PLANNING AND MATCHMAKING IN A MULTI-AGENT SYSTEM FOR SOFTWARE INTEGRATION

Aurora C. Díaz[*], Stephen J. Cranefield[**], and Martin K. Purvis[**]

[*]Institute for Information Technology, National Research Council Canada,
M50 Montreal Road, Ottawa, Ontario, Canada K1A 0R6
Phone: (1-613) 993-8560 Fax: (1-613) 952-7151
E-mail: Aurora.Diaz@nrc.ca

[**]Department of Information Science, University of Otago,
PO Box 56, Dunedin, New Zealand
Phone: (64-3) 479-8142 Fax: (64-3) 479-8311
E-mail: {scranefield, mpurvis}@commerce.otago.ac.nz

ABSTRACT

Computer users employ a collection of software tools to support their day-to-day work. Often the software environment is dynamic with new tools being added as they become available and removed as they become obsolete or outdated. In today's systems, the burden of coordinating the use of these disparate tools, remembering the correct sequence of commands, and incorporating new and modified programs into the daily work pattern lies with the user. This paper describes a multi-agent system, DALEKS, that assists users in utilizing diverse software tools for their everyday work. It manages work and information flow by providing a coordination layer that selects the appropriate tool(s) to use for each of the user's tasks and automates the flow of information between them. This enables the user to be concerned more with what has to be done, rather than with the specifics of how to access tools and information. Here we describe the system architecture of DALEKS and illustrate it with an example in university course administration.

KEYWORDS

Agent architecture, software interoperability

INTRODUCTION

The day-to-day work of many people involves the use of a continually changing set of software tools. These may include general-purpose utilities designed to support work in many domains, such as word processors and spreadsheets, as well as special-purpose tools designed for the user's problem domain. Currently the onus is on the user to manage workflow and determine how the different software tools work together to achieve an objective. We are developing a multi-agent system, DALEKS ("Distributed Agents Linking Existing Knowledge Sources"), that assists users by facilitating the interoperation of diverse and distributed software tools used by a person in his or her daily work.

A software tool produces and/or consumes information, therefore, making two tools interoperate involves matching the information produced by one to the information consumed by the other. This matching process should work in an open environment where the set of tools available for use is constantly changing. A key issue to address is how to abstract away from the formats of information sources and the protocols used to access them (e.g., reading a file or getting information from a database) so that the system works in an environment where tools may be dynamically added or removed, and where these may produce information in differing formats. Our approach separates task selection (planning) from tool selection (matchmaking). Planning determines the tasks to be done to achieve the user's objective and manages workflow, whereas matchmaking selects the tool to use to perform a task in the plan and manages information flow between the tasks.

This paper describes the DALEKS system architecture focussing on how planning, matchmaking, and execution are interleaved to help automate the user's tasks.

SYSTEM ARCHITECTURE

DALEKS uses an extended version of the *federation architecture* used in previous research on Agent-Based Software Interoperability [Genesereth *et al.*, 1995]. In the federation architecture, software tools and information servers are encapsulated as *agents* that receive and reply to requests for services and information. These agents use a declarative knowledge representation language, KIF (Knowledge Interchange Format), an inter-agent communication language, KQML (Knowledge Query and Manipulation Language), and a library of formal ontologies defining the vocabulary of various domains. A federated system of agents includes *facilitators* that receive messages and forward them to the most appropriate agent depending on the content of the message. A new tool is added to the system by providing it with a *wrapper* or a *transducer* and then registering it with the facilitator. A wrapper adds code to the tool itself to allow it to communicate with other agents using some agent communication language. A transducer is a separate piece of code that acts as an interface to the tool and translates messages in an agent communication language to the tool's own communications protocol.

Previous work [Cranefield and Purvis, 1995; 1997] has proposed extending the federation architecture to provide basic work and information flow by adding a specialized planning agent to automate the coordination of tools on behalf of the user and a user agent that acts as the interface between a user and the system. The DALEKS system builds on this work and investigates in more depth the mechanisms of planning and matchmaking required to support automated interoperation of tools in an open and extensible software environment.

To demonstrate DALEKS a prototype for university course administration is being developed. In this domain, information processing and management tasks include the addition or deletion of students from the class roll, marking student assignments, changing marks when necessary, producing statistical summaries, etc. Information may be created, deleted, or modified at each stage of the process. At the University of Otago, these tasks are performed using a toolkit approach, whereby the course administrator uses a number of different tools to perform the tasks, some being general-purpose tools and others being specially written for work in this problem domain. Figure 1 shows the system architecture of the current prototype, where some agents run under Windows NT and others under Solaris. The system consists of custom-built facilitator and user agents and agent-encapsulated tools including a planner, utilities for manipulating text files, a DBMS, and a marking tool that enables a tutor to systematically find, run, and record marks for electronically submitted programming assignments. The figure also shows data, such as methods, plans, Uniform Resource Characteristics (URCs), and operator specifications, that are stored in the user and facilitator agents. Inter-agent communication is via KQML.

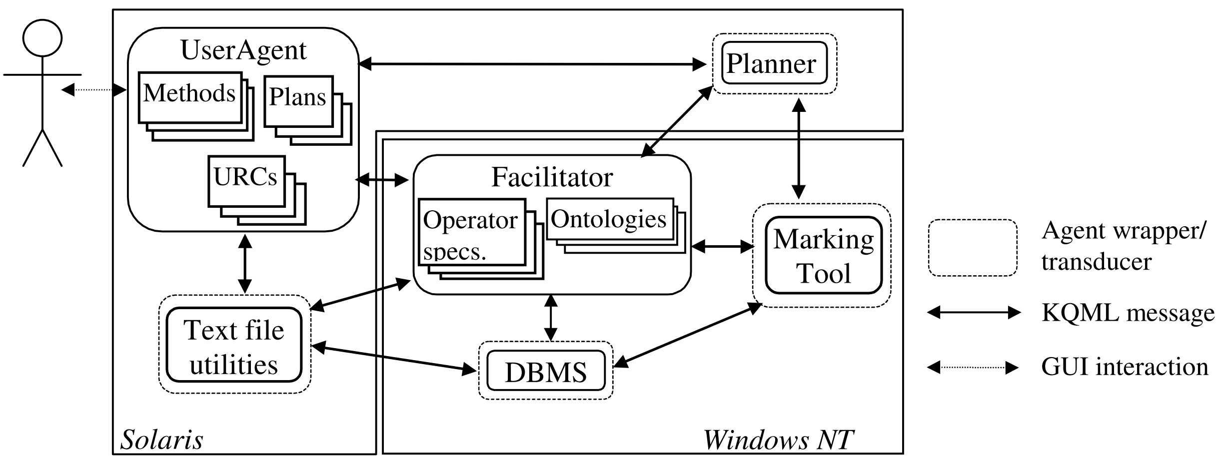

Figure 1. Prototype system architecture

TASK SELECTION: PLANNING

DALEKS assists a user who is attempting to solve problems in a particular domain. In this system, the planning agent [Cranefield *et al.*, 1997] manages workflow and selects the tasks that have to be done to accomplish a goal that solves a problem. Before planning can be done, the user must provide the system with ontologies, operator specifications, and method definitions. This input, together with other information produced by DALEKS, is stored in different agents, as shown in Figure 1.

Before using the DALEKS system, the user must create an ontology defining the vocabulary specific to the user's domain. We believe that for many domains, simple representations such as relational data models will suffice [Cranefield and Purvis, 1995]. This domain ontology, together with other more generic ontologies, such as those describing common data formats, is stored in the facilitator.

As part of the task of domain definition, the user must also define the generic actions that can be performed in the domain. An example for the university course administration domain is the mark action that generates marks for a given assignment for a set of students. These generic actions are specified by planning-style operator specifications [Cranefield *et al.*, 1997] containing the name of the action, its pre- and post-conditions, and information about its information requirements and products expressed in terms of the domain ontology.

In addition to these domain-specific actions, some operators corresponding to domain-independent actions are predefined in the DALEKS system. These include the retrieval and update of relations (specified by relational algebra expressions) stored in an information source.

Tools to be used in a DALEKS system must also have one or more operators defined. Each operator describes a particular task that can be performed by the tool. An operator may specify how the tool can be used to achieve one of the generic tasks in the user's domain, in which case its definition should be a specialization of the corresponding generic operator with additional information describing the formats and access protocols of its information requirements and products. In addition, tool operators may describe how the tool can be used for actions that are not domain-specific, such as text file transformation operations. In this case the operator specification is given in terms of some general-purpose ontology such as one describing text file formats. When a tool agent is added to the system, it advertises its capabilities to the facilitator by sending it messages that contain these operator specifications. The facilitator stores all the operator specifications of the currently available tool agents and these are used in the tool selection process. A graphical user interface will be developed to support non-expert users in these activities.

By providing a task to plan for, the user triggers the planning process. The planning agent in DALEKS is based on hierarchical task network (HTN) planning [Erol *et al.*, 1994], where a task hierarchy is developed using user-defined methods that describe possible ways of decomposing a task. In addition to methods, operator specifications must also be provided for primitive actions or tasks. Using this information, the planner determines the tasks and its ordering(s) that, when executed, accomplish the goal. We chose HTN planning because the intended users of DALEKS already have strategies for coordinating their tools, which are used to define the methods that expand the task hierarchy.

Plans developed by the planner identify the tasks that have to be performed to achieve a goal. They do not include detailed information, such as which agent is to execute a primitive task, how it is to be executed, and what information flows between tasks in a plan. The resolution of these details is left as late as possible, when the task is about to be performed. This way, the current environment is taken into account when deciding how to execute a task.

TOOL SELECTION: MATCHMAKING

We use the matchmaking approach [Kuokka and Harada, 1995] to determine how the tasks in the plan are to be executed and to find potential information sharing paths between information providers and consumers. In matchmaking both consumers and providers play active roles; providers *advertise* their capabilities to the matchmaker and consumers send *requests* to the same matchmaker. The matchmaker matches advertisements to requests.

Kuokka and Harada [1995] identify several modes of matchmaking that differ in the route information sharing takes. There is the *recommend* mode where the consumer asks the matchmaker to recommend a provider for a particular request, with the consumer directly communicating its request to the provider. In the *recruit* mode, the consumer asks the matchmaker to forward its request to the appropriate provider, with all replies going straight back to the consumer. In the *broker* mode, the matchmaker acts as an intermediary between the consumer and the provider, with the request and replies passing through the matchmaker. DALEKS mostly uses the recruit mode.

In DALEKS the facilitator takes on the role of matchmaker. Agents that can perform tasks serve as providers and agents that request tasks to be done for them act as consumers. When an agent is added to the system, it must tell the facilitator what it can do and what services it can provide. A consumer's request for service, sent to the facilitator, gives the name of the task to be done, and, optionally, a list of preferences, which are used by the facilitator to choose among providers with capabilities to perform the request. There are mechanisms in DALEKS to handle simple preferences, such as naming a preferred provider, preferring one that is most recent (i.e., the newest provider in the system as determined by the time and date of its advertisement), or choosing the provider that can understand a specific language or ontology. The consumer may also prioritize its preferences, which the facilitator uses when selecting a tool. Upon receiving a request, the facilitator locates and selects a provider using the providers' advertisements and consumer's preferences. It then forwards the request to the selected provider for execution. Results of the request are sent directly back to the consumer.

Plan execution that leads to achieving a goal is triggered in DALEKS by a user agent (UA) upon the user's request. For each task to be performed, UA sends a KQML message to the facilitator to recruit a tool agent that can do the requested task. The facilitator, using the advertisements it has received from the different agents and (optionally) the preferences sent by UA, selects an appropriate one for the current task. After tool selection, it checks if the selected tool agent requires any input, which it knows from the tool agent's advertisement. If no input is required, the facilitator forwards the request (contained in the content of the recruit KQML message) to the selected tool agent. After performing the task, the tool agent replies back to UA with the outcome and a description of the information or data it has produced. This description comes as URCs (Uniform Resource Characteristics) [LAN-ACL, 1995] that provide meta-data about the information source, including the URL (Uniform Resource Locator) that specifies where the resource may be found and a description of its contents and physical form properties, such as its data format and access protocol. The actual resource (e.g., files and databases) is kept with the tool agent that produced it. When UA receives the reply, it goes on to the next task in the plan.

Figure 2 illustrates part of the information-sharing path in the university course administration prototype, which follows the recruit mode of matchmaking. In this figure, KQML messages sent between agents only show the KQML *performative*, e.g., ask, reply, achieve, and the *content*, enclosed in parentheses and specified using informal notation.

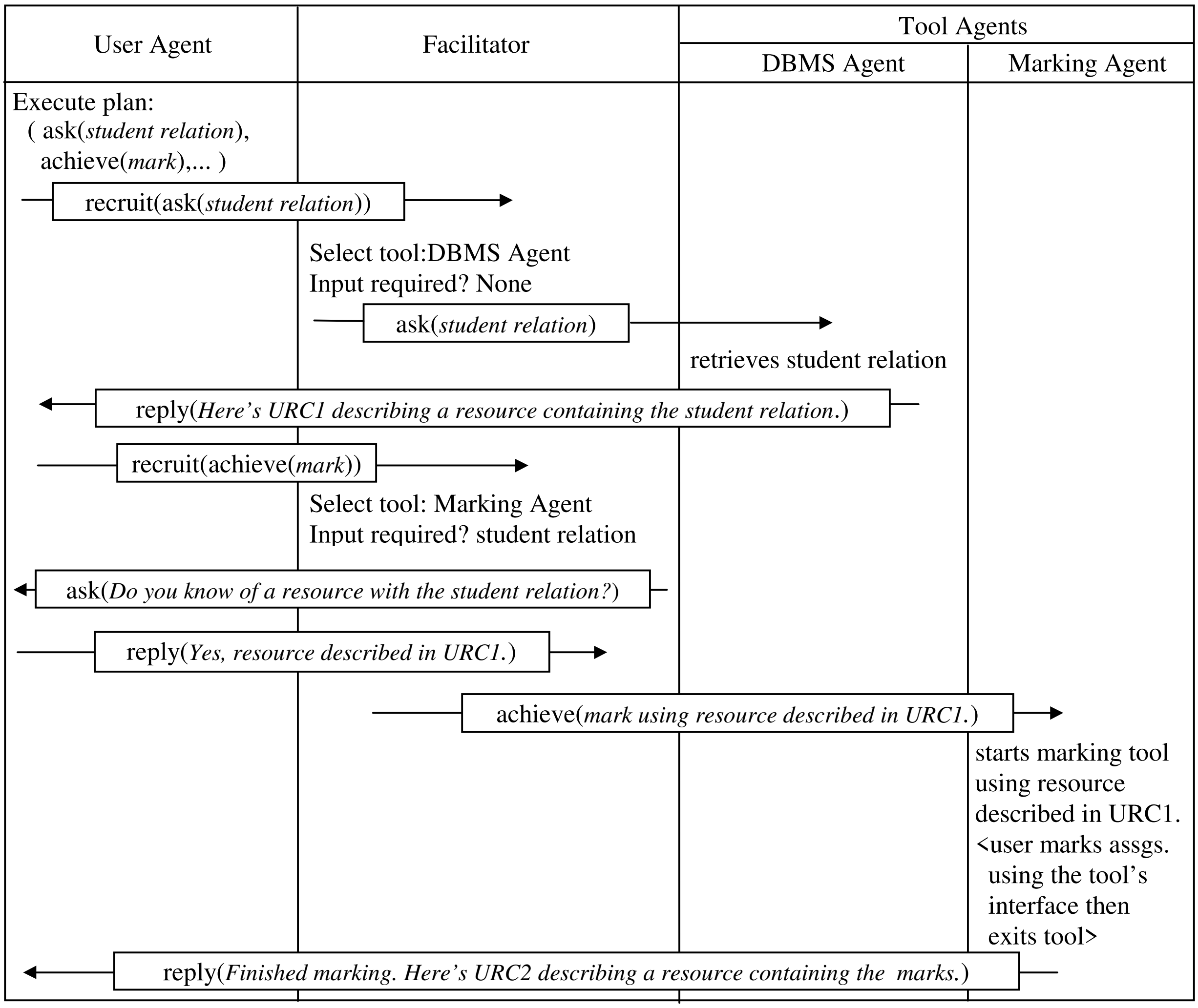

Figure 2. Information routing during plan execution in DALEKS

If the facilitator finds that a selected tool agent requires input, it sends a KQML message to UA to ask if any of the previous plan steps executed has resulted in the creation of information that matches the input requirement. UA searches the URCs describing the information produced so far for a match. If found, UA passes the URC of the resource containing the information back to the facilitator. The facilitator, when forwarding the request to the tool agent, specifies where the tool agent may find its input requirements. The tool agent takes care of locating this and accessing the information it requires.

If an information source with the input requirement does not exist (for example, none of the information currently produced matches the intellectual content or format of the input required), the facilitator invokes the planner to derive a plan that will create or modify existing information sources to match the requirement. UA still serves as the consumer of this planning task so the planner will reply to UA with the plan, which then executes it. This is one situation where planning is interleaved with plan execution. Another is when the plan being executed is not fully reduced, i.e., it contains non-primitive tasks that can be further decomposed by some method. Planning is initiated during execution to further elaborate the plan.

DISCUSSION

In DALEKS, an agent does not have information about other agents. For example, it does not know about the intentions, goals, or plans of other agents; it only knows its own plans that it uses when performing tasks asked of it. Only the facilitator keeps a model of the other agents in terms of their capabilities and defined as operator specifications. We do not touch on predetermined and pre-negotiated commitments except in assuming that if an agent says it has a capability then it can perform the tasks corresponding to the capability and will do so when asked. The facilitator will select only from the tools currently available in the system; therefore, agent-encapsulated tools may be added and removed without having to change the other agents in the system.

Although nothing in our architecture precludes having multiple facilitator and planning agents, our prototype has only one facilitator and one planner. We realize that these may become bottlenecks in a system with a large number of software tools, information servers, and users. Future work involves investigating scale-up issues including devising ways of organizing multiple facilitators and planners.

Not all tools will operate using the same ontology. Different agents, particularly the tool agents, may use domain-specific ontologies or other more generic ones not only to describe their capabilities but also to work in. Facilitating the interoperation of tools that use different ontologies is another issue for future work.

This paper presents our approach to providing a tool for integrating the use of various software tools. We envision this to be a component of the middleware layer in open systems that allows the user to access applications or software tools, just as the object component provides services for accessing different objects.

REFERENCES

[Cranefield and Purvis, 1995] S. J. S. Cranefield and M. K. Purvis. Agent-based integration of general-purpose tools. In *Proceedings of the Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management*, 1995. Also in http://www.cs.umbc.edu/~cikm/iia/proc.html.

[Cranefield and Purvis, 1997] S. J. S. Cranefield and M. K. Purvis. An agent-based architecture for software tool coordination. In L. Cavedon, A.S. Rao, and W. Wobcke, editors, *Intelligent Agent Systems: Theoretical and Practical Issues,* Lecture Notes in Artificial Intelligence, number 1209, pages 44-58. Springer, 1997.

[Cranefield *et al.*, 1997] S. J. Cranefield, A. C. Diaz and M. K. Purvis. Planning and matchmaking for the interoperation of information processing agents. Discussion Paper 97/1, Department of Information Science, University of Otago, 1997. Submitted to the European Conference on Planning 1997.

[Erol *et al.*, 1994] K. Erol, J.Hendler, and D.S. Nau. UMCP: A sound and complete procedure for hierarchical task-network planning. In K. Hammond, editor, *Proceedings of the 2nd International Conference on AI Planning Systems*, pages 249-254, 1994.

[Genesereth *et al.*, 1995] M.R. Genesereth, N.P. Singh, and M.A.Syed. A distributed and anonymous knowledge sharing approach to software interoperation. *Int. Journal of Cooperative Information Systems,*4(4):339-367, 1995.

[Kuokka and Harada, 1995] D. Kuokka and L. Harada. Matchmaking for information agents. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 1, pages 672-678, 1995.

[LAN-ACL, 1995] Uniform Resource Characteristics Web page, Advanced Computing Laboratory, Los Alamos National Laboratory. http://www.acl.lanl.gov/URC/, November 1995.