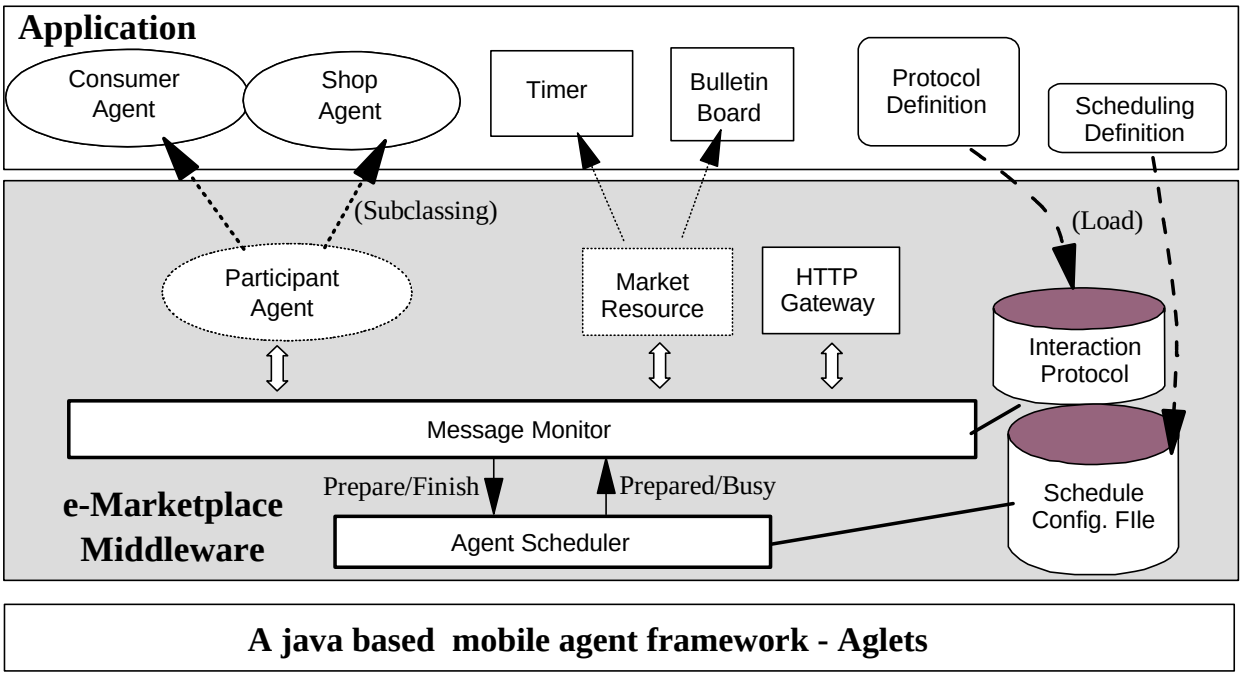
http://www.tabican.ne.jp/ (Japanese only)

- Hosting service for travel agencies by IBM Japan
- Providing airline tickets and package tours (airline and hotels) information for consumers



Architecture of e-Marketplace

- Manage agent interaction based on interaction protocols defined in a XML file.
- Schedule agent activities to host thousands of agent in a single agent server.



FM 1.00 A test-bed for Trading Agents in e-Auctions

Juan A. Rodríguez-Aguilar, Francisco J. Martín, Miguel Mateos
Oscar Molina, Pere Garcia, Carles Sierra

Institut d'Investigació en Intel.ligència Artificial

Campus de la Universitat Autonòma de Barcelona

08193 Bellaterra Spain

email: {jar,martin,sierra}@iiia.csic.es

Abstract

Auction-based e-commerce is an increasingly interesting domain for developing trading agents competing in multi-agent electronic markets. We present a framework for defining trading scenarios based on fish market auctions. In these scenarios, trading (buyer and seller) heterogeneous (human and software) agents of arbitrary complexity participate in e-auctions under a collection of standardized market conditions and are evaluated against their actual market performance. Such competitive situations constitute convenient problem domains in which to study issues related with agent architectures in general and trading strategies in particular. The proposed framework, FM, constitutes a test-bed for trading agents in auction tournament environments.

1 Introduction

Internet is spawning many new markets. In this sense, we observe that the proliferation of on-line auctions in the Internet—such as Auctionline (http://www.auctionline.com), Onsale (http://-www.onsale.com), InterAUCTION (http://www.interauction.com), eBay (http://www.eBay.com), and many others — has established auctioning as a main-stream form of electronic commerce. Thus, agent-mediated auctions appear as a convenient mechanism for automated trading, due not only to the simplicity of their conventions for interaction when multi-party negotiations are involved, but also to the fact that on-line auctions may successfully reduce storage, delivery or clearing house costs in many markets. This popularity has spawned research and development in agent-mediated auction houses as well as in trading agents endowed with intelligent auction strategies.

The matter of trading within an auction house appears to be numbingly complex, because of the numerous variables coming into play. The actual conditions for deliberation are not only constantly changing and highly uncertain—new goods become available, buyers come and leave, prices keep on changing; no one really knows for sure what utility functions other agents have, nor what profits might be accrued — but on top of all that, deliberations are significantly time-bounded. Hence there is the intricate matter of providing agent developers with some support to help them face the arduous task of designing, building, and tuning their trading agents before letting them loose in wildly competitive markets.

The FishMarket project[6] conducted at the Artificial Intelligence Research Institute (IIIA-CSIC) attempts to contribute in that direction by developing FM, an agent-mediated electronic auction house which has been evolved into a test-bed for electronic auction markets. The resulting framework, FM[1], constitutes an example of an agent-mediated electronic institution in the sense proposed in [5]. Conceived and implemented as an extension of FM96.5[3] (a Java-based version of the Fishmarket auction house), FM allows to define auction-based trading scenarios. It provides the framework wherein agent designers can perform *controlled experimentation* in such a way that a multitude of experimental market scenarios—that we regard as *tournament* scenarios due to the competitive nature of the domain— of varying degrees of realism and complexity can be specified, activated, and recorded; and trading (buyer and seller) heterogeneous (human and software) agents compared, tuned and evaluated. We argue that such competitive situations constitute convenient problem domains in which to study issues related with agent architectures in general and auction strategies in particular.

23

2 System Features

The current version of FM, FM1.00[4], is now available and can be downloaded from the FishMarket project web page. Next, we summarize the most salient features of this very first release:

- FM is completely written in Java.
- The customizability of FM allows for the specification, and subsequent activation, of a large variety of market scenarios: from simple toy scenarios to complex real-world scenarios, i.e., from extremely simple market scenarios in which the same auction is repeated over and over till market scenarios that make FM behave like the actual market. This capability of scenario generation allows the repeatability of the experiments (tournaments) to be conducted.
- FM supports an easily extensible library of auction protocols (English, Dutch, First Price Sealed bid, and Vickrey).
- FM is multi-user. It allows multiple users to spawn their agents in their own machines in order to make these to participate in remote tournaments.
- FM remains architecturally–neutral since no particular agent architecture (or language) is assumed or provided for building trading agents. Alternatively, a library of agent templates written in Java, C, and Lisp accompanies this release in order to assist agent programmers to build their agents. In this way, the programming effort narrows down to developing auction strategies. Importantly, these templates handle the connection of the trading agent to the FM interagents: autonomous software agents which intermediate the communication between the trading agents and the institution, the market, enforcing them to follow the rules of the game[2].
- A built-in agent builder facility allows for the automatic generation of agents with customizable auction strategies, so that families of agents capable of simulating different trading behaviours can be easily created.
- Auctions can be monitored step-by-step thanks to the FM Monitoring Agent. This keeps track of every single event taking place during a tournament in order to obtain a visual, global representation of the agents' flow from scene to scene within the market as well as the communication flow (what the agents utter and when).
- The FM database stores the information to be used by trading agents to carry out market analysis and auditing.
- FM has been designed to be as user-friendly as possible. Thus, the FM GUI allows the whole interaction between the users and FM to be done through graphical interfaces.

References

[1] Juan A. Rodriguez-Aguilar, Francisco J. Martin, Pablo Noriega, Pere Garcia, and Carles Sierra. Competitive scenarios for heterogenous trading agents. In *Second International Conference on Autonomous Agents (AGENTS'98)*, pp. 293–300, 1998.

[2] F. J. Martín, E. Plaza, and Juan A. Rodríguez-Aguilar. An Infrastructure for Agent-based Systems: An Interagent Approach. In *International Journal of Intelligent Systems* (to appear).

[3] Juan A. Rodríguez-Aguilar, P. Noriega, C. Sierra, and J. Padget. Fm96.5 a java-based electronic auction house. In *Second International Conference on The Practical Application of Intelligent Agents and Multi-Agent Technology(PAAM'97)*, pp. 207–224, 1997.

[4] Juan A. Rodríguez-Aguilar, F. J. Martín, Oscar Molina, Miguel Mateos. FM 1.00 Users Guide. Institut d'Investigació en Intel.ligència Artificial. Technical Report, 1999.

[5] Juan A. Rodríguez-Aguilar, Francisco J. Martín, Pere Garcia, Pablo Noriega, and Carles Sierra. Towards a Formal Specification of Complex Social Structures in Multi-agent Systems. In *Lecture Notes in Artificial Intelligence*,1999 (to appear).

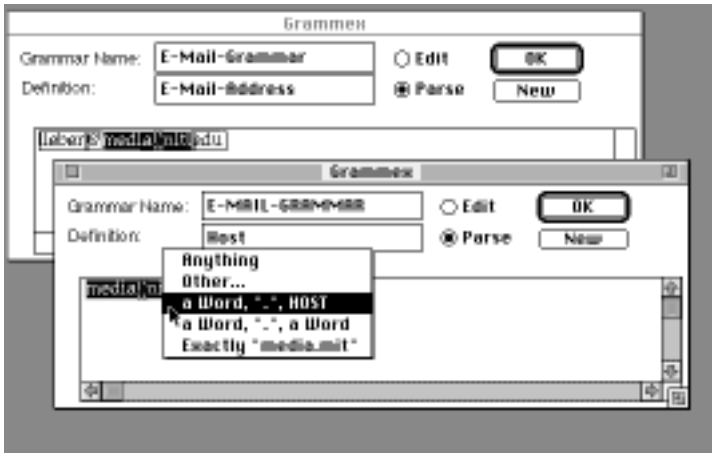
[6] The FishMarket Project. http://www.iiia.csic.es/Projects/fishmarket.

Training Agents to Recognize Text by Example (Demo)

Henry Lieberman
Media Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139 USA
(1-617) 253-0315
lieber@media.mit.edu

Bonnie A. Nardi
AT&T Labs West
75 Willow Road
Menlo Park, CA 94025
(1-650) 463-7064
nardi@research.att.com

David Wright
Apple Computer
1 Infinite Loop
Cupertino, CA 95014 USA
(1-408) 974-6018
dave.wright@apple.com



1. ABSTRACT

An important function of an agent is to be "on the lookout" for bits of information that are interesting to its user, even if these items appear in the midst of a larger body of unstructured information. But how to tell these agents which patterns are meaningful and what to do with the result?

Especially when agents are used to recognize text, they are usually driven by parsers which require input in the form of textual grammar rules. Editing grammars is difficult and error-prone for end users. *Grammex* ["Grammars by Example"] is the first direct manipulation interface designed to allow non-expert users to define grammars interactively. The user presents concrete examples of text that he or she would like the agent to recognize. Rules are constructed by an iterative process, where Grammex heuristically parses the example, displays a set of hypotheses, and the user critiques the system's suggestions. Actions to take upon recognition are also demonstrated by example.

2. Grammex: A demonstrational interface for grammar definition

Grammex is the interface we have developed for defining grammars from examples. It consists of a set of Grammex rule windows, each containing a single text string example to be used as the definition of a single grammar rule. Text may be cut and pasted from any application. The task of the user is to create a description of that example in terms of a grammar rule.

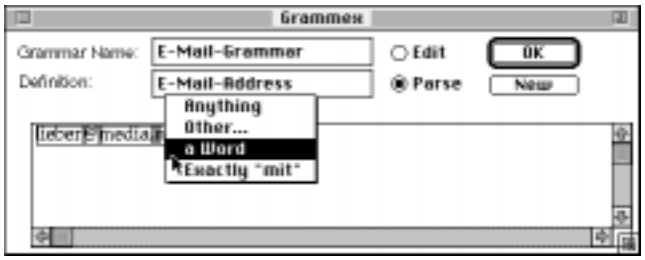
Grammex parses the text string according to the current grammar, and makes mouse-sensitive the substrings of the example that correspond to grammar symbols in its interpretation. Clicking on one of the mouse-sensitive substrings brings up a list of heuristically computed guesses of possible interpretations of that substring. The user can select sets of adjacent substrings to indicate the scope of the substring to be parsed. At any time, a substring can be designated as a new example, spawning a new Grammex rule window, supporting a top-down grammar definition strategy.

There is also an overview window, containing an editable list of the examples and rules defined so far.

2.1 An example: Defining a grammar for e-mail addresses

We start defining the pattern for E-Mail-Address by beginning with a new example that we would like to teach the system to

handle. We get a new Grammex rule window, and type in the name for our grammar, E-Mail-Grammar, the name of the definition, E-Mail-Address, and the example text *lieber@media.mit.edu*. In Parse Mode, Grammex tries to interpret the text in the example view, and the user can interactively edit the interpretation. Grammex makes pieces of the text mouse sensitive. Initially, "lieber", "@", "media", ".", "mit" "." and "edu" are identified as separate pieces of text, using the parser's lexical analysis. Each displays a box around it. Clicking on a piece of text brings up a popup menu with Grammex's interpretations of that piece of text. Here, the user clicks on "mit".



Interpretations of the string "mit"

2.2 Top-down definition: examples can spawn new examples

The concept of a Host is more complex than that of a person, because we can have hosts that are simply names, such as the machine named "media", or we can have hosts that consist of a path of domains, separated by periods, such as "media.mit.edu". Thus, the definition for Host requires two examples: one of each important case.

We describe "media" as being an example of a Host being a single word, in the same way we did for "lieber" as a Person. Note that when we choose the word "media" the possible interpretation [plausible, but wrong] of "media" being a Person crops up.

"media" is a Host

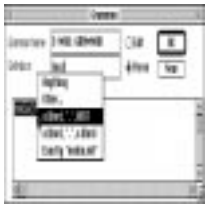
2.3 Definition of recursive grammar rules

The second example for a Host describes the case where there is more than one component to the host name, for example "media.mit". We select the substring "media.mit" from our original example, "lieber@media.mit.edu", and invoke New.

The default interpretation of "media.mit" would be as a Word, followed by a ".", followed by another Word. However, while this is a possible interpretation, it does not describe the general case in such a way as could accommodate any number of host components. For that, we need to express the idea that following a "." we could then have another sequence of a word, then another "." then a word.... that is, we could have another Host. In our example, then, we need to change the interpretation of "mit" to be a Host rather than a word, so that we could have not just "media.mit" but also "media.mit.edu", "media.mit.cambridge.ma.us", etc. This is done by simply selecting "mit" and choosing the interpretation Host from the popup menu.

A Host is recursively defined

The result is now that if we ask what the interpretation of " media.mit" is, we get a Word, then ".", then a Host.



Verifying the interpretation of "media.mit"

This is an important and subtle idea, the concept of defining a *recursive* grammar definition through multiple examples.

Verifying "lieber@media.mit.edu"

3. REFERENCES

[1] Lieberman, H., Nardi, B., and Wright, D. Training Agents to Recognize Text by Example, International Conference on Autonomous Agents [Agents-99], Seattle, May 1999.

PROJECT JAMES
A Mobile Agent Platform for the Management of Telecommunication and Data Networks

This project is inserted in the area of Information and Telecommunication Technology and its main goal is to apply the concept of Mobile Agents to the Management of Telecommunication Systems and Data Networks.

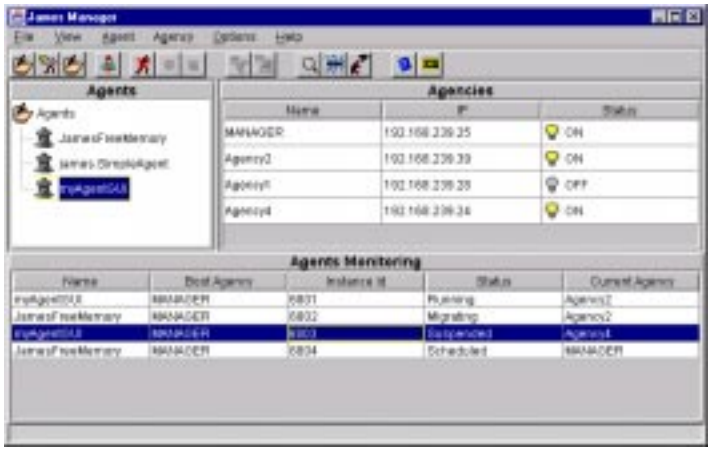
A Consortium was created mixing the experts on Mobile Agents and Java - University of Coimbra - and experts in Telecommunications and in the real knowledge about the market needs which are Siemens SA and Siemens AG, respectively.

University of Coimbra, Portugal

SIEMENS

SIEMENS Portugal SA
SIEMENS AG

Eureka Project: E!1921



- JAMES Agencies (that support the execution of agents);
- Support for remote upgrading of Agents and Agencies;
- Agent monitoring and profiling;
- Efficient agent migration;
- Fault-tolerance mechanisms;
- Reconfigurable itinerary;
- Caching and Prefetching mechanisms to optimize the agent migration;
- Support for parallel execution;
- Disconnected computing;
- Java-based SNMP implementation;

Contact: Luis Moura Silva
Affiliation: University of Coimbra, Portugal
Email: luis@dei.uc.pt
Web page: http://james.dei.uc.pt

LARKS: Matchmaking Among Software Agents in CyberSpace*

Katia Sycara and **Seth Widoff**
The Robotics Institute, Carnegie Mellon University, Pittsburgh, USA.
{katia, swidoff}@cs.cmu.edu

1 Introduction

One of the basic problems facing designers of open, multi-agent systems for the Internet is the connection problem, that is, finding the other agents that may offer the services that an agent needs. *Middle agents*(Decker, Sycara, and Williamson 1997), such as matchmakers, brokers, billboards, etc. have been proposed as a way to solve the connection problem and allow service *requester agents* to find service *providers* with desired capabilities. Since, in general, agents are heterogeneous, there is a need for standardized ways for agents to communicate their services and requests. We present LARKS (Language for Advertisement and Request for Knowledge Sharing) that allows agents to express service capabilities and requests, as well as a powerful matchmaking algorithm that allows different types of partial matches. LARKS and the matchmaking algorithms have been implemented and are currently extensively testedand incorporated within our RETSINA multi-agent infrastructure framework (Sycara, et al. 1996).

The following figure shows the user interface of the matchmaker agent.

Matchmaking is the process of finding an appropriate provider for a requester through a middle agent, and has the following general form: (1) Provider agents advertise their capabilities to middle agents, (2) middle agents store these advertisements, (3) a requester asks some middle agent whether it knows of providers with desired capabilities, and (4) the middle agent matches the request against the stored advertisements and returns the result, a subset of the stored advertisements. [1].

While this process at first glance seems very simple, it is complicated by the fact that not only local information sources but even providers and requesters in the Cyberspace are usually heterogeneous and incapable of understanding each other. This gives rise to the need for a common language for describing the capabilities and requests of software agents in a convenient way. In addition, it is necessary to have an efficient mechanism to determine a structural and semantic match of descriptions in that language.

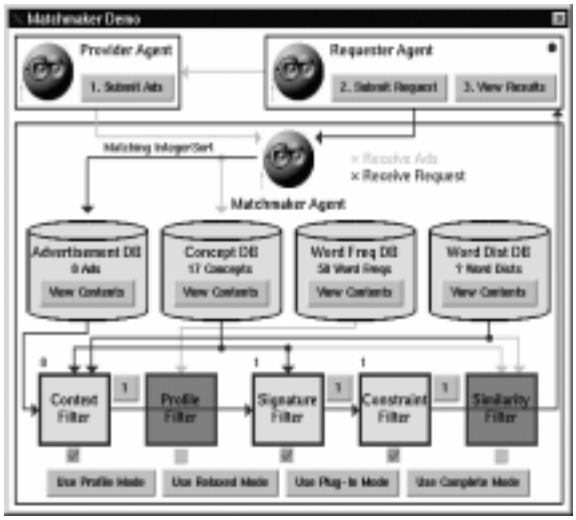


Figure 1: The User Interface of the Matchmaker Agent.

2 The Agent Capability Description Language LARKS

There is an obvious need to describe agent capabilities in a common language before any meaningful service matchmaking or brokering among the agents can take place. Some of the main desired features of such a language are the following:

- **Expressiveness** The language should be expressive enough to represent not only data and knowledge, but also the meaning of program code. Agent capabilities should be described at an abstract rather than implementation level.

*This research has been sponsored in part by Office of Naval Research grant N-00014-96-16-1-1222, and by DARPA grant F-30602-98-2-0138. We want to acknowledge the contributions of Matthias Klusch and Jianguo Lu.

[1] We assume the existence of multiple middle agents on the Internet. We have developed protocols for efficient, distributed matchmaking among multiple middle agents (Jha, et al. 1998).

- **Inferences**. Inferences on descriptions written in this language should be supported. Automated reasoning and comparison on the descriptions should be possible and efficient.
- **Ease of Use**. Descriptions should not only be easy to read and understand, but also easy to write by the user. The language should support the use of domain or common ontologies for specifying agents capabilities.

A specification in LARKS is a frame with the following slot structure.

`Context`	Context of specification
`Types`	Declaration of used variable types
`Input`	Declaration of input variables
`Output`	Declaration of output variables
`InConstraints`	Constraints on input variables
`OutConstraints`	Constraints on output variables
`ConcDescriptions`	Ontological descriptions of used words
`TextDescription`	Textual Description of specification

Local Domain Ontologies: As mentioned above LARKS offers the option to use application domain knowledge in any advertisement or request. This is done by using a local ontology for describing the meaning of a word in a LARKS specification. In our implementation of the matchmaking process it is assumed that any local ontology is defined in the concept language ITL (Sycara, Lu, and Klusch 1998).

Any user or agent, requester or provider, may browse through the matchmaker's ontology and use the included concepts for describing the meaning of words in a specification of a request or advertisement in LARKS[2].

3 Matchmaking Using LARKS

Every specification in LARKS can be interpreted as an advertisement as well as a request; the specification's role depends on the agent's purpose for sending it to a matchmaker agent, and it is indicated in the wrapper language by an appropriate performative (advertise or request). Every LARKS specification must be wrapped by the sending agent in an appropriate message that indicates if the message content is to be treated as a request or an advertisement.

The matching engine of the matchmaker agent contains five different filters (described below).

Context Filter: Any matching of two specifications has to be in an appropriate context. In LARKS to deal with restricting the advertisement matching space to those in the same domain as the request, each specification supplies a list of keywords meant to describe the semantic domain of the service. Word distance is computed using the trigger-pair model. If two words are significantly co-related, then they are considered trigger-pairs, and the value of the co-relation is domain specific. In the current implementation we use the Wall Street Journal corpus of one million word pairs to compute the word distance.

Profile Filter: Although context matching is most efficient, it does not consider the whole specification itself. This is done with a profile filter that compares two LARKS specifications by using a variant of the known TF-IDF (term frequency-inverse document frequency) technique (Salton and Wong 1975).

Similarity Filter: Computation of similarity relies on a combination of distance values as calculated for pairs of input and output declarations, and input and output constraints. Each of these distance values is computed in terms of the distance between concepts and words that occur in their respective specification section.

Signature and Constraint Filters: The similarity filter takes into consideration the semantics of individual words in the description. However, it does not take the meaning of the logical constraints in a LARKS specification into account. This is done in our matchmaking process by the signature and constraint filters. Signature matching checks if the signatures of input and output declarations match. It is performed by a set of subtype inference rules as well as concept subsumption testing (see (Sycara, Lu, and Klusch 1998) for details).

[2]This is similar to the common use of domain namespaces in XML for semantically tagging Web page contents.

References

K. Decker, K. Sycara, M. Williamson. Middle-Agents for the Internet. Proc. 15th IJCAI, pages 578-583, Nagoya, Japan, August 1997.

S. Jha, P. Chalasani, O. Shehory and K. Sycara. A Formal Treatment of Distributed Matchmaking. In Proceedings of the Second International conference on Autonomous Agents (Agents 98), Minneapolis, MN, May 1998.

G. Salton, A. Wong. A vector space model for automatic indexing. Communications of the ACM, 18, 613-620, 1975.

K. Sycara, J. Lu, and M. Klusch. Interoperability among Heterogeneous Software Agents on the Internet. Carnegie Mellon University, PA (USA), Technical Report CMU-RI-TR-98-22.

K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed Intelligent Agents. IEEE Expert, pp. 36 - 46, December 1996.

LiveMarks: Collaborative Information Gathering

LiveMarks builds on mutuality: letting everybody profit a great deal from the accumulated results of other people's work with minimal additional effort for each person involved. While a user submits queries to a search engine, browses and assesses the results, software agents in the background can apply their accumulated knowledge about the users and look for recommendations from other people and find related documents by inspecting large collections of documents using text mining techniques.

LiveMarks uses BSCW, a shared workspace system on the Web, as its front-end. BSCW supports cross-platform cooperative work in widely dispersed working groups by the provision of "shared workspaces", i.e. repositories in which users can upload arbitrary electronic documents, collect URLs, hold threaded discussions, and are kept aware of the activities of others to coordinate their own work. BSCW is integrated with an unmodified Web server and is accessible from standard Web browsers.

For agent-based information collection we have extended BSCW in two ways: at the user interface we have introduced a new type of object, the query, and a rating and annotation feature for URLs; at the back-end we have enabled BSCW to communicate with LiveMarks agents.

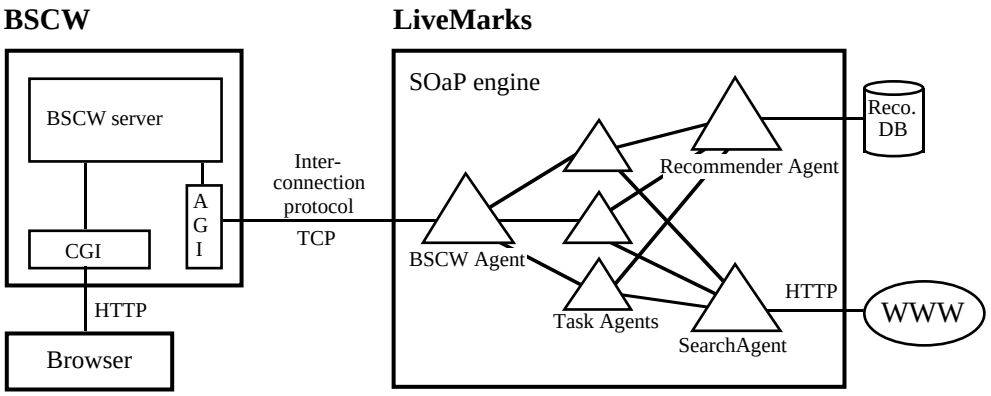
Agent platform

LiveMarks agents are implemented on our SOcial Agents Platform (SOaP). SOaP forms an extension of the Java Virtual Machine and constitutes a minimal operating system for multi-agent applications. It is tailored to our application requirements, namely openness, scalability, robustness and security.

An agent consists of at least one thread and communicates via asynchronous message object passing using mailboxes. Agents run concurrently in a single Java VM, called "agent engine", or may be distributed among several agent engines.

SOaP is conceptually divided into four abstraction layers. The two lower layers deal with the local agent life cycle management and additionally provide basic agent services, e.g. a name service. The remaining layers implement location transparent general and application specific service agents.

The LiveMarks application employs four types of agents: the BSCW agent interfaces with the BCSW server and spawns task agents which are associated to BSCW workspaces and process the queries within these workspaces. Search and recommender agents are service agents which both wrap external information sources: search engines like AltaVista or Infoseek and the recommender database of rated Web documents.



User perspective

Whenever a user creates a query for Web documents, this query is propagated to the software agents that work in the background. The agents forward the query to search engines, collect the results, and enrich them with their own recommendations. The recommendations are derived from an internal database that stores references and descriptions of Web documents along with user ratings and annotations. The agents produce recommendations by searching this database for highly rated documents whose descriptions match the query. The best-ranked results are transmitted back to BSCW where they are presented within the query to which they belong; the query operates as a folder for its results.

After having received the results, users may inspect the documents as well as rate and annotate them for the benefit of their fellow users with whom they share the workspace. Rated and annotated URLs

are automatically moved out of the query folder one level up to a more prominent position, URLs judged irrelevant disappear for good. The ratings and annotations are also propagated to the LiveMarks agents which store them in their document database for future recommendations.

The BSCW interface of LiveMarks showing two active query objects, rated and annotated URLs, and a document produced by the group.

As long as a query is active, new results will continue to flow in when the query folder can take more. The capacity of a query folder is limited in order to ensure a better overview for the users and to avoid flooding them with too many results. The agents make sure that, at any time, the folder will contain only the best results. The flow of results can be stopped by inactivating a query.

For its members, a BSCW shared workspace serves as the context for an information collection task. The workspace contains all queries and all relevant results. Within this context, the agents will minimize redundant information: same or similar URLs are suppressed, material that has been judged irrelevant within a workspace, or has been removed from the workspace will not be produced again as response to a new query.

By integrating the agent-based information retrieval services of LiveMarks into the BSCW groupware system we believe to have created an environment that addresses the needs of information acquisition tasks:

- Information seeking extends over time; intermediate queries and results need to be preserved so that the activity may be interrupted and resumed easily.
- Information seeking is not a stand-alone activity. Support tools need to be integrated into an electronic working environment.
- Information seeking is not a solitary activity in most cases. Queries and search results need to be shared, assessed and structured in a working group.

Additionally, the group setting of LiveMarks motivates serious and responsible rating and annotating which in turn improves the quality of LiveMarks recommendations.

Contact:. A. Voss, V. Paulsen, GMD; angi.voss@gmd.de; http://orgwis.gmd.de/projects/Coins/

MailCat: An Intelligent Assistant for Organizing E-Mail

Software Demo

Richard B. Segal and Jeffrey O. Kephart
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
rsegal@watson.ibm.com, kephart@watson.ibm.com

Abstract

MailCat is an intelligent assistant that helps users organize their e-mail into folders [2]. It uses a text classifier to learn each user's mail-filing habits. MailCat uses what it learns to predict the three folders in which the user is most likely to place each incoming message. It then provides shortcut buttons to file each message into one of these three folders. When one of MailCat's predictions is correct, the effort required to file a message is reduced to a single button click.

1 Introduction

Most mail readers allow users to organize their messages into folders to ease later retrieval. One might suppose that the effort required to file a message using these programs would be negligible. In practice, however, many users find the cognitive burden of deciding where to file a message plus the time spent interacting with the user interface to be a substantial barrier. This barrier is significant enough that many users quickly fall behind and let unfiled messages pile up in their mailboxes.

MailCat is an intelligent assistant that helps users organize their e-mail into folders. MailCat uses a text classifier to learn each user's mail-filing habits. MailCat uses what it learns to predict the three folders in which the user is most likely to place each incoming message. It then provides shortcut buttons to file each message into one of these three folders. When one of MailCat's predictions is correct, the effort required to file a message is reduced to a single button click.

MailCat provides its assistance without placing any additional burdens on the user. When MailCat is first installed, it analyzes the user's existing folders to learn her mail-filing habits. MailCat immediately starts providing shortcut buttons. If the user likes MailCat's suggestions, she can use the shortcut buttons to quickly file her e-mail. If the user does not like MailCat's suggestions, she can file messages in the usual way. MailCat continuously improves itself and adapts to changes by learning from each new message the user files.

2 MailCat

Figure 1 shows how MailCat simplifies the task of organizing messages. MailCat places three buttons above each message that allow the user to quickly file each message into one of the three folders it suggests. When one of the three buttons is clicked, the message is immediately moved to the indicated folder.

MailCat uses a text classifier to predict the likely destination folders for each message. MailCat builds its text classifier by learning from user actions. Maes [1] suggests that it can take some time for a user to file enough messages for an e-mail assistant to learn a good classifier. Maes proposes collaborative learning as a solution to this problem in which each e-mail agent learns from other e-mail agents whose users have similar mail-filing habits. While this works well if one can find a user with similar mail-filing habits, finding such a user seems unlikely in practice given the diversity of mail-filing schemes.

An alternative solution is to learn from messages previously filed by the user. Most e-mail users already have a large database of previously-filed messages which can be used to bootstrap the text classifier — the messages currently stored in their folders. This database provides ample training data to get the classifier quickly up to speed. When MailCat is first installed, it reads the user's database of previously-filed messages and uses what it finds to train the text classifier. After this initial training, MailCat can immediately begin making useful predictions.

The initial training of the classifier is only half the battle. Users are constantly creating, deleting and

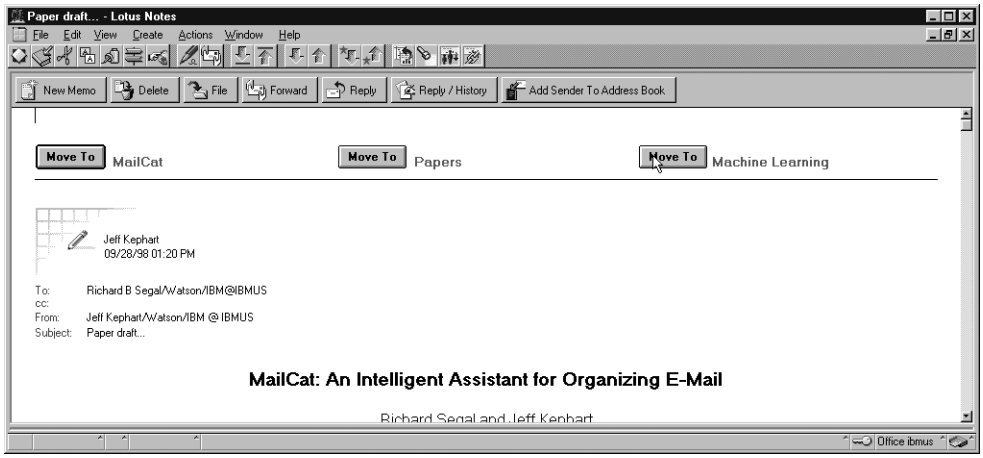


Figure 1: MailCat creates shortcut buttons for the three folders in which it predicts the user is most likely to place each message. When its predictions are correct, the user can file each message using a single button click.

reorganizing their folders. Even if the folders remain the same, the type of messages placed in a folder changes over time. MailCat solves this problem by using incremental learning. Once the classifier has been trained, MailCat's incremental-learning daemon watches for messages that are added to or deleted from each folder. Whenever a message is added or deleted, MailCat uses its incremental learning algorithm to update its user model. After updating, the classifier will generate the same predictions as if it were trained on the entire database.

Incremental learning allows MailCat to quickly respond to changes. If the user creates a new folder and adds a few messages, MailCat instantly learns about the folder and can start predicting which messages should go in the new folder.

MailCat provides three shortcut buttons rather than one to increase the chances that one of the buttons it provides is useful. MailCat can use three buttons because it was designed to provide advice rather than automatically file messages. Since a message can be automatically filed in only one folder, automatic categorization systems have to rely on the accuracy of their first prediction.

3 Experiments

We evaluated MailCat by simulating its performance on the mailboxes of six real users. The table below shows the accuracy of MailCat for each user when providing from one to three shortcut buttons. The accuracy of MailCat with N buttons is the frequency that one of the first N buttons it provides will move the message into the correct folder.

Buttons	1	2	3
R. Segal	77.8	88.5	91.8
J. Kephart	59.8	72.8	80.2
User #3	64.9	78.1	84.8
User #4	65.9	75.7	81.1
User #5	70.1	88.1	93.6
User #6	81.1	94.4	98.1

The experiment shows that MailCat is fairly accurate with just one button, getting between 59.8% and 81.1% accuracy. MailCat improves its overall performance by simply providing more than one shortcut button. By using three buttons, MailCat improves its accuracy to 80.2% to 98.1% — a factor of two reduction in its error rate.

4 Conclusions

MailCat simplifies the task of filing messages by analyzing the user's mail-filing habits to predict the three most-likely folders for each message and then providing shortcut buttons to quickly file each message into one of its predicted folders. Since its predictions are accurate over 80% to 90% of the time, MailCat substantially reduces the effort required to file messages.

References

[1] P. Maes. Agents that reduce work and information overload. *CACM*, 37(7):31–40, July 1994.

[2] R. Segal and J. Kephart. MailCat: An intelligent assistant for organizing e-mail. In *Proceedings of Agents'99*, May 1999.

Market Maker

David Wang, Giorgos Zacharia
dwang,lysi@media.mit.edu

Market Maker is an electronic marketplace project at the MIT Media Laboratory. Developed based on the Kasbah concepts, Market Maker utilizes software agents to assist users in making transactions online. It incorporates the concept of trust and collaborative reputation mechanisms to facilitate reliable online transactions. Transaction categories in the marketplace are fully extensible, providing support for a wide range of possible products and services to be traded online. Market Maker is currently used to facilitate a consumer-to-consumer marketplace at MIT. Categories of items being transacted are similar to those found in traditional marketplaces such as classified Ads and flea markets. Unlike the traditional buying and selling process, users of Market Maker do not need to monitor, identify, and negotiate with prospective buyers and sellers. Users can create software agents to conduct transactions on their behalf, while retaining control and various levels as appropriate.

Software agents in Market Maker have the ability to monitor, identify, and negotiate over products in all categories being transacted. Prior to handling a transaction, an agent receives knowledge from the marketplace on the category in it resides. Upon processing, the agent self-customizes and incorporate market rules specific to the particular category, including information on criteria it will use to find appropriate buyers or sellers, given a set of instructions for its user. Agents may also choose to receive market-wide information, such as prior transaction trends, supply and demand, in order to obtain better position during negotiations. Users may also customize agent's negotiation process and logic. Agents can receive instructions on item utilities and valuations from user's prospective, as well as information on urgency of transaction and styles of negotiations. Currently users can select from a pre-defined set of agent behaviors and customize them accordingly. User defined agent logic may be incorporated into Market Maker in the future, such that more sophisticated transaction algorithms can be implemented by users.

Unlike Kasbah, Market Maker has a much more modular architecture which allows the maintainer of the system to create new categories through a user through either the MS SQL server GUI or through a web-based interface. Using the Market Maker interface the maintainer can create hierarchical ontologies of goods or services, define the attributes of each category and the matchmaking behavior of the agents on the attribute level. The changes are reflected on the marketplace in real time, since no recompilation is necessary.

Software agents in Market Maker will pro-actively evaluate and negotiate with interested buyers and sellers, represented by their respective agents. Software agents can be created with any set of desired behaviors, thereby enabling the consumer to have a virtual presence in the marketplace to further his or her interest, while freeing the consumer from constant monitoring of market progress. This kinds of marketplace introduces two major issues of trust among the users of the system:

1. The potential buyer has no physical access to the product of interest while he/she bids or negotiates. Therefore the seller could misrepresent the condition or the quality of his/her product in order to get more money.
2. The seller or buyer may decide not to abide by the agreement reached at the electronic marketplace asking at some later time to renegotiate the price, or even refusing to commit the transaction. In order to solve the above mentioned problems, we incorporate in the system a reputation brokering mechanism, so that each user can actually customize his/her pricing strategies according to the risk implied by the reputation values of the potential counterparts.

In this demonstration we show two reputation mechanisms:

1. Sporas is a simple reputation mechanism which can be implemented irrespectively of the number of rated interactions, and

2. Histos is a more complex reputation mechanism that assumes that the system has been somehow bootstrapped (by using Sporas) so that there is an abundance of rated interactions to create a dense web of pairwise ratings.

Sporas provides a reputation service based on the following principles:

1. New users start with a minimum reputation value, and they build up reputation throughout their activity on the system.
2. The reputation value of a user should not fall below the reputation of a new user no matter how unreliable the user is.
3. After each rating the reputation value of the user is updated based on the feedback provided by the other party to reflect his/her trustworthiness in the latest transaction.
4. Two users may rate each other only once. If two users happen to interact more than once, the system keeps the most recently submitted rating.
5. Users with very high reputation values experience much smaller rating changes after each update.

For the calculation of the personalized Histos reputation values, we represent the pairwise ratings in the system as a directed graph, where nodes represent users and weighted edges represent the most recent reputation rating given by one user to another, with direction pointing towards the rated user. If there exists at least one connected path between two users, say from A to B, we can compute a more personalized reputation value for B. We do that by finding all the connected paths of ratings from user A towards user B. If user A has rated user B directly we use that rating alone to calculate the subjective opinion of user A for user B. Otherwise we proceed to the next level in a Breadth First Search manner, and we recalculate the subjective opinions of user A for each one of the users who are one edge further away from user A. The calculation is repeated level by level, until we finally reach user B. We will demonstrate a visualization of the reputational relations in the marketplace.

MASMaS: a Multi-Agent Simulation Management System[1]

Hamilton Link (helink@sandia.gov)
Advanced Information Systems Lab
B 836 / MS 0455
Sandia National Laboratories
Albuquerque, NM 87185

We are demonstrating a multi-agent simulation management system (MASMaS), developed recently at Sandia National Labs by the Advanced Information Systems Lab (AISL). The agents form a collection of independent autonomous nodes in a network that are all motivated to interact with people and collaborate with one another to complete tasks. The system was developed using the Standard Agent Architecture (SAA), a framework for rapidly developing collaborative networked multi-agent systems. The SAA also provides foundation technology for the Border Trade Facilitation System (BTFS), submitted for demonstration at this conference. In MASMaS, the agents interact with people to specify simulation tasks, collaborate to perform team formation and divide up the task, run the individual simulations, and collect and display the results when the user returns. Currently the simulator being run by the agents is being used to support collective robotics research.

The working system demonstrates the agents interacting with humans to describe the simulation batch run desired. Once any agent has been given such a description, the agent contacts the other agents in the collective, who collaboratively form a team and establish a joint persistent goal (JPG) to run a number of simulations and gather statistical data for the user. Work allocation is decentralized, so the agents agree among themselves how the work should be most appropriately allocated, rather than having a central authority figure make the decisions, because the agent collective is a homogeneous community with no identifiable leader. The simulation run by the agents at this time provides a virtual three-dimensional environment for situated actors that can sense, move, and communicate. It is being used to develop decentralized control algorithms that can be put into a collection of robots that then work together to achieve higher-level goals. The discrete-time simulation kernel can simulate these generalized actors with a variety of constraints on movement and communication. For example, the actors can be represented as point masses or physical objects that can interfere with one another.

In addition to the particular application of the SAA to the simulation management task, the demonstration showcases a number of other AISL core technology frameworks, including dynamic object-based web page generation, object brokering, a persistent objectbase, and the goal-based deliberative mechanism used by the agents. The SAA provides a default goal-based reasoning mechanism by which its agents accept, reason about, and act on goals presented to them. The framework is extended by adding new goals and goal satisfaction methods. The default methods are primarily placeholders for extensions that give the agents utility in real problem domains. We created the simulation agents by adding elicitation and analysis goals and methods to the agents' repertoire. The agents collaborate with one another to share the task of running a large number of simulations to explore regions of simulation parameter space. The agents also assist humans in viewing and analyzing the results. Using KQML as a

[1] This work was performed at Sandia National Laboratories, which is supported by the U.S. Department of Energy under contract DE-AC04-94AL85000

communications protocol and specialized goal classes that support the joint persistent goal (JPG) model of collaboration, a group of agents are able to share the task of running a large number of simulations with a variety of parameters to help collect data and analyze the results.

The agents interact with a person through a web browser, eliciting information for generating random variates, determining the type of simulation to be run and any other data needed to populate the simulation. The web interface system enables the agent to guide the user through this process, constraining or requesting changes to the input before moving on. These features are provided by a standard communication framework, called CHI (CLOS to HTML Interface). CHI enables the rapid construction of web-based interfaces that permit agents to initiate and conduct dynamic communication with human informants. To assist in this process we have developed a software mechanism called HCHI (HTML to CHI) that converts HTML into the appropriate nested CHI instances. The essence of CHI is automatic connection of form input elements to named objects in the internal object-oriented environment. The connectivity preserves state information so that an agent can conduct a session-based dialogue with a human, preserving temporal and state information as necessary. The dynamic capabilities of CLOS also permit classes to be defined at runtime, offering the possibility of interfaces designed dynamically based on user input and discovered information.

We are using other AISL technologies as well, including DCLOS (Distributed CLOS), to allow multiple agents to run transparently on several processes being executed by several CPU's connected by a network, and SpireStore, our persistent objectbase, to store simulation results. A key element of the simulation process is the “simulation seed,” an object containing not only the specification of the parameter settings for the simulation run, but also the initialization seeds for the necessary random variates so that the simulation run is entirely deterministic given its seed. In practice, batch runs are created by building a batch object specifying a number of these simulation seeds. The task is shared (once allocation has been decided upon) by creating several different batch objects, the completion of all of which accomplish the batch goal. The batch task is approximately linearizeable and fine-grained, which means that the task can be divided easily (one simulation run is very like another). Sharing non-linearizeable tasks is a research area, particularly when the tasks have interlocking dependencies. The seeds are then divided among the cooperating agents in a way that allows approximately equal clock time on the task by each agent involved in the collaboration based on past behavior on similar tasks. This is case-based reasoning with Bayesian assumptions. Other areas of research are anytime processing, which involves interrupting the collective more-or-less regardless of what it is doing to inquire about the results of an ongoing distributed task, dynamic progress monitoring, and dynamic goal redistribution. The ongoing AISL research agenda is also exploring individual agent integrity, agent collective integrity, and propagation of learned behavior.

MASMaS represents ongoing research in team formation, distributed, decentralized load-balancing, and collective behavior in response to failed commitments in the collective. Currently the simulation interface is being developed to allow more detailed specification of the desired simulation to be run and to allow a greater amount of control on the information presented when the user returns – although the system already supports such interfaces, the user interface has not been the primary direction of work. A case-based reasoning system could also be used to populate the specification form in the first place as a labor-saving device. The simulator itself is being developed to allow finer control of the resources it uses, including the addition of self-monitoring capabilities to the simulation and dynamic data structures which adapt to time and memory restrictions given the nature of data collections in the simulation.

A Personal News Agent that Talks, Learns and Explains

Daniel Billsus and Michael J. Pazzani
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{dbillsus, pazzani}@ics.uci.edu

Towards Portable, Intelligent Information Devices

Most work on intelligent information agents has thus far focused on systems that are accessible through the World Wide Web. As demanding schedules prohibit people from continuous access to their computers, there is a clear demand for information systems that do not require workstation access or graphical user interfaces. We present *News Dude*, a personal news agent that is designed to become part of an intelligent, IP-enabled radio. For example, an intelligent car radio that learns about the driver's interests is a useful application of this technology. Our system uses synthesized speech to read news stories, and allows users to provide feedback via voice commands. Based on this feedback, the system uses machine learning algorithms to automatically adapt to the user's preferences and interests.

System Overview

We have implemented a Java Applet that uses Microsoft's *Agent* library to display an animated character that reads news stories to the user. Although our ultimate goal is to work towards a speech-driven agent that does not require graphical user interfaces, we use the web as a medium that allows us to make the system available to a large user base for data collection and testing purposes. Furthermore, we believe that there are a variety of useful applications for speech-driven agent technology for the web. For example, a talking news agent that reacts to voice commands could prove useful for the visually impaired.

Figure 1 shows the News Dude user interface. Currently, the system provides access to stories from six different news channels: Top Stories, Politics, World, Business, Technology and Sports. When the user selects a news channel, the Applet connects to a news site on the Internet and starts to download stories. Since the Applet is multi-threaded, stories continue to download in the background while the synthesizer is reading, which typically allows filling a queue of stories to be read without any waiting time. The user can interrupt the synthesizer at any point and provide feedback for the story being read. One of the design goals for our system was to provide a variety of feedback options that go beyond the commonly used *interesting/uninteresting* rating options. For example, we might want to tell the agent that we already know about a certain topic, or request information related to a certain story. In addition, we would like to be able to ask the agent for reasons why a certain story was rated as interesting or uninteresting, just as we would ask a friend about reasons for a particular recommendation. In summary, the system supports the following feedback options: *interesting, not interesting, I already know this, tell me more,* and *explain.*

Challenges

Building an agent that learns about a user's interests in daily news stories poses several challenges. Traditional Information Retrieval approaches are not directly applicable to this problem setting, because they assume the user has a specific, well-defined information need. In our setting, however, this is not the case. If at all, the user's query could be phrased as: "What is new in the world that I do not yet know about, but should know?" Computing satisfactory results for such a query is non-trivial. The difficulty stems from the range of topics that could interest the user, and the user's changing interest in these topics. We must also take into account that it is the novelty of a story that makes it interesting. Even though a certain topic might match a user's interests perfectly, the user will not be interested in the story if it has been heard before. Therefore, we need to build a system that acquires a model of a user's multiple interests, is flexible enough to

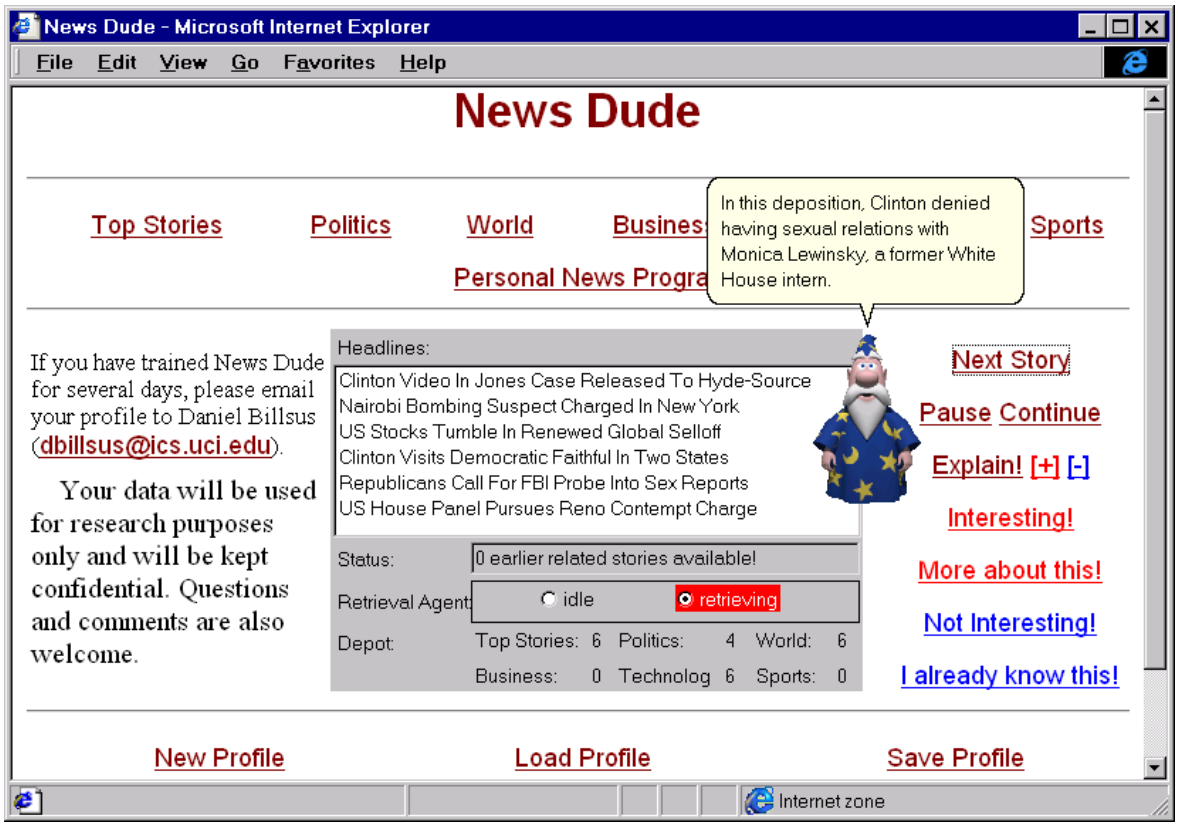


Figure 1: News Dude User Interface

account for rapid interest changes, and keeps track of information the user knows.

Technical Contributions

The system uses a combination of machine learning techniques to induce a user's interest profile. This can be seen as a text classification task, where a learning algorithm uses a set of rated text documents, here news stories, to induce a classifier that can label future stories with respect to the user's interests. Taking the domain-specific challenges and requirements of our application into account, the system uses three novel techniques that extend text classification algorithms previously reported in the literature.

1. *Time-Coded Feedback* – The system converts a user's rating to a fine-grained scale, depending on the length of time the user listened to a story.
2. *Multi-Strategy User Modeling* – The user model consists of two separate models, one for the user's short-term interests, the other for long-term interests. The short-term model is based on a Nearest Neighbor classifier, allowing for identification of previously rated news threads with only a few rated stories. The long-term model is based on a Naïve Bayesian Classifier, using a general hand-selected vocabulary, expressing common reoccurring themes in daily news stories.
3. *Concept Feedback* – The agent can construct explanations for its predictions and users can critique these explanations. This form of feedback can be incorporated into the learning process, allowing for faster acquisition of more accurate user models.

These techniques are described in detail in [1].

System Availability

The system is publicly available at http://www.ics.uci.edu/~dbillsus/NewsDude Comments and feedback are welcome.

References

[1] Billsus, D. and Pazzani, M. (1999). "A Personal News Agent that Talks, Learns and Explains" In *Agents '99: Proceedings of the Third International Conference on Autonomous Agents*, ACM Press, May 1999.

Intelligent Interfaces for Decision-Theoretic Systems

Scott M. Brown
Crew Systems Interface Division
Air Force Research Laboratory
Wright-Patterson AFB, OH 45433
+1 937 255 8883
sbrown777@acm.org

Eugene Santos Jr.
Dept. of Computer Science & Engineering
University of Connecticut
Storrs, CT 06269-3155
eugene@eng2.uconn.edu
http://www.eng2.uconn.edu/cse/IDIS/

ABSTRACT
This demonstration presents PESKI, a probabilistic expert system shell. PESKI provides users with an integrated suite of knowledge elicitation tools for decision-theoretic systems, from "standard" knowledge acquisition tools, data mining tools, and verification and validation tools to a distributed inference engine for querying knowledge in the system. PESKI uses a number of techniques to reduce the inherent complexity of developing a cohesive, real-world knowledge-based system. In addition to providing multiple communication modes for human-computer interaction, the expert system knowledge representation is endowed with the ability to detect problems with the knowledge acquired and to alert the user to these possible problems. Furthermore, we show PESKI's use of an intelligent assistant to assist users with the acquisition of knowledge and the use of the myriad of tools.

INTRODUCTION
Most everyday decisions involve some level of uncertainty. Expert systems, also known as knowledge-based systems, attempt to capture an expert's knowledge for use by non-experts. Among the advantages to using expert systems are wide distribution, accessibility, and preservation of scarce expertise, ease of modification, consistency and explanation of the answers.

One of the greatest disadvantages to expert systems is their construction. To aid experts in the arduous task of designing expert systems, a number of expert systems shells exist today. Most of these shells allow the expert system designers to capture an expert's knowledge, verify and validate that knowledge, and query this knowledge, i.e., perform inference. The tools available within a shell vary between each shell. Some provide a graphical means of acquiring knowledge from users. Most incorporate some form of verification and validation of the knowledge. However, none of these systems provide an integrated suite of tools for acquiring knowledge, testing that knowledge via verification and validation, and inference. Furthermore, these systems typically require complete information before they are of any use.

BAYESIAN KNOWLEDGE BASES
A Bayesian knowledge base (BKB) is a probabilistic knowledge representation meeting the preceding qualities. A BKB supports theoretically sound and consistent probabilistic inference — even with incomplete knowledge — with the intuitiveness of "if-then" rule specification. The representation is similar to Bayesian Networks; it is a directed graph capable of representing uncertainty in knowledge via probabilistic relationships between random variables (called components in PESKI). However, Bayesian networks do not allow for incompleteness.

Inherent in the BKB knowledge representation are several consistency constraints endowing the resulting knowledge base the ability to detect problems with the knowledge acquired and alert the user to these possible problems. As a result of these consistency constraints, all knowledge elicited is validated against these constraints. Any inconsistencies with the elicited knowledge results in a status message to the user. Certain consistency constraint violations can be corrected without user intervention, with an appropriate status message displayed to the user. For others violations, user intervention is required. Users may correct the violation using one of the PESKI tools (e.g., knowledge acquisition, data mining, verification and validation) discussed next.

THE PESKI ENVIRONMENT
PESKI (Probabilities, Expert Systems, Knowledge, and Inference) is an integrated probabilistic knowledge-based expert system shell utilizing Bayesian knowledge bases as its knowledge representation. PESKI provides users with knowledge acquisition, verification and validation, data mining, and inference engine tools, each capable of operating in various communication modes.

The architecture consists of four major components:

- **Intelligent Interface Agent** - translates English questions into inference queries and translates the analyses/inference results back into English; provides for the communication exchange between the user and the system; provides intelligent assistance to the user.
- **Inference Engine** - contains the intelligent control strategies for controlling the selection and application of various inference engine algorithms (e.g. A*, 0-1 integer linear programming (ILP), genetic algorithms (GAs)) to obtain conclusions to user queries based on knowledge and facts in our knowledge base.

- **Explanation & Interpretation** - keeps track of the reasoning paths the inference engine used in reaching its conclusions; allows the user to query the system about how and why an answer was derived.
- **Knowledge Acquisition & Maintenance** - provides the facility for automatically incorporating new or updated expert knowledge into the knowledge base.

PESKI's Integrated Tool Suite

We briefly describe the tools integrated into the PESKI architecture.

- *Knowledge Acquisition* - PESKI uses the MACK tool for knowledge acquisition. MACK contains routines designed to automatically and incrementally confirm consistency of the knowledge elicited from the expert and provides assistance via knowledge base status messages. Regular incremental checks preserve both probabilistic validity and logical consistency by flagging the inconsistent data points to the expert as they are entered and presumably under his/her current consideration.
- *Verification & Validation* - PESKI verification and validation is performed using two tools - BVAL and a graphical incompleteness tool. BVAL validates a knowledge base against its requirements using a test case-based approach. A test case is a set of evidence and expected answers. A knowledge engineer submits a test suite to the BVAL tool and BVAL determines if the given evidence is supported by the answers by submitting a query to the inference engine and comparing the solution with the test case's expected answer. Under certain conditions, the knowledge base can be corrected via reinforcement learning of the probabilities. For those test cases that indicate incompleteness exists not meeting the conditions (typically a result of a missing causal relationship between two random variables), the graphical incompleteness tool may be used to visualize the knowledge base incompleteness and correct it. Figure 3 shows an example of the use of this tool in PESKI. The tool uses data visualization of the BKB and data mining to assist the user in eliciting the needed knowledge.
- *Inference Engine* - The PESKI inference engine uses a performance metric-based approach to intelligently control a number of possible *anytime* and *anywhere* inferencing algorithms (e.g., A*, genetic algorithms). Results are returned to the user via the Explanation & Interpretation subsystem of PESKI, as they become available.
- *Data Mining* - PESKI uses a *goal-directed* methodology for data mining for association rules and incorporation of these rules into the knowledge base. Data mining within PESKI can either be a knowledge acquisition or verification and validation process. In the latter case, an expert attempts to correct problems discovered as a result of performing verification and validation. In the former, using empirical and/or legacy data, an expert is able to mine for specific rules relating two or more database attributes (i.e., random variables in the BKB). Additionally, the data-mining tool can be used to find new states of a random variable and to elicit the probabilities of a single state.

Each tool in PESKI displays the current status of the BKB, alerting the user to any problems with the knowledge base. PESKI supports incremental knowledge elicitation in a number of ways. During knowledge acquisition, the user is alerted to any inconsistencies in the BKB knowledge representation. For example, if the user attempts to add a rule that creates a cycle in the knowledge base, PESKI will display an error message to the user.

Intelligent Assistance

Determining which tools to use given a particular situation in PESKI is difficult for most users. The use of a particular tool is dependent on a number of variables including the context (e.g., a BKB constraint violation exists) and user preferences for the tools and various communication modes. Determining the correct tool to use at the correct time can be a daunting task.

To aid users in efficiently utilizing the power of the PESKI tool suite offered, we have integrated an intelligent assistant into PESKI. The assistant takes the form of an interface agent, "looking over the shoulder" of the user. The overall goal of the assistant is to offer timely, beneficial assistance to the user as he/she interacts with PESKI. To accomplish this goal, an accurate cognitive model of the user is maintained. The user model captures the goals and needs of the user within the PESKI environment, as well as possible system events that occur, within a Bayesian network representation of the PESKI environment. Additionally, a user profile is maintained on each user of PESKI so assistance may be custom-tailored to individual users. The interface agent determines the how, when, what, and why of offering assistance to the user by inferencing over the user model. The agent is capable of offering assistance for such goals as which tool to use to correct a BKB consistency constraint violation as well as suggesting the user preferred communication mode for a given tool.

We are currently modifying the agent's architecture to allow the agent to collaboratively elicit information from the user based on what goals he/she is trying to achieve, his/her preferences, and past actions. To that end, we are adding "deep" domain knowledge of BKBs to the interface agent's user model.

Remote Agent Demonstration

Gregory A. Dorais
Caelum Research,
NASA Ames Research Center
MS 269-2, Moffett Field, CA 94035
650-604-4851
gadorais@ptolemy.arc.nasa.gov

James Kurien
NASA Ames Research Center
MS 269-2, Moffett Field, CA 94035
650-604-4745
kurien@ptolemy.arc.nasa.gov

Kanna Rajan
Caelum Research,
NASA Ames Research Center
MS 269-2, Moffett Field, CA 94035
650-604-0573
kanna@ptolemy.arc.nasa.gov

ABSTRACT

We describe the computer demonstration of the Remote Agent Experiment (RAX). The Remote Agent is a high-level, model-based, autonomous control agent being validated on the NASA Deep Space 1 spacecraft.

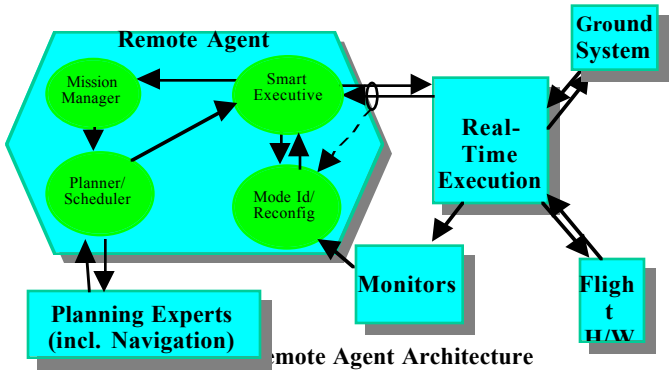
Keywords

Model-based autonomous agents, model-based inference, executives, planners, spacecraft.

1. INTRODUCTION

The Remote Agent (RA) is autonomous control software that uses models to reason about the system that it controls and the environment it is in. It does so to accomplish goals over extended periods including diagnosing and recovering from failures without contact with human operators. RA is being validated on the NASA Deep Space 1 spacecraft (DS1) during the Remote Agent Experiment (RAX) scheduled for mid-May, 1999. During RAX, RA will control DS1 and perform several activities including taking pictures, thrusting the ion propulsion engine, and diagnosing and recovering from simulated failures. RA, its major components, and RAX have been described in several papers [1][5][6][7][8][9]. This paper describes a computer demonstration that was designed to aid people unfamiliar with spacecraft and autonomous agent technologies to better understand RA and RAX.

2. REMOTE AGENT ARCHITECTURE



emote Agent Architecture

As illustrated in figure 1, RA consists of four components: the Planner/Scheduler (PS), the Mission Manager (MM), the Smart Executive (Exec), and the Mode Identification and Reconfiguration module (MIR).

2.1 Planner/Scheduler and Mission Manager

The Planner/Scheduler (PS) generates the plans that RA uses to control the spacecraft [5]. Given the initial spacecraft state and goals, PS generates a set of synchronized high-level activities that, once executed, will achieve the goals. Mission goals are maintained by MM [1].

PS consists of a heuristic chronological-backtracking search operating over a constraint-based temporal database [5]. PS begins with an incomplete plan and expands it into a complete plan by posting additional constraints in the database. These constraints originate from the goals and from constraint templates stored in a model of the domain. PS consults domain-specific planning experts to access information that is not in its model. The temporal database and the facilities for defining and accessing model information during search are provided by the HSTS system [4].

2.2 Smart Executive

Exec is a reactive, goal-achieving, control system that is responsible for:

- Requesting and executing plans from the planner
- Requesting and executing failure recoveries from MIR
- Executing goals and commands from human operators
- Managing system resources
- Configuring system devices
- Reach and maintain an appropriate safe-mode as necessary
- System-level fault protection

Exec is goal-oriented rather than command-oriented. We define a goal as a state of the system being controlled that must be maintained for a specified length of time. For example, consider the goal: keep device A on from time x to time y. If Exec were to detect that device A is off during that period, it would perform all the commands necessary to turn it back on. This ability is particularly useful in hostile environments where exogenous events can cause devices to behave unpredictably.

Exec controls multiple processes in order to coordinate the simultaneous execution of multiple goals that are often inter-dependent. In order to execute each goal, Exec uses a model-based approach to create a command procedure, which is often complex, designed to robustly achieve the goal.

2.3 Mode Identification/Reconfiguration

The Livingstone inference engine provides the mode identification (MI) and mode reconfiguration (MR) functionality in MIR. To track the modes of system devices, Livingstone eavesdrops on commands that are sent to the

spacecraft hardware by the Exec. As each command is executed, Livingstone receives observations from spacecraft's sensors, abstracted by monitors in the spacecraft's control software. Livingstone combines these commands and observations with declarative models of the spacecraft components to determine the current state of the system and report it to the Exec. If any such failures occur, Livingstone will be used to find a repair or workaround that allows the plan to continue execution.

Livingstone uses algorithms adapted from model-based diagnosis [2] to provide the above functions. The key idea underlying model-based diagnosis is that a combination of component modes is a possible description of the current state of the spacecraft only if the set of models associated with these modes is consistent with the observed sensor values. This method does not require that all aspects of the spacecraft state are directly observable, providing an elegant solution to the problem of limited observability.

3. REMOTE AGENT EXPERIMENT

RAX was designed to demonstrate the capabilities of RA on DS1. During RAX, RA will plan how to thrust DS1's ion engine, when to take pictures of asteroids, and when to communicate with Earth. False data will be injected at certain times, unknown to RA, that simulate spacecraft failures. RA will diagnose the cause of these failures and often will be able to find an action that repairs the failure. Otherwise, RA will put the spacecraft into a safe state and find a new plan that accommodates the problem. In addition to operating on its own, RA will demonstrate cooperation with mission controllers by accepting new mission goals and advice on health of the spacecraft.

4. REMOTE AGENT VISUALIZATION

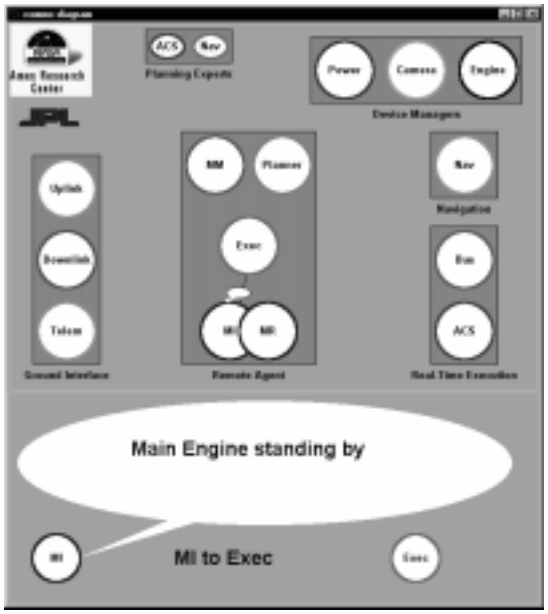


Figure 3. The Remote Agent Demonstation Window

To demonstrate RA, we use a window, in figure 3, that shows the messages as they pass between RA and the other spacecraft software and between RA components. This visualization of the RA can run in real-time while RA is running to show RA's current state, or from a log file of a prior RA run.

The top part of the window has a circle for each component of the RA and spacecraft flight software components RA communicates with. For example, RA sends messages to the attitude control system (ACS) to point the spacecraft toward Earth for communication or toward an asteroid for imaging. A small "speech balloon" travels back and forth between the software components showing which two are currently communicating. In the bottom portion of the window, the current message being transmitted is converted into a simplified English representation. Sensor observations from the spacecraft to RA are shown as moving yellow spheres. In figure 3, MIR is confirming to Exec that the main engine is ready. The demonstration shows a typical 6-day scenario including the ground uplink the command for RA to start its mission, PS interacting with the planning expert modules to create three plans, Exec executing the plans, and MIR sending diagnoses and recoveries to Exec.

5. ACKNOWLEDGMENTS

Our thanks to Bob Kanefsky for developing the visualization software for the RAX demo and to the RAX team cited in [1].

6. REFERENCES

[1] Bernard, D.E., Dorais, G.A., Fry, C., Gamble Jr., E.B., Kanfesky, B., Kurien, J., Millar, W., Muscettola, N., Nayak, P.P., Pell, B., Rajan, K., Rouquette, N., Smith, B., and Williams, B.C. Design of the Remote Agent experiment for spacecraft autonomy. Procs. of the IEEE Aerospace Conf., Snowmass, CO, 1998.

[2] de Kleer, J., and Williams, B. C. Diagnosis With Behavioral Modes. Procs. of IJCAI-89, 1989.

[3] Gat, E., and Pell, B. Abstract Resource Management in an Unconstrained Plan Execution System, Procs. of the IEEE Aerospace Conf., Snowmass, CO, 1998.

[4] Muscettola, N. HSTS: Integrating planning and scheduling, in Fox, M., and Zweben, M., (eds.), Intelligent Scheduling, Morgan Kaufman, 1995.

[5] Muscettola, N., Smith, B., Chien, S., Fry, C., Rabideau, G., Rajan, K., and Yan, D. On-board Planning for Autonomous Spacecraft, Procs. of i-SAIRAS, July 1997.

[6] Muscettola, N., Nayak, P.P., Pell, B.,Williams, B.C., Remote Agent: to boldy go where no AI system has gone before. Artificial Intelligence, 103(1/2), August, 1998.

[7] Pell, B., Gamble, E., Gat, E., Keesing, R., Kurien, J., Millar, W., Nayak, P.P., Plaunt, C., and Williams, B.C. A hybrid procedural/deductive executive for autonomous spacecraft. Procs. of Autonomous Agents, 1998.

[8] Pell, B., Gat, E., Keesing, R., Muscettola, N., and Smith, B. Robust periodic planning and execution for autonomous spacecraft. Procs. of IJCAI-97, 1997.

[9] Williams, B. C., and Nayak, P. A model-based approach to reactive self-Configuring systems, Procs. of AAAI-96, 1996.

Demonstration of Rational Communicative Behavior in Coordinated Defense

Sanguk Noh and Piotr J. Gmytrasiewicz
Department of Computer Science and Engineering
University of Texas at Arlington
{noh, piotr}@cse.uta.edu

1 Introduction

The primary goal of our demonstration is to show our communication results for artificial and human agents interacting in a simulated air defense domain. For artificial agents, we advocate a decision-theoretic message selection mechanism which maximizes the expected utility of the communicative decisions. Thus, the agents compute the expected utility of alternative communicative behaviors, and execute the one with the highest value [2]. Our demonstration consists of our RMM and human agents interacting in three different air defense scenarios in cases when communication is, and is not, available. We will show how communication can benefit the agents in coordination tasks, and compare performance of RMM and human agents.

2 Demonstration Settings

In our implementation, the anti-air defense simulator with communication was written in Common LISP and built on top of the MICE simulator [1]. Our demonstration is intended to compare the performance achieved by RMM team with that of human team, with and without communication, in three different scenarios.

In the anti-air defense domain, two defense units are faced with an attack by seven incoming missiles, as depicted in Figure 1. The warhead sizes of missiles are 470, 410, 350, 370, 420, 450, and 430 unit for missiles A through G, respectively. The positions of defense units are fixed and those of missiles are randomly generated. Each of two defense units is assumed to be equipped with three interceptors, if they are not incapacitated. Thus, they can launch one interceptor at a given state, and do it three times during a course of one defense episode.

Figure 1: A complex air defense scenario.

For all settings, each defense unit is initially assumed to have the following uncertainties (beliefs) in its knowledge base:

- The other battery is fully functional and has both long and short range interceptors with probability 60%;
- The other battery is operational and has only long range interceptors with probability 20% (In this case, it can shoot down only distant missiles, which are higher than a specific altitude.);

• The other battery has been incapacitated by enemy fire with probability 10%;
• The other battery is unknown with probability 10%.

In each demonstrated scenario we allow for one-way communication at a time between defense units. Thus, if both agents want to send messages, the speaker is randomly picked in the RMM team, and the human team flips a coin to determine who will be allowed to talk. The listener is silent and can only receive messages. Each of human subjects is presented with the scenarios, and is given a description of what is known and what is uncertain in each case. They are then asked to indicate which of the 11 messages is the most appropriate in each case. In all of the anti-air defense scenarios, each battery is assumed to have a choice of the following communicative behaviors:

- "No communication."
- "I'll intercept missile A."

...

- "I'll intercept missile G."
- "I have both long and short range interceptors."
- "I have only long range interceptors."
- "I'm incapacitated."

Given the uncertainties and the communicative behaviors, we set up three different scenarios. For each scenario, RMM and human agents intercept incoming targets with and without communication, respectively. We demonstrate their target selection sequences in all settings by retrieving them from http://dali.uta.edu.

To evaluate the quality of the agents' performance, we express the results in terms of (1) the number of selected targets, i.e., targets the defense units attempted to intercept, and (2) the total expected damage to friendly forces after all six interceptors were launched. The total expected damage is defined as a sum of the residual warhead sizes of the attacking missiles. Thus, if a missile was targeted for interception, then it contributed $\{(1 - Probability\_of\_Hit) \times warhead\_size\}$ to the total damage. If a missile was not targeted, it contributed all of its warhead size value to the damage.

3 Conclusion

Our demonstration presents the implementation and evaluation of the decision-theoretic message selection used by automated agents coordinating in an anti-air defense domain. We measure the increase in performance achieved by rational communicative behavior in the RMM team, and compare it to the performance of the human-controlled defense batteries. The results are intuitive: as expected, communication improves the coordinated performance achieved by the teams. An interesting aspect of the demonstration is that it shows the differences between the communicative behaviors exhibited by RMM and human agents. While human communicative behaviors are often similar to those selected by the RMM agents, there are telling differences that, in our experimental runs, allow the RMM team to achieve a slightly better performance. It may be that the differences in processing of probabilistic information about the uncertainties involved explain why decision making achieved by artificial agents tends to be somewhat superior to that of human agents.

References

[1] E. H. Durfee and T. A. Montgomery. MICE: A flexible testbed for intelligent coordination experiments. In *Proceedings of the 1989 Distributed AI Workshop*, pages 25–40, Sept. 1989.

[2] S. Noh and P. J. Gmytrasiewicz. Implementation and evaluation of rational communicative behavior in coordinated defense. To appear in *Proceedings of the Third International Conference on Autonomous Agents*, May 1999.

ROPE: Role Oriented Programming Environment for Multiagent Systems

Micheal Becht, Jürgen Klarmann, Matthias Muscholl
{becht, klarmann, muscholl}@informatik.uni-stuttgart.de

1 OVERVIEW OF THE ROPE PROJECT

ROPE is a programming environment and architecture for the development of agent based cooperative applications using a role based approach.

Figure 1 shows the components developed in the ROPE project. Since we aim to describe cooperation ROPE bases on a general model of cooperation.

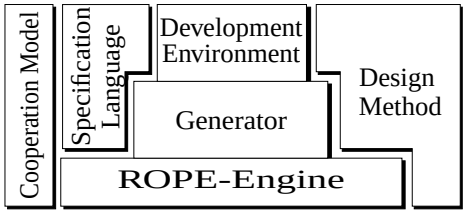


Figure 1. Components of ROPE.

To be able to describe cooperation processes graphically yet with well defined formal semantics we have developed a formal specification language extending high level petri nets ([Becht et al., 1998], [Klarmann et al., 1998]).

Our goal has been to develop a distributed MA environment and not a simulation running on a single machine. The ROPE engine provides all the necessary code for the distributed execution of a cooperation process, thereby imposing very few requirements on the participating agents.

The development environment consists of a visual editor to graphically design cooperation processes. Additional editors allow to specify the behavior of the elements of a cooperation net. The transitionEditor is used to specify the firing behaviour of transitions through edge annotations and guards. The stateEditor is used to specify which types of roles are available in a certain state. The roleEditor is used to specify the interface between the roles and the cooperation net and how the required behavior has to be provided by the agents. The agents have to supply an application dependent service interface to be able to take part in a cooperation process.

A generator which transforms cooperation processes described in the specification language to a “ready to run” cooperation process can be invoked and the running cooperation proceess can be monitored from within the environment. A detailed design methodology on how to develop specific cooperation processes is left for future work.

2 MODEL AND LANGUAGE

The aim of our cooperation model and the specification language is to support transformable corporate structures[1]. Transformability requires that it must be possible to change the organizational structure without the need to restructure the tasks and vice versa. It should be possible to insert new cooperation processes without changing the agents and without changing the running cooperation processes or stopping and restarting the whole system. It should also be possible to replace agents taking part in a cooperation process by other agents and to add new or change running cooperation processes.

To fulfil these requirements we introduce our key concepts in section 2.1. Further design decisions lead to the implementation of ROPE. They are summarized in section 2.2.

2.1 Key concepts

The characteristic feature of ROPE is its strong emphasis on the role concept. It is introduced for satisfying the agent’s requirements mentioned above.

Roles provide a well defined interface (Role-Agent Interface) between agents and cooperation processes, which enables an agent to read and follow the normative rules given by the cooperation process even if not known to the agent before.

In the specification of a cooperation process we introduce a role as an abstraction of an agent. The agent carries out the actions initiated by the role. The fundamental idea is to decouple the organization of the agents in the multiagent system from the structure of cooperation processes. By that, changes in the agent organization do not affect the cooperation process specification and vice versa[2]. This is the prerequisite for more transformable agent cooperation.

We understand cooperation as a process which is controlled by normative rules to which the cooperating partners commit themselves, when accepting a certain role. The rules are defined with respect to the cooperation, using a global view which is independent from certain agents.

[1] See http://www.sfb467.uni-stuttgart.de for more information on the joint research project SFB 467 „Transformable Corporate Structures in Multi-Variant Serial Production”

[2] In fact we see the organization structure of agents as long-term cooperation processes.

On a more concrete view we understand a cooperation process as a set of stages which model interactions. In a running cooperation process one or more stages are active. After the goal of a stage has been reached the control proceeds to one or more successive stages. The precondition of a stage is given by the goals of the preceding stages.

During the execution of a cooperation process an agent can change its role. This allows to have small roles designed for a particular purpose and are therefore easy to maintain.

2.2 Design decisions

Implementing the key concepts we had to make design decisions, which are explained in the following.

Because the functionality of an application domain is considered as stable we base our model on the existence of a service model which has to be developed specific to a certain application-domain. A service may be concrete, leaving no possibility for interpretation, or may be more abstract, requiring intelligence and autonomy for execution. Additionally services have primitives for asking the service provider to make a decision.

An agent then provides a set of services describing its capabilities and permissions. The service interface is independent of programming languages and allows to execute roles as clients remote from agents as servers.

Capabilities of agents are realized as services. Therefore a role specifies the services needed by an agent. The obligation is specified by an action related to a role. The action describes how the required services have to be used. A role entering a stage executes its action on the services provided by the agent.

In detail an instantiated role runs asynchronously to its agent. It is responsible for the coordination part of a task and thereby controls the agent accordingly. The responsibility of the agent is to supply the role with knowledge and decisions that enables the role to fulfil its part.

The last design decision concerns the specification language. Because we want to describe cooperation processes and their normative rules prescriptively, we use a high-level petri net class (predicate-transition nets) which is extended by the role concept.

Petri nets are known to be suitable for modelling discrete, event based, distributed systems. In our extension we model an agent playing a role as a token. Therefore we are able to describe the dynamic behaviour of interacting agents. Different types of tokens are used to represent different types of roles. Roles export certain local states of their action, so that guard expressions of transitions can be specified as boolean functions over these states. Guard expressions describe goals of interactions.

The advantage of this choice is the adequacy. Tokens allow a good visualization of the distributed state in a cooperation process and thereby simplify the design and test of agent cooperation. Communication between roles is either performed directly between roles in a stage or is done during the firing of a transition.

To summarize, a ROPE multiagent application consists of a multiagent layer implementing the functionality of the application and a cooperation layer on top specifying all cooperation networks and roles needed for cooperation.

Our design goal is to have an entirely distributed system avoiding a single point of failure.

3 ACKNOWLEDGEMENT

Peter Burger, Thorsten Gurzki, Jens Hoffmann, Ulrich Frank.

4 REFERENCES

[Becht et al., 1998] Becht, M.; Muscholl, M; Levi, P.: Transformable Multi-Agent Systems: A Specification Language for Cooperation Processes. In: Proceedings of the World Automation Congress, ISOMA'98, May 10-14, 1998, Anchorage, Alaska, USA, Jamshidi, M.; de Silva, C. W.; Pierrot, F.; Fathi, Bien, Z.; and Kamel, M., 1998.

[Klarmann et al., 1998] Klarmann, J.; Becht, M; Muscholl, M.: Modellierung flexibler Workflows mit teilausführbaren Aktivitäten (in german). In: Proceedings of the D-CSCW'98 Workshop "Flexibilität und Kooperation in Workflow-Management-Systemen", University of Münster, Angewandte Mathematik und Informatik, Technical Report No. 18/98-I, pp. 44-55

Autonomous Mars Rovers: Sequence Generation, Testing, and Execution

Project lead: John Bresina
Sequence generation: Corin Anderson, Ted Blackmon, John Bresina, Laurent Nguyen, David E. Smith
On-board architecture: John Bresina, Keith Golden, Katherine Smith, Trey Smith, Rich Washington
Simulation and visualization: Ted Blackmon, Vineet Gupta, Eric Zbinden

1 Introduction

The Pathfinder mission demonstrated the potential for robotic Mars exploration, but at the same time indicated the need for increased rover autonomy. The highly ground-intensive control with infrequent communication and high latency limited the effectiveness of the Sojourner rover. When failures occurred, Sojourner often sat idle for extended periods of time, awaiting further commands from earth. Significant advances in rover autonomy are needed to cope with increasing task complexity and greater execution uncertainty that will be inherent in future missions. In order to increase the flexibility and robustness of Mars rovers, we have developed a contingent sequence language, a contingent planner/scheduler to support generation of such sequences, and an onboard executive system that can execute contingent sequences, manage resources, and perform fault diagnosis. These work together with user interface and visualization tools for specifying goals and refining schedules. A realistic simulator can take the place of the rover for testing and visualizing sequences.

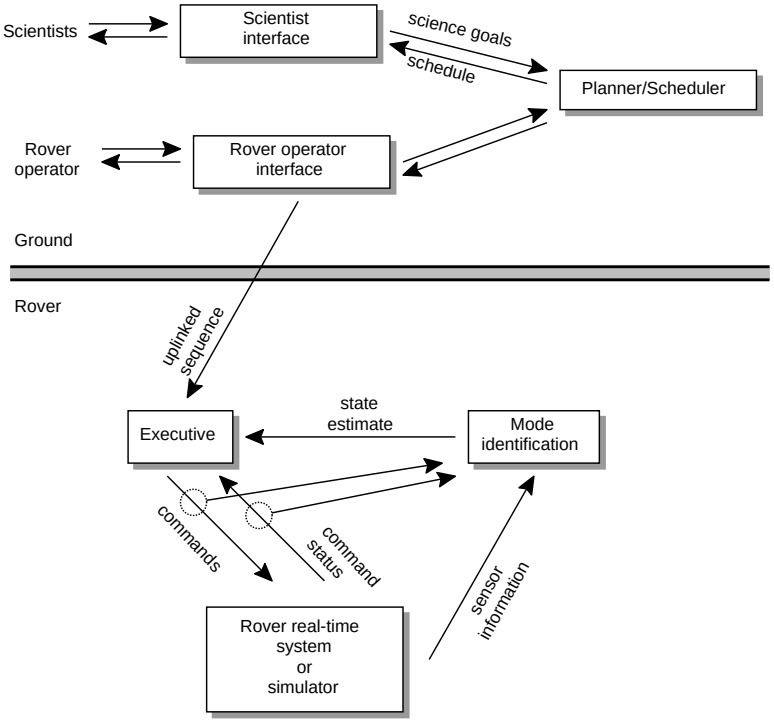


Figure 1: Rover architecture.

2 Rover Autonomy System

2.1 Sequence preparation

In our current rover system, the scientists specify high-level goals by interacting with a 3D VR interface, MarsMap. These goals are then given to the contingency planner/scheduler, CPS, which generates a temporally flexible schedule along with contingency plans to deal with possible execution failures and serendipitous science opportunities. The contingent schedule is refined through interaction with the scientist PI and rover operators.

To account for execution uncertainty, CPS can actively plan for, and take advantage of, possible contingencies. Thus, if an operation takes longer than a certain amount of time, or the power remaining drops below a specified value, a different pre-planned sequence of operations can be performed. Building contingency plans is, in general, intractable, so contingency planners tend to be slow. To overcome this problem, CPS employs the Just-in-Case (JIC) approach. The basic idea of JIC is to take an existing schedule and look for the places where it is most likely to fail. The JIC scheduler then generates alternative schedules for each of those situations.

2.2 Robust sequence execution

Once the ground personnel have produced a schedule, it is sent to the on-board conditional executive, CX. CX is responsible for interpreting the command sequence, monitoring plan execution, and potentially selecting alternative plan branches if the situation changes. A plan consists of a nominal sequence and a set of contingent branches. The nominal sequence is the sequence that will be executed if there are no deviations from the a priori expectations of the environment and actions. The contingent branches specify alternative courses of action. Within any contingent branch there may be further contingent branches, hence the primary plan is a tree of alternative courses of action. In addition, CX has a library of alternate plans, which are applicable at any time their conditions are satisfied. Enabling events may include unexpected opportunities, plan failures, or conditions such as resource shortfalls and component degradation.

The Mode Identification component (MI) eavesdrops on commands sent by CX to the rover. As each command is executed, MI receives observations from low-level monitors, which extract qualitative information from the rover sensors. For example, a current monitor may map the continuous-valued current into the set of qualitative values low, nominal, high. MI is informed whenever the qualitative value returned by a monitor changes. Based on monitor inputs, the commands executed on the rover, and a declarative model of the rover, MI infers the most likely current state. MI also provides a layer of abstraction to the executive, allowing plans to be specified in terms of component modes, rather than in terms of low-level sensor values.

2.3 Rover simulation for visualization and verification

For sequence testing and visualization, a rover simulator can replace the actual rover. The simulator receives and sends messages compatible with the rover real-time software, so the rest of the system can operate with no knowledge of whether the sequences are being executed on the real rover or in simulation. Currently the simulator is being used to visualize sequence execution, but it could be used as well to produce more accurate estimates of resource usage within the sequence-preparation process.

The core of the simulation is a hybrid discrete-continuous model of rover kinematics. This model operates in conjunction with the MarsMap VR software, which uses a 3-D terrain model generated from stereo images. The terrain model and the kinematics model combine to produce realistic sensor information, as well as realistic pose information for visualization. The simulated rover can be viewed using the MarsMap software.

3 Conclusion

The particular characteristics of Mars rover operation require a significant level of rover autonomy and an ability to handle resource constraints and unpredictable events. We have designed an architecture for rover autonomy that includes contingency planning on ground and flexible, robust execution of conditional sequences on board. The on-board executive draws on model-based fault diagnosis and dynamic resource management to maximize its science return. The architecture is supported by a suite of visualization and simulation tools for sequence development and verification.

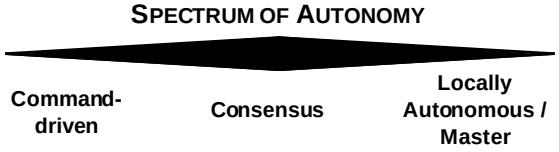
Sensible Agents: Demonstration of Dynamic Configuration of Agent Organizations for Responsive Planning Operations

K. Suzanne Barber
The Laboratory for Intelligent Processes and Systems
Electrical and Computer Engineering
The University of Texas
Austin, TX 78712
barber@mail.utexas.edu

Research Overview

Decision makers must often respond to dynamic and unexpected events. Additionally, decisions rarely occur in isolation. A decision-maker must not only assess its own possible actions but also the behaviors and resources of others possessing the ability to either assist with planning and execution, accidentally interfere, or maliciously interfere. Dynamic Adaptive Autonomy (DAA) is the fundamental technology of Sensible Agents that permits a decision making agent (responsible for planning and execution) to react, adjust, and respond to unpredictable environments. Sensible Agents can (1) assess current and potential roles others play in interactions, and (2) establish beneficial roles in these interactions. To address these issues, dynamic configuration of decision-making agent organizations is a must. Most current agent-based systems assign organizational problem-solving structures a priori. Previous research has addressed agent modification; organizational self-design (Ishida et al., 1992), partial global planning (Durfee, 1996), dynamic participation in agent groups or teams (Decker and Sycara, 1997; Tambe, 1997), and dynamic, market-based task allocation (Smith, 1980).

DAA provides control strategies to form, modify, and dissolve cooperative problem-solving agreements with other agents in a robust and flexible manner. As a member of a problem-solving organization, Sensible Agents establish their role in interacting with others by selecting an autonomy level for each goal they intend to pursue: (1) **Command driven**—agent does not plan but obeys orders given by another agent, (2) **Consensus**—agent works as a team member to devise plans, (3) **Locally Autonomous / Master**—the agent plans alone, unconstrained by other agents, and may or may not give orders to command-driven followers.



Each Sensible Agent (Barber and Martin, 1999) is composed of the following components: (1) the *Action Planner*; (2) the *Perspective Modeler*; (3) the *Conflict Resolution Advisor*; and (4) the *Autonomy Reasoner*. Domain-specific information, processing rules, and state are restricted to the Action Planner module, while remaining modules are domain-independent.

Sensible Agents are capable of performing: (1) trade-off assessment regarding the impact of local decision-making and goal satisfaction on system objectives, (2) their own behaviors by planning for a goal (local or system) and/or executing actions to achieve the goal, (3) group behaviors by forming binding autonomy agreements (e.g. consensus groups, master agent planning for group of command-driven agents) (4) self-organization by determining the best problem-solving organization, autonomy level, to optimally satisfy a goal, and (5) preferential learning for associating autonomy levels to situations. Dynamic adaptive autonomy assignments allow the most appropriate distributed agent ensembles to be defined for resource management and task performance in a dynamic or unpredictable environment.

Demonstration

The Sensible Agent (SA) Testbed provides an infrastructure of well-defined, publicly available interfaces where distributed agents operate and communicate. The end-user can interact with the testbed from the viewpoint of (1) the environment, by defining scenarios and injecting contingencies, or (2) the decision maker, by participating in planning and execution and receiving assistance from other Sensible Agents.

Sensible Agent capabilities will be demonstrated in the naval radar frequency management (NRFM) domain. This domain requires maintaining a set of position and frequency relationships among geographically distributed radars such that radar interference is minimized. Radar interference occurs primarily when two or more radars are operating in close proximity at similar frequencies. For a typical group of naval ships, it may take hours or days for a human assisted by a rule-based system to determine an optimal position and frequency. Unfortunately, the environment typically changes much faster than the human can respond. Local decisions impact the entire system,

requiring tradeoffs between local goal (e.g. keep my radars interference free) and system goals (e.g. keep radars in my group of ships interference free).

The NRFM Sensible Agent demonstration is used to determine the performance of Sensible Agents under different problem solving organizations. Agents monitor a naval radar for interference from external sources, and, if interference is detected, attempt to eliminate it by working alone or with others (Goel et al., 1998). Several different operating scenarios are demonstrated. Each Sensible Agent has the following capabilities:

Communication: the ability to send messages to another agent and to asynchronously respond to sent messages. Communication takes the form of (1) requesting information, (2) reporting a conflict, (3) supplying information, or (4) reporting a solution to a conflict.

Sensing: the ability to sense the position of other ships. Agents can also sense their level of interference, but cannot sense the source. If an agent detects interference it initiates problem solving to minimize the interference.

Environmental modeling: the ability to maintain an internal, local, model of the agent's world, separate from the simulation model of the world. Each agent is aware of the initial state of the system (ship positions and frequencies), however as the simulation progresses, an agent's local model may deviate from the world model. The agents use communication and sensing to update their local models.

Planning: the ability to plan at each of the autonomy levels described above. Successful planning for this problem hinges on an agent's ability to determine interference-free frequency assignments. Agents do this by modeling the spectrum of available frequencies and the necessary frequency differences (delta frequencies) for each known pair of radars. Agents then attempt to make assignments that meet all delta-frequency constraints within the restricted frequency space. Three algorithms are available to each agent's planner and are associated with the appropriate autonomy level classification.

An agent attempting to resolve interference in a locally autonomous fashion will plan alone. The agent will use its internal world model to find a frequency that is likely to be interference-free. The frequencies of other radars in the system are modeled as constraints on the search process. If no frequencies are found, searching continues at regular time intervals until one is found or a random "deadlock" time limit is reached. If the agent determines that the system is in deadlock (with respect to its interference state), it will choose a random frequency to pull the system out of deadlock. Note that agents acting in a locally autonomous fashion do not communicate in order to plan. However, if communication is available, locally autonomous agents may request and receive state information from other agents.

Only the master plans in a master/command-driven relationship. When the master or its command-driven agents are experiencing interference, the master attempts to eliminate the interference through iterative assignments. First, it chooses its own frequency in the manner described above, but without considering the frequencies of its command-driven agents as constraints. It then determines an interference-free frequency for each command-driven agent, adding frequencies to its constraint list, until all assignments have been made. If no set of satisfying assignments is found, the planning process is restarted. Once a solution has been found, the assignments are passed to the command-driven agents. Command-driven agents may report back to the master if they are still experiencing interference after the assignment. This may occur when the master's internal model does not match the world state.

Each agent involved in consensus interaction plays an equal part in determining frequency assignments. First, each agent independently carries out the master/command-driven planning algorithm with the other members of the consensus group treated as command-driven agents. At the conclusion of this planning phase, each agent proposes its solution to the rest of the consensus group during a synchronization phase. Along with this proposal, each agent includes an estimate (based on its internal model) of the expected interference for each radar. Each consensus member deterministically selects the proposal with the least amount of estimated interference, and the agents assign frequencies accordingly.

Acknowledgements

The research was funded in part by The Texas Higher Education Coordinating Board Advanced Technology Program, The National Science Foundation and The Naval Surface Warfare Center.

References

Barber, K. S. and Martin, C. E. 1999. Applying Dynamic Planning Frameworks to Agent Goals. Accepted to *AAAI-SSS99 Agents with Adjustable Autonomy*. Palo Alto, CA.

Decker, K. S. and Sycara, K. P. 1997. Intelligent Adaptive Information Agents. *Journal of Intelligent Information Systems* 9(3): 239-260.

Durfee, E. H. 1996. Planning in Distributed Artificial Intelligence. In *Foundations of Distributed Artificial Intelligence*, *Sixth-Generation Computer Technology Series*, O'Hare, G. M. P. and Jennings, N. R., Eds. New York: John Wiley & Sons, Inc., 231-245.

Goel, A., Liu, T. H., White, E., and Barber, K. S. 1998. Implementing Sensible Agents in a Distributed Simulation Environment. In *Proceedings of the 1998 Western Multi-Conference*. San Diego, CA.

Ishida, T., Gasser, L., and Yokoo, M. 1992. Organization Self-Design of Distributed Production Systems. *IEEE Transactions on Knowledge and Data Engineering* 4(2): 123-134.

Smith, R. G. 1980. The Contract Net Protocol: High-level Communication and Control in a Distributed Problem-Solver. *IEEE Transactions on Computers* 29(12): 1104-1113.

Tambe, M. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research* 7: 83-124.

ScienceIndex

(formerly CiteSeer)

Intelligently Augmented Search and Browsing of Scientific Literature on the Web

Kurt D. Bollacker, Steve Lawrence and C. Lee Giles
NEC Research Institute
Princeton, NJ 08540
http://www.neci.nec.com/

The future of scientific literature is expected to take the form of sophisticated digital libraries, of which the World Wide Web is one of the largest. Finding relevant scientific publications on the Web is often a challenge because of the problems of poor organization, inadequate search tools, and the large amount of literature available. ScienceIndex is a system that greatly enhances the ability of users to locate, search through, browse among, and be kept up to date on interesting Web based scientific publications. The NEC Research Institute has made the ScienceIndex software freely available and is providing a prototype service for public use.

ScienceIndex Features:

Location and Search

- **Autonomous location of articles**
  ScienceIndex uses search engines and crawling to efficiently locate papers on the Web.
- **Autonomous Citation Indexing (ACI)**
  ScienceIndex uses ACI to autonomously create a citation index, similar to the Science Citation Index, which can be used for literature search and evaluation. Compared to traditional citation indices, ACI provides improvements in cost, availability, comprehensiveness, efficiency, and timeliness.
- **Query-sensitive summaries**
  ScienceIndex provides the context of how query terms are used in articles instead of a generic summary, improving the efficiency of search.
- **Full-text indexing**
  ScienceIndex indexes the full-text of the entire articles and citations. Full boolean, phrase and proximity search is supported.
- **Name disambiguation**
  ScienceIndex allows using author initials to narrow a citation search.

Browsing and Analysis

- **Citation context**
  ScienceIndex can show the context of citations to a given paper, allowing a researcher to quickly and easily see what other researchers have to say about an article of interest.

- **Citation statistics**
  ScienceIndex provides a count of citations to a particular paper including how many different sites cite each paper and identifies self-citations.

- **Related documents**
  ScienceIndex locates related documents using citation and word based measures and displays an active and continuously updated bibliography for each document.

- **Overlapping documents**
  ScienceIndex shows the percentage of matching sentences between documents.

- **Citation graph analysis**
  ScienceIndex analyzes the graph of citations, e.g. to provide hubs and authorities ranking (a la Kleinberg).

Timeliness

- **Awareness and tracking**
  ScienceIndex provides automatic e-mail and Web based notification of new citations to given papers, as well as tracking of specific authors, title keywords, and related papers. ScienceIndex learns from user activity to enhance user profiles and recommend new potential keywords of interest. Collaborative filtering is used to leverage information from other users' profiles.

- **Up-to-date**
  ScienceIndex continuously updates from the Web so as to insure database freshness.

Public Availability

- For more details, to get a copy of the ScienceIndex software, or to use a demonstration ScienceIndex service focusing on machine learning and artificial intelligence literature, see `http://www.scienceindex.com/` or write to `scienceindex@research.nj.nec.com`.

References

Steve Lawrence, Kurt Bollacker, and C. Lee Giles. Autonomous citation matching. In Oren Etzioni, editor, *Proceedings of the Third International Conference on Autonomous Agents*, New York, 1999. ACM Press.

Steve Lawrence, C. Lee Giles, and Kurt Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 1999. accepted for publication.

C. Lee Giles, Kurt Bollacker, and Steve Lawrence. CiteSeer: An automatic citation indexing system. In Ian Witten, Rob Akscyn, and Frank M. Shipman III, editors, *Digital Libraries 98 - The Third ACM Conference on Digital Libraries*, pages 89–98, Pittsburgh, PA, June 23–26 1998. ACM Press.

Kurt Bollacker, Steve Lawrence, and C. Lee Giles. CiteSeer: An autonomous Web agent for automatic retrieval and identification of interesting publications. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, 1998. ACM Press.

SETA: an agent architecture for personalized Web stores

Project: *User Adaptive Web-based Systems*

http://www.di.unito.it/~seta/www/seta-uk.htm

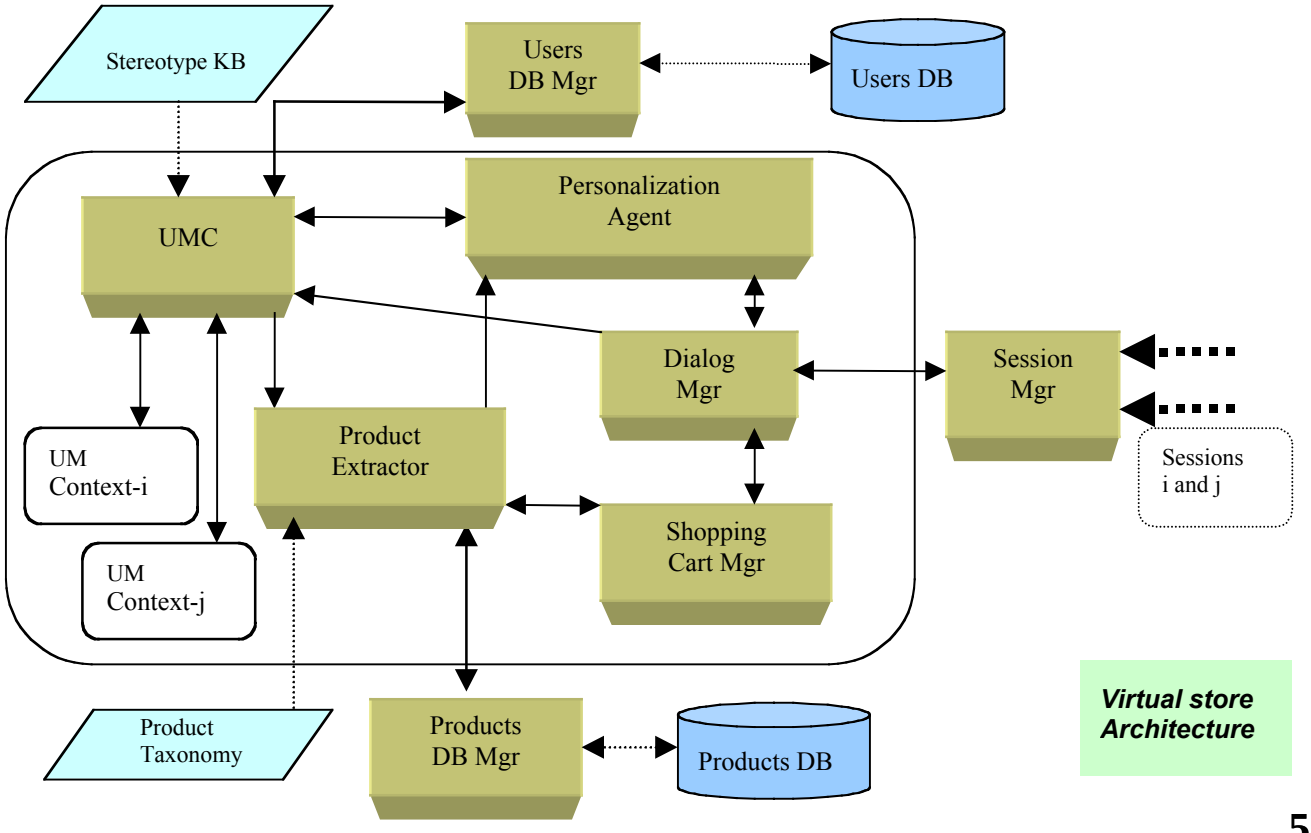
Dipartimento di Informatica (University of Torino, Italy)

Main features of the system:

- A multi-agent system for electronic commerce on the Web, offering personalized interactions with the users
- An intelligent guide through products sold in the on-line shop (the system suggests the items most suited to the customer's needs)
- Dynamic generation of product descriptions, where both the amount and the form of the information is tailored to the user's profile

Main Methodologies:

Human-Computer Interaction ♦ User Modeling ♦ Distributed Agent-based Systems ♦ Knowledge-Based Systems



Virtual store Architecture

Ongoing work:

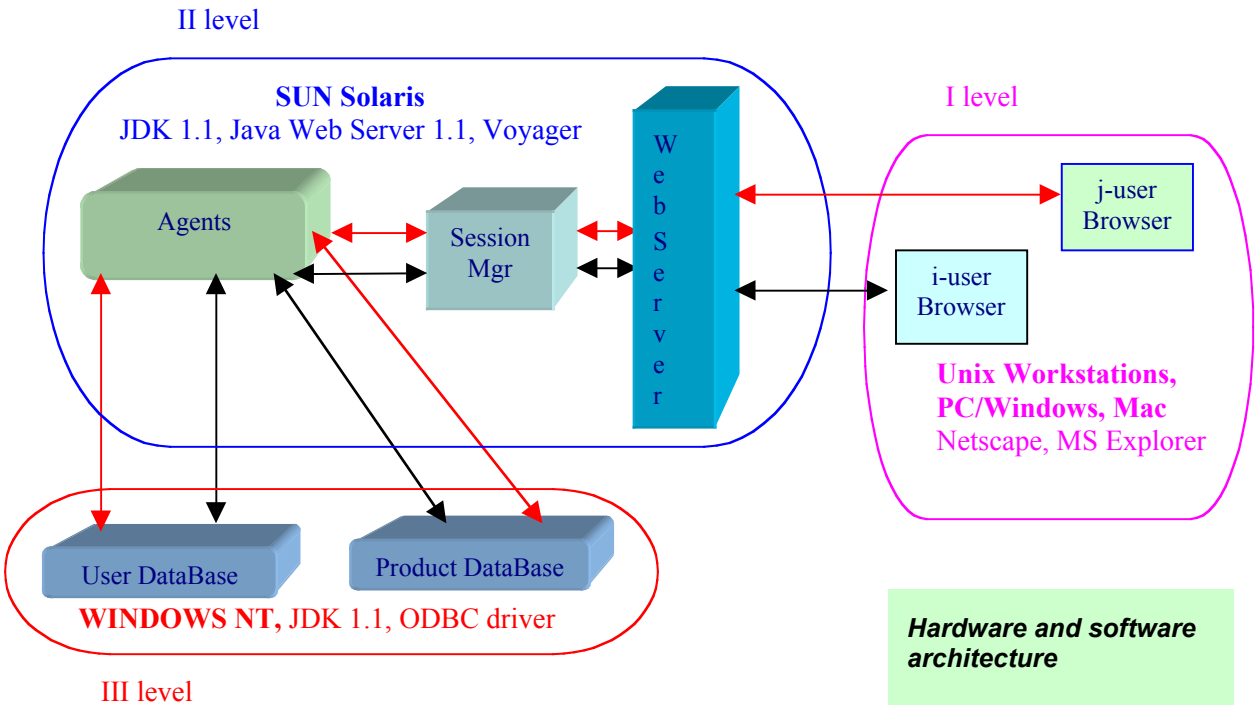
- Although the current prototype is instantiated in the telecommunication domain, the system can be configured to present products in other sales domains (two configuration tools are currenly available to design the systems's knowedge bases. Moreover, an editor to modify part of the electronic shop interface is under development).

Technical details:

Development language: ***Java***

System Architecture: ***Three Tier Application***

- **First level: *Java-enabled browsers***
  - **User Interface: *Java Applets*, *HTML***
- **Second level:**
  - ***Java Servlet***, supporting the interface to the Web
  - ***Voyager*,** for agents communication
  - **Server: *Java Web Server***
- **Third level:** databases, accessible through the Java ***JDBC***
  - The communication between the agents and the DB Managers is supported by ***Remote Method Invocation***



Hardware and software architecture

sicsDAIS. A Dynamic Agent Interaction System

Introduction

User-computer interaction has changed in the history of computing; from batch systems to command line based systems and on to directly manipulated graphical systems. There is now a need for a new change, a need to incorporate *delegation*. Delegation gives users the option to offload tasks to software systems – agents – that perform the tasks for the user. This enables users to perform tasks that are difficult to perform using graphical user interfaces, tasks such as searching and retrieving data in large distributed networks or scheduled tasks that depend on future events.

In a near future, users will have to interact with multiple agents. The question is what this interaction will be like.

One possible form of interaction is through a common, mainly graphical interface for all the agents. In such a system, users will access the agents' individual graphical user interfaces to receive information and describe and deploy tasks, while the interface application provides means for the agents to cooperate and coordinate their efforts by communicating and sharing data. Agents provide their own interfaces to SICS Dynamic Agent Interaction System (sicsDAIS)[1] as smaller versions of themselves, much as mobile agents, and sicsDAIS coordinates the presentations of these.

sicsDAIS [1] is an example of a model of one interface for many agents. It is the central point where the user interacts with all agents, but it is not a pre-defined interaction, since agents can dynamically come and go, and the methods of interaction can change.

This approach is alternative to two other approaches. In the first, all agents provide their own disparate interfaces to the user. This makes coordination and sharing of data between agents difficult. In the second, there is one interface for all agents and all agents must conform to this interface without exception. This constrains agents in their expressiveness of the interface and it makes an open system difficult to achieve.

As opposed to FIPA's suggestions for user-agent interaction in [2], we choose to view the interaction process between the user and agents as initiated and controlled by the user. The FIPA document describes the interaction as a dialog, between agents and the user, that is controlled by the agents.

Content handlers

sicsDAIS is a stand-alone Java application for simultaneous interaction with multiple agents. It combines the distributed interfaces – *content handlers* – of networked agents into one easily accessible user interface. It is a mainly graphical approach to interaction with multiple agents although content handlers may employ any means and modalities for interacting with users.

Agents are represented in the interface by the smaller *content handlers* – graphical units of Java code that may be combined in the interface to achieve the overall presentation or interaction experience for the user.

At any given moment in the course of interaction with agents, a content handler may represent a single agent or multiple agents. Conversely, several content handlers may work together to represent a single agent.

Some content handlers may even be considered "orphans", as they have no direct ties to any agents. Some of these may be invisible and they act behind the scenes in sicsDAIS, performing functions such as modeling of the user or synchronization of other content handlers or agents.

sicsDAIS

The sicsDAIS system provides several functions for content handlers and agents:

- Layout. The layout engine in sicsDAIS performs automatic layout of content handlers in the interface according to specifications from the agents.
- Data exchange. Blackboard like data exchange is available in sicsDAIS. It includes on-change notification triggers.
- Event registry. Content handlers, and through these, agents, can log all events in the system (as well as react to them).
- Event handling model. Basic Java level events in content handlers are mapped to sicsDAIS level events. Thus a third party may use a pre-constructed content handler to achieve any functionality as a result of an interface event in Java – without modifying the Java code of the content handler. The modification is done to the script that is tied to the corresponding sicsDAIS level event. Several content handlers (and thus agents) can be scripted to react to such events. Content handlers may change the event mappings during run time.
- Exception handling. sicsDAIS provides global exception handling for all content handlers. Content handlers (and agents) may register to receive notification and to handle any exceptions – even exceptions of other content handlers.
- Scripting language. Internal communication within sicsDAIS and between content handlers is provided using a scripting language that is interpreted at runtime. Scripts can be modified on the fly by content handlers (and thereby agents).
- Dynamic method invocation (as part of the scripting language). Methods in content handlers may be called while evaluating scripts.

References

1. Espinoza, F., *sicsDAIS: Managing user interaction with multiple agents*, in *Department of Computer and Systems Sciences*. 1998, Stockholm University/Royal Institute of Technology: Stockholm, Sweden.
2. Miyazaki, Y., Pohl, W., Aparicio, M., *Human Agent Interaction (DRAFT)*, . 1998: Geneva, Switzerland.

STALKER: A Hierarchical Approach to Wrapper Induction *

Ion Muslea, Steve Minton, and Craig Knoblock
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292-6695
{muslea, minton, knoblock}@isi.edu

Abstract

With the tremendous amount of information that becomes available on the Web on a daily basis, the ability to quickly develop information agents has become a crucial problem. A vital component of any Web-based information agent is a set of wrappers that can extract the relevant data from semistructured information sources. Our novel approach to wrapper induction is based on the idea of hierarchical information extraction, which turns the hard problem of extracting data from an arbitrarily complex document into a series of easier extraction tasks. We introduce an inductive algorithm, STALKER, that generates extraction rules based on user-labeled training examples. Labeling the training data represents the major bottleneck in using wrapper induction techniques, and our experimental results show that STALKER can learn high-accuracy extraction rules based on just a handful of examples.

1 Introduction

With the expansion of the Web, computer users have gained access to a large variety of comprehensive information repositories. However, the Web is based on a browsing paradigm that makes it difficult to retrieve and integrate data from multiple sources. The most recent generation of *information agents* (e.g., WHIRL (Cohen 1998), or Ariadne (Knoblock *et al.* 1998)) address this problem by enabling information from pre-specified sets of Web sites to be accessed via database-like queries. Information agents generally rely on *wrappers* to extract information from *semistructured* Web pages (a page is semistructured if the desired information can be located using a concise, formal grammar). Each wrapper consists of a set of extraction rules and the code required to apply those rules. Some systems, such as TSIMMIS (Chawathe *et al.* 1994) and ARANEUS (Atzeni, Mecca, & Merialdo 1997) depend on humans to write the necessary grammar rules. However, there are several reasons why this is undesirable. Writing extraction rules is tedious, time consuming and requires a high level of expertise. These difficulties are multiplied when an application domain involves a large number of existing sources or the format of the source documents changes over time.

In order to cope with these problems, Kushmerick(Kushmerick 1997) introduced the concept of *wrapper induction*, which is based on the idea of learning extraction rules based on user-provided examples of extraction tasks. In this demonstration, we present STALKER(Muslea, Minton, & Knoblock 1999), which is a new machine learning method for wrapper construction that enables unsophisticated users to painlessly turn Web pages into relational information sources.

2 Hierarchical Information Extraction

Because Web pages are intended to be human readable, there are some common conventions for structuring HTML pages. The information on a page often exhibits some hierarchical structure; furthermore, semistructured information is often presented in the form of lists of tuples, with explicit separators used to distinguish the different elements. For example, the document in Figure 1 provides a typical Zagat's restaurant description; besides the restaurant name, rating (i.e., food, decor, service, and cost), cuisine, and review, the document also includes a *list* of addresses and phone numbers.

With these observations in mind, we developed the *embedded catalog* ($\mathcal{EC}$) formalism, which can describe the structure of a wide-range of semistructured documents. The $\mathcal{EC}$ description of a page is a tree-like structure in which the leaves are the items of interest for the user (i.e., they represent the relevant data). The internal nodes of the $\mathcal{EC}$ tree represent *lists* of

*This work was supported in part by USC's Integrated Media Systems Center (IMSC) - an NSF Engineering Research Center, by the National Science Foundation under grant number IRI-9610014, by the U.S. Air Force under contract number F49620-98-1-0046, by the Defense Logistics Agency, DARPA, and Fort Huachuca under contract number DABT63-96-C-0066, and by a research grant from General Dynamics Information Systems. The views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of any of the above organizations or any person connected with them.

Figure 1: **A Sample Document from Zagat's.**

k-tuples (e.g., lists of addresses and phone numbers), where each item in the k-tuple can be either a leaf or an *embedded* list. In Figure 2 we show the $\mathcal{EC}$ description of a typical Zagat's document, which can be seen as a 7-tuple that includes a list of addresses, where each individual address is a 4-tuple `street, city, area-code,` and `phone-number`

Given the $\mathcal{EC}$ description of a document together with an *extraction rule* attached to each edge and a *list iteration rule* associated with each list node, a wrapper can extract any item of interest (i.e., any leaf) by simply determining the path P from the root to the corresponding leaf and by successively extracting each node $x \in P$ from its parent p. In order to extract x from p, the wrapper applies the extraction rule r that is attached to the $edge(p, x)$; if p is a list node, the wrapper has to apply first the iteration rule that decomposes p into individual tuples, and then it applies r to each extracted tuple.

Our wrapper induction tool is based on the STALKER learning algorithm. A graphical user interface allows the user to provide the $\mathcal{EC}$ description and to label the items to be extracted from a few sample documents. Based on this information, STALKER generates all the necessary extraction rules that are required in order to fully specify the wrapper.

Our approach to wrapper induction has two major advantages. First of all, the *hierarchical* extraction based on the $\mathcal{EC}$ tree allows our agent to wrap information sources that have arbitrary many levels of embedded data. Second, as each node is extracted independently of its siblings, our approach does not rely on there being a fixed ordering of the items, and we can easily handle extraction tasks from documents that may have missing items or items that appear in various orders. Consequently, in the context of using an inductive algorithm that generates the extraction

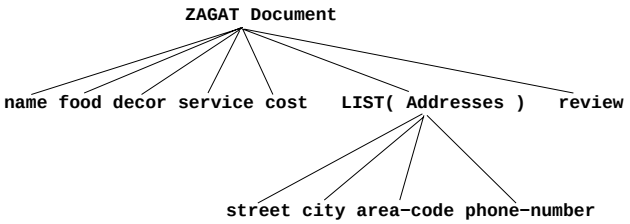


Figure 2: $\mathcal{EC}$ **description for Zagat's documents.**

rules, our approach turns an extremely hard problem into several simpler ones: rather then finding a single extraction rule that takes into account all possible item orderings and becomes more complex as the depth of the $\mathcal{EC}$ tree increases, we create several simpler rules that deal with the easier task of extracting each item from its $\mathcal{EC}$ tree parent.

3 Conclusions and Future Work

The primary contribution of our work is to turn a potentially hard problem – learning extraction rules – into a problem that is extremely easy in practice (i.e., typically very few examples are required). In order to further improve our wrapper induction system, we plan to use *active learning* techniques to minimize the amount of labeling that the user has to perform.

References

Atzeni, P.; Mecca, G.; and Merialdo, P. 1997. Semi-structured and structured data in the web: going back and forth. *Proceedings of ACM SIGMOD Workshop on Management of Semi-structured Data* 1–9.

Chawathe, S.; Garcia-Molina, H.; Hammer, J.; Ireland, K.; Papakonstantinou, Y.; Ullman, J.; and Widom., J. 1994. The tsimmis project: integration of heterogeneous information sources. *10th Meeting of the Information Processing Society of Japan* 7–18.

Cohen, W. 1998. A web-based information system that reasons with structured collections of text. *Proceedings of Autonomous Agents AA-98* 400–407.

Knoblock, C.; Minton, S.; Ambite, J.; Ashish, N.; Margulis, J.; Modi, J.; Muslea, I.; Philpot, A.; and Tejada, S. 1998. Modeling web sources for information integration. *Proceedings of the Fifteenth National Conference on ASrtificial Intelligence* 211–218.

Kushmerick, N. 1997. Wrapper induction for information extraction. *PhD Thesis, Dept. of Computer Science, U. of Washington, TR UW-CSE-97-11-04.*

Muslea, I.; Minton, S.; and Knoblock, C. 1999. A hierarchical approach to wrapper induction. *Third International Conference on Autonomous Agents.*

DEMONSTRATION DESCRIPTION

We will demonstrate Watson, the first in a class of systems called Information Management Assistants (IMAs). IMAs observe users interact with everyday applications and then anticipate their information needs using a model of the task at hand. IMAs then automatically fulfill these needs using the text of the document the user is manipulating and knowledge of how to form queries to traditional information retrieval systems (e.g., Internet search engines, newspaper archives, etc.). IMAs automatically query information systems on behalf of users as well as provide an interface by which the user can pose queries explicitly. Because IMAs are aware of the user's task, they can augment their explicit query with terms representative of the context of this task.

Watson is the first of the IMAs we have implemented. Watson is a client-side application that observes users as they browse the Web with Internet Explorer or compose documents in Microsoft Word. When a user visits a page (in Explorer) or changes a document significantly (in Word), Watson attempts to find documents related documents. Watson uses the text of the document at hand to construct a query that is sent to online information sources (e.g., Alta Vista, ProQuest, etc.). The query is composed of terms from the document that are structurally significant and have a high frequency of occurrence. When the results of Watson's query are returned, Watson removes redundant entries by clustering them according to several inexpensive heuristic similarity metrics. Watson then presents a representative from each cluster in a window for the user to browse.

Watson also attends to structural cues in documents in order to recognize opportunities to perform special-purpose search. For example, in Microsoft Word, when a user inserts a caption with no image to fill it, Watson retrieves images from an image search engine using the text of the caption inserted. Watson attends to specific structures in Web documents, as well. For example, when a user encounters a page containing a street address, Watson provides the user with access to a map of that location. These examples are part of an overall strategy of analyzing the regularities of Watson's domain of interaction in order to identify structural artifacts that indicate specific information needs.

Finally, Watson allows the user to enter explicit queries. When a user submits a query to Watson, it combines the new query terms with the query it previously constructed for finding related documents, and sends the combined query to information sources. In this way, Watson brings the implicit context of the user's current task to bear directly on the process of servicing a user's explicit query.

Watson's performance at finding related web pages was consistently better than frequent users of Internet search engines in an initial informal study. In general, we have been very excited with Watson's performance. It has been able to achieve dramatic results by augmenting explicit information requests with frequently elided contextual information. This is a great improvement over the performance of typical information retrieval systems working on their own. Moreover, Watson, and IMAs in general, significantly extend the functionality of everyday applications by integrating them seamlessly into a ubiquitous *just-in-time* information environment, providing resources to the user without requiring the construction of an explicit request. In these ways, we believe IMAs provide a compelling new framework for research in intelligent

information retrieval and contribute significantly to advancing the state of the art in human interaction with information systems.

Watson is part of an overall agenda driving research at the Intelligent Information Laboratory. Our goal is to create systems that compliment user activity in everyday environments by understanding their behavior in order to predict and fulfill their needs. Tied to this view of the world is the notion that queries to information systems, and term vectors, in general, should be treated as first class *representational objects* that can be modified and transformed by knowledge-based systems in service of a user's information need. Vector space representations of documents are particularly good for computing document to document similarity. Work on Watson shows that coupling such a representation with semantic knowledge of a particular task produces promising results.

During this demonstration, we will allow audience members to try out Watson first hand. Users will be allowed to browse and write, evaluating the suggestions Watson gives for themselves.

References

Budzik, J., and Hammond, K. 1999. Real Time Heuristic Analysis for Just-in-time Information Environments. In review, AAAI '99.

Budzik, J., and Hammond, K. 1999. Just-In-Time Information Environments. In review, AAAI '99.

Hammond, K., and Budzik, J. 1999. Watson: An Information Management Assistant. In review, Eighth International World-Wide Web Conference.

Budzik, J; Hammond, K.; Marlow, C.; and Scheinkman, A. 1998. Anticipating Information Needs: Everyday Applications as Interfaces to Internet Information Sources. In *Proc. WebNet '98: World Conference on the WWW, Internet and Intranet*.

Papers are available online at http://www.infolab.nwu.edu/~jlbudzik/watson/

Introduction

The construction of multi-agent systems involves long development times and requires solutions to some considerable technical difficulties. Hence we believe that dedicated methodologies and industrial-strength, reusable agent components are required. This has motivated our development of the ZEUS toolkit, a library of software components and tools that facilitate the rapid design, development and deployment of agent systems.

Our demonstration illustrates the three main functional components of the ZEUS toolkit, the Agent Component Library that makes possible fully featured multi-agent applications, the Agent Building Software that facilitates their construction, and the Visualisation tools that permit the applications to be observed and, where necessary, debugged.

The Agent Component Library

The agent component library is a collection of software components that implement the functionality necessary for multi-agent systems. Amongst the components provided with the ZEUS toolkit there are:

- A TCP/IP-based message passing mechanism capable of transmitting KQML and FIPA ACL performatives.
- A library of predefined co-ordination strategies, represented in the form of recursive transition network graphs; these include several variants on contract-net, and auction protocols for more commercially oriented behaviour.
- A co-ordination engine that drives agent interactions by executing co-ordination strategies.
- Support for several types of organisational relationships within agent societies.
- A general purpose planning and scheduling mechanism to support goal-driven intelligent behaviour.
- Support for agent competencies in terms of primitive actions, summary plans, forward chaining rules and self-executing behaviour scripts.
- Representations to store and exchange information on tasks and ontology concepts.
- An agent-to-legacy system interface to facilitate inter-operability with existing software systems.
- Full implementations of three different utility agents that provide runtime support services: agent name-to-network location resolution (Name Servers), service discovery (Facilitators) and persistent storage (Database Proxies).

By providing a set of high quality, pre-written and pre-tested agent components, we hope to liberate developers from the minutiae of agent technology, allowing them to concentrate on solving their application's problems instead.

The Agent Building Software

The ZEUS toolkit provides an integrated suite of editors that guide developers through the stages of our comprehensive agent development methodology. During this process developers describe the agents within their application, how they interact, and the tasks they perform. Amongst the tools are:

- An Ontology Editor for defining the concepts, attributes and constraints within a domain.
- An Agent Definition Editor for describing agents logically, e.g. their tasks, initial resources, planning abilities etc.
- A Task Description Editor for describing the attributes of tasks and for graphically composing summary tasks.
- An Organisation Editor for defining the organisational relationships between agents, and agents' beliefs about the abilities of other agents.
- A Co-ordination Editor for selecting the set of co-ordination protocols with which each agent will be equipped, and the strategies that influence the agent's behaviour.

Once defined the ZEUS Code Generator tool can automatically convert the agent definitions into executable Java source code, enabling applications built with ZEUS to run on any hardware platform

The Visualisation Tools

The Visualisation Tools collect information on agent activity, interpret it and display various aspects in real-time. This is an attempt to solve the inherently difficult problem of analysing and debugging a multi-agent system where all of the data, control and active processes are distributed; to this end the following tools are supplied:

- A Society Viewer that shows all known agents, their organisational relationships and the messages they exchange.
- A Reports Tool that shows the society-wide decomposition/distribution of active tasks and the execution states of the various tasks.
- An Agent Viewer that enables the internal states of agents to be observed and monitored.
- A Control Tool that is used to remotely review and/or modify the internal states of individual agents.
- A Statistics Tool that displays individual agent and society-wide statistics in a variety of formats.

The multi-perspective approach provided by the visualisation tools gives users the flexibility to choose what is visualised, how it is visualised and when it is visualised. In addition, as well as viewing events as they happen, the Visualiser tools can save agent sessions for offline analysis using their 'Video Replay' facilities.

Further information and a link to download the ZEUS toolkit can be found on our web site at:

http://www.labs.bt.com/projects/agents/

Contact Information

Dr. Divine Ndumu

MLB1, PP12,
BT Laboratories,
Martlesham Heath, Suffolk,
United Kingdom, IP5 3RE

Email: ndumudt@info.bt.co.uk
Telephone: +44 (01473) 605666

Dr. Jaron Collis

MLB1, PP12,
BT Laboratories,
Martlesham Heath, Suffolk,
United Kingdom, IP5 3RE

Email: jaron@info.bt.co.uk
Telephone: +44 (01473) 605462