Planning and Matchmaking for the Interoperation of Information Processing Agents

Stephen Cranefield, **Aurora Díaz** and **Martin Purvis**
Department of Information Science, University of Otago
P.O. Box 56, Dunedin, New Zealand
{scranefield,adiaz,mpurvis}@commerce.otago.ac.nz

Abstract

In today's open, distributed environments, there is an increasing need for systems to assist the interoperation of tools and information resources. This paper describes a multi-agent system, DALEKS, that supports such activities for the information processing domain. With this system, information processing tasks are accomplished by the use of an agent architecture incorporating task planning and information agent matchmaking components. We discuss the characteristics of planning in this domain and describe how information processing tools are specified for the planner. We also describe the manner in which planning, agent matchmaking, and information task execution are interleaved in the DALEKS system. An example application taken from the domain of university course administration is provided to illustrate some of the activities performed in this system.

1 Introduction

With the open and increasingly interconnected world of information sources, the problem of how to make effective use of the information in a heterogeneous and distributed environment is a significant challenge. A characteristic of this emerging environment is that new information sources and processing tools continually become available and can make existing ones obsolete. Although facilities exist for searching the Internet, there remain problems of organising and effectively using information in such a dynamic environment.

One approach is to encapsulate tools and sources by means of software agents that are capable of providing information to each other concerning their specific capabilities. The advantage of such an approach, where tools and information sources can communicate using a common agent-communication protocol, such as KQML [Genesereth and Ketchpel, 1994], is that the user need not be burdened with the details of how to address each of the individual information sources. This enables the user to be concerned more with the demands of the problems at hand, rather than with the specifics of how to access tools and information. It also supports the frequently necessary "toolkit" approach to problem solving, whereby a number of different tools, some general-purpose and some custom-built, are combined to achieve a user's information processing goal. This toolkit approach is a typical solution to information processing and management problems at many sites, especially where legacy systems must be used.

Although the notion of a software agent architecture offers advantages concerning information access in a dynamic environment, there still remains the problem of how to select and organise the appropriate tools in order to carry out a particular task. This paper discusses the DALEKS ("Distributed Agents Linking Existing Knowledge Sources") system, which addresses this problem by offering planning and matchmaking facilities to facilitate the goal of information processing in a dynamic, open environment.

Other researchers have investigated the use of planners in software agent frameworks [Golden *et al.*, 1994; Knoblock, 1995; Kwok and Weld, 1996; Williamson *et al.*, 1996], but this work has primarily focused on planning purely for information *gathering* actions (which do not change the world state except to increase the agent's knowledge of an external but static body of information). The work presented in this paper is concerned not only with planning for information gathering tasks, but also with planning for information processing tasks, where information can be created or altered by the actions of agents.

Planning issues associated with information processing domains are discussed in Section 3, the manner in which planning, matchmaking and execution are interleaved in the DALEKS system are discussed in Section 4, and an illustrative example of the system applied to an application in university course administration is discussed in the Section 5. Related work and conclusions are covered in Sections 6 and 7.

2 Organising Information Processing Tasks

Previous work [Cranefield and Purvis, 1995; 1996] has proposed a "desktop utility agent architecture" to automate the

interoperation of disparate and distributed tools to achieve a user's information processing tasks. This architecture extends the federation agent architecture developed by researchers in the field of agent-based software engineering [Genesereth and Ketchpel, 1994] by adding a planning agent and a user agent. These components, in conjunction with the federation architecture's facilitator agent, are used to automate the task of determining when and how to access information or tools so that the user is shielded from the constant change associated with the available tools and information resources. In the DALEKS system discussed in this paper, information processing tasks are organised by separating the processes of task selection (via planning) from that of tool selection (via matchmaking). The planning agent chooses the appropriate generic information processing tasks to be performed, and the facilitator agent then acts as a matchmaker, deciding which tool can be used to perform each task in the plan. In this paper the focus of the discussion is on

- the modelling of information processing tasks for the purposes of planning,
- the representation of information resources, and
- the protocol for distributed control of interleaved planning and execution through the cooperation of a user proxy agent, a Hierarchical Task Network (HTN) planning agent and a facilitator (matchmaker) agent.

3 The Formal Model

A natural way to model problems in the information processing domain is to imagine an abstract "information state" that contains at any moment the theoretical information content of the domain. As a conceptual entity, this information state is not accessible by agents but instead serves as a reference to which the contents of real physical information resources can be compared during planning. With this model, the pre- and postconditions of planning operators must declare how the operator affects the information state, what resources are required and produced by the action, what information is contained in them, and to what stage of the complete information processing task the information corresponds.

3.1 The Model of Action

Traditionally in planning, actions are modelled as functions that map a prior world state to the state of the world after the action is performed. The world state is modelled by a set of facts that hold in that state.

For the DALEKS system we wish to make explicit the role played by the information resources required and produced by actions. First, we assume there is a domain ontology, called the *Domain Information Model*, describing the language used to represent information about the domain. Currently we restrict this to being a relational data model, including information about the candidate keys and foreign key relationships between the base relations in the domain. We then model the world state as consisting of three components:

The Information State This represents the current state of the information processing domain. As actions are performed, information may be created, become invalid or be altered. For example, in a university course administration domain a "mark assignments" action will create new information: the marks awarded to each student for the given assignment. A "re-mark assignment" action will change the mark awarded to the given student for that assignment. The information state represents the theoretical information content of the current state. It can be thought of as "God's database" — this information cannot be directly known by agents; they must instead access physical resources that they know contain an up-to-date copy of the information they require. For example, although the "mark assignments" action creates information, the only way that information can be accessed later is if the action stores the marks in some physical resource (such as a file or database).

The Resource Pool This is a set of descriptions of available information resources. These descriptions specify the location of the resource, the protocol used to access it, the information contained by the resource, the ontology used to express the information, and the (possibly prior) state described by the information (resources may become out of date as information processing actions are performed). The contents of resources are described by relational algebra expressions in terms of the base relations defined in the Domain Information Model.

These resource descriptions are collections of metadata and can be represented as Uniform Resource Characteristics (URCs) [LANL-ACL, 1995].

The Local State This is a set of facts as traditionally used in planning to model the world. Here it represents the aspects of the world not represented by the Information State and the Resource Pool, such as the physical world and the agent's mental state. The Local State is directly observable by agents.

3.2 Operator Specifications

As usual in planning, the domain actions are described by operator specifications specifying the actions' parameters, preconditions and effects. In accordance with our decomposition of the world state into three components, the operator's preconditions and effects are expressed separately for the Information State, Resource Pool and Local State (see Figure 1 for an example of an operator specification).

The Information State part of the operator specification describes the theoretical information processing operations performed by the operator. New information may be generated, even if this is not recorded anywhere for future use. The operator specification lists the relations of the Domain Informa-

tion Model that are affected by the operator and describes the changes to these relations using a set of constraints.

The resources required and produced by the operator are also listed. At run time, each resource is described by a URC which lists the resource's location and other metadata such as its intellectual content and format. The operator's pre- and postcondition resource specifications list can therefore specify values for any of the tags that may appear in a resource's URC, in particular, the required intellectual content of the resource can be expressed as a relational algebra expression in terms of the Domain Information Model.

Finally, the Local State may be specified in terms of preconditions and an add and delete list and is treated as in traditional planning frameworks. The Local State is not used in the example discussed in this paper; however, its inclusion in the formal model gives users the option of representing some of an operator's preconditions and effects using simple facts without having to define their format in the Domain Information Model.

Figure 1 shows an example operator specification in the domain of university course administration.

mark(Ass, Class_IDs, !ID_Set)

Information State
affects: assess.
constraints:
rel(assess, After) = rel(assess, Before) ∪ NewData,
key_values(NewData, stu_cmpt) = !ID_Set × {Ass},
!ID_Set ⊆ Class_IDs.

Resources

pre:
R1 : ontology = course_data,
content = rel(student, Before).
post:
R2 : ontology = course_data,
content = rel(select(assess, [stu_id ∈ !ID_Set,
cmpt_id = Ass]),
After).

Local State No pre- or postconditions

Figure 1: An example operator specification

The Domain Information Model for this example is a relational data model called `course_data`. It includes the base relations `student` (recording student details including the student identification number — attribute `stu_id`) and `assess` (which records student marks for each assessment component, and has the attributes `stu_id`, `cmpt_id` and `mark`).

The operator `mark(Ass, Class_IDs, !ID_Set)` represents the invocation of an interactive tool that allows a tutor to systematically run each student's submission for a particular programming assignment (`Ass`) and record a mark for it. Not all submissions may be marked in one session, so the third argument, a list of student IDs, is declared (using the "!" prefix) to be a run-time variable [Ambros-Ingerson and Steel, 1988] — one that will be instantiated at run time to indicate which students' assignments were marked during the execution of this operator.

The `affects` clause in the Information State part of the specification is used to address the frame problem for the Information State. It states that the operator only alters the relation `assess`.

The first two Information State constraints specify how this operator changes the relation `assess` (using the special variables Before and After to name the states before and after this operator is executed). In brief, this operator is declared to create new information in the Information State: marks for the assessment component `Ass` for all students in `!ID_Set`. The first constraint declares that the contents of the `assess` relation after the operator executes is the union of the contents of `assess` beforehand, and a set of new tuples, represented by the variable `NewData`. This set is then constrained to consist of a tuple for each student in `!ID_Set`, with each tuple having its `cmpt_id` attribute equal to `Ass` (the constraint states that the set of values in `NewData` for the key `stu_cmpt`, consisting of the pair of attributes `(stu_id, cmpt_id)`, is equal to the cross product of `!ID_Set` and the singleton set `{Ass}`).

Integrity constraints in the Domain Information Model also cause implicit constraints to be asserted when this operator is used, for example, there should not already be marks in `assess` for component `Ass` and the students in `!ID_Set`, otherwise the new set of tuples in `assess` would violate the constraint that `stu_id` and `cmpt_id` together are the primary key for `assess`.

The final part of the operator specification declares the resources required and produced by this operator. Before the operator is executed, there must be a resource available containing up-to-date information about the students taking the course (the student relation). Once the operator has been executed, the resource postcondition guarantees the existence of a physical resource containing the new information. The content of this will be a subset of the tuples that will now belong to the Information State, in particular, those representing the new marks that have been awarded.

3.3 Information State Constraints

Information State constraints do not necessarily completely specify the information processing process performed by the operator. They can instead be regarded as a specification that valid implementations of the operator must satisfy. For the example above, the assignment marks for each student are left unspecified. It is not for the planner to completely simulate the marking process, instead its role is to determine when

to perform the marking task as a step in a larger process. In our current framework, a hierarchical task network (HTN) planner [Erol *et al.*, 1994] is used and the incorporation of tasks into the plan is performed by the application of user-defined methods to expand tasks into sequences of subtasks[1]. The task networks used to represent plans in an HTN planner consist of a set of compound (as yet unrefined) and primitive tasks, together with a set of constraints on their ordering, variable bindings, etc. Incorporating each selected operator's Information State constraints into the current task network and defining appropriate constraint reduction rules is a straightforward extension of this procedure.

3.4 Constraints in Task Reduction Methods

The extension of task networks to include more general types of constraint (compared to existing HTN planners, *e.g.* [Erol *et al.*, 1994]) also allows task reduction methods to be more expressive. For instance, Figure 2 shows a method for splitting an assignment marking task into two parts: first to mark the submitted course assessment component (named `Cmpt`) for students whose identification numbers appear in the set `!ID_Set1`, and then to (recursively) mark the rest of the students' work. When this method is

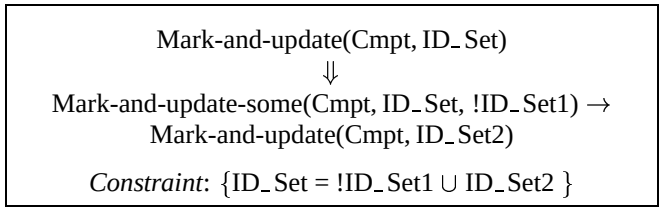

Figure 2: An example task reduction method

used, it is intended that the task `Mark-and-update` will be called with `ID_Set` instantiated. `!ID_Set1` is a run-time variable and its value will only be known after the task `Mark-and-update-some` has been executed (this task involves executing a tool which a tutor can use to systematically mark student assignments, however it cannot be predicted in advance how many assignments will be marked in a single session). The constraint ensures that `ID_Set2` will then become instantiated to the set of identification numbers for students whose work have not yet been marked.

4 Planning, Matchmaking and Execution

In information processing domains, the user's task of achieving some information processing goal involves the use of a variety of software tools, where each tool can be seen to produce or consume information. Making two tools interact involves matching the information produced by one to the information consumed by another. The DALEKS system facilitates the interoperation of diverse software tools by explicitly representing and reasoning about the properties of its information resources, in particular, their intrinsic properties which include intellectual content and physical form.

Two tools may create and use information resources with similar intellectual content but of differing physical forms. Intellectual content properties of an information resource can thus be seen as being more general (less tool-specific) than its physical form properties (such as the format of the information and the protocol used to access it). DALEKS takes advantage of this by separating information processing task selection (planning) from tool and resource selection (matchmaking). Planning, which determines what has to be done to achieve a goal, uses the more general intellectual content properties. Matchmaking, on the other hand, also includes the more variable physical form properties in its reasoning process. Extracting and explicitly representing the intellectual content produced or consumed by an information source, and separating it from its physical form, provides physical data independence, thus protecting a plan from the more variable aspects of an information source and allowing plans to be made independent of the physical forms used.

Execution in the DALEKS system consists of three steps: (1) determining the information processing tasks required to achieve the user's goal, (2) for each task, selecting a tool to use to execute it, and (3) invoking the selected tool.

The control is distributed among three special-purpose agents: the planner, the facilitator and the user agent.

Planner This determines the sequence of information processing steps to perform. It generates a plan that does not specifically identify the resources to use or tool to invoke for each step in the plan. Instead, it specifies what has to be done in a plan step and the intellectual content of both the input and output of the plan step (as described above in the operator specification section). The domain specifications needed by the HTN planner form the input required by the planning agent. These are passed to the planner from the agent requesting the plan via the facilitator.

Facilitator This performs tool selection and determines how each task is to be executed. It uses the matchmaking technique [Kuokka and Harada, 1995] and brings information providers together with information consumers. This matching process is made possible by information providers actively advertising their capabilities to the facilitator and information consumers sending requests for some service to the same facilitator. The facilitator then identifies the advertisements that are relevant to the requests and creates a communication link between the two parties.

A domain is initially set up by supplying the facilitator with Domain Information Models, Domain tasks and the HTN methods for expanding them, and the generic op-

[1] We restrict ourselves to sequential plans until the problem of reasoning about the currency of resources in the presence of partially ordered plans has been studied further.

erators for these domains.

When agent-encapsulated tools become available to the system, they send planning operators to the facilitator to advertise their capabilities. At present these must be more specialised versions of task operators for some domain; meaning that these advertisements include additional resource information for the inputs and outputs of the tool.

User agent This contains the user's goals and other information pertinent to a particular user. It initiates planning when required and triggers execution by sending `recruit` requests to the facilitator for each step in the plan. In addition, the user agent keeps track of the current plan and the current system state (what operators have been executed and the URCs of the available resources).

5 Application: University Course Administration

In the domain of university course administration, information processing and management tasks include the addition or deletion of students from the class roll, marking student assignments, changing marks when errors in marking are detected, producing statistical summaries of the class marks, etc. Information may be created, deleted or modified at each stage of the process. At the authors' institution, these tasks are performed using a toolkit approach: the course administrator uses a number of different tools to perform the tasks, some being general-purpose tools he is familiar with, and some being specially written for work in this problem domain. This domain is therefore ideal as a testbed for the DALEKS system.

5.1 Example

Using the DALEKS system to support work in the university course administration domain involves initially starting up the facilitator, planning and user agents, plus the tool agents encapsulating the available tools. When the tool agents are started, they send `advertise` messages that contain specifications of their capabilities to the facilitator. These advertisements contain the specialised operators that define the actions the tool can perform, including resource information such as the formats of its inputs and outputs. They are used by the facilitator, at run time, to select a tool agent to invoke.

The user agent initiates planning and plan execution, and keeps a library of plans it can use to process the user's goals. Figure 3 shows an example plan from the university course administration domain. This is a linearly ordered plan with the tree structure depicting its hierarchical development from the initial non-primitive task `Do-marking` (task parameters naming the assessment component to be marked have been omitted). The marking task is decomposed into the generic information processing operations of finding an information resource for the `student` relation and then repeatedly marking batches of student assignments and recording the marks in an information resource for the `assess` relation.

Planning and execution may be interleaved and in this case, as the recursive call to the task `mark-and-update` (underlined) is a non-primitive task, once the user agent has asked the facilitator to execute the actions appearing in the other leaf nodes (causing the run-time variable `!ID_Set` to be instantiated), it must then request the facilitator to recruit a planning agent to further elaborate the plan[2].

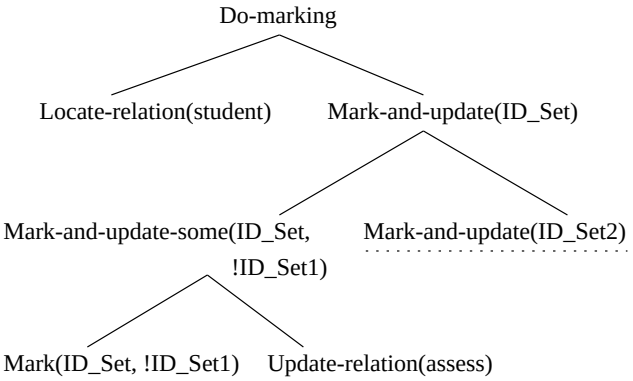

Figure 3: An example plan for the marking task

Another situation where planning is interleaved with execution is when the input of a tool does not match any of the resources that have been produced so far (for example, a mismatch in resource formats). After selecting a tool to use to execute a plan step, the facilitator, using the tool advertisements, will determine if the tool's input requirements are met. It may find that, although a resource exists with the required information content, this resource is not in a format readable by the selected tool. The facilitator then calls the planner to derive a plan to do the translation and format conversion. This plan causes the creation of a new resource, which is added to the list of URCs stored in the user agent.

6 Related Work

There are a number of planners designed to plan for gathering information from large dynamic networks of information sources ([Golden *et al.*, 1994; Knoblock, 1995; Kwok and Weld, 1996; Williamson *et al.*, 1996]).

XII [Golden *et al.*, 1994] is a general-purpose planner to allow for sensing of the world as well as causal actions. Although its actions can change the world, it makes a distinction between information goals and causal goals. Although it

[2] The question of how far the planner should expand the plan each time is a subject for further research: in this case, the presence of the run-time variable `!ID_Set1` and the constraint in the method for `Mark-and-update` (See Figure 2) could be used to infer that expansion of the underlined task be delayed until `ID_Set2` is instantiated.

could probably be applied to information processing tasks, its action language is not designed to describe such domains.

Occam [Kwok and Weld, 1996] is an algorithm to generate plans for efficiently accessing multiple information sources in order to satisfy information gathering queries. Occam models the available information as a relational database schema, but as Occam plans for information gathering from an unchanging world, this information model is regarded as static. The available information resources are modelled by associating information retrieval actions with the relations of the world model that are returned when these actions are executed.

Williamson *et al.* [1996] extend the HTN planning paradigm by explicitly modelling tasks' *provisions* (named interface 'slots' with an attached queue for storing incoming values), *outcomes* (indicating the result status of the task) and its *result* (a value produced by executing the task). Task networks are extended to include links between the results and provisions of tasks, indicating a flow of information. This mechanism is claimed to unify and generalise the methods by which operators can obtain information in traditional planning frameworks: by parameter binding, the passing of information from other operators via the world state, and through the use of run-time variables. Provisions also have a role to play in controlling the execution of plans, with primitive tasks being (re)activated whenever all their required inputs are available.

Our framework takes a different approach to representing and reasoning about information flow between actions. By explicitly modelling the domain's information state and the changes that operators make to it, the intellectual content of information resources can be described in terms of which relation in the information state they correspond to, and at which stage of the information processing process (described by the sequence of actions that have been executed to reach that state). Together with our use of matchmaking, this allows agents to use any available resource that contains the required information — the provider of the information does not need to be hard-wired into the plan.

7 Conclusions

We have described the DALEKS system for information tool interoperation, which separates higher-level planning from lower-level matchmaking activities. This facilitates the development of an information processing domain model that can include information creation and other changes to the information state as operations are conducted. It also enables the interleaving of planning, matchmaking, and execution so that dynamic changes to the environment can be more easily accommodated.

Work on the system is continuing in order to extend the domain model beyond the relational model (such as using concepts and roles, as in SIMS [Knoblock, 1995]) and also to operate with existing information represented in different ontologies.

References

[Ambros-Ingerson and Steel, 1988] J. Ambros-Ingerson and S. Steel. Integrating planning, execution and monitoring. In *Proceedings of the 7th National Conference on Artificial Intelligence (AAAI-88)*, pages 735–740, 1988.

[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*, December 1995.

[Cranefield and Purvis, 1996] S. J. S. Cranefield and M. K. Purvis. An agent-based architecture for software tool coordination. In *Proceedings of the Workshop on Theoretical and Practical Foundations of Intelligent Agents, Pacific Rim International Conference on Artificial Intelligence*, 1996. (to appear in Lecture Notes in Artificial Intellence, Springer, 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 and Ketchpel, 1994] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, July 1994.

[Golden *et al.*, 1994] K. Golden, O. Etzioni, and D. Weld. Omnipotence without omniscience: Efficient sensor management for planning. In *Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94)*, pages 1048–1054, 1994.

[Knoblock, 1995] C. A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, volume 2, pages 1686–1693, 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.

[Kwok and Weld, 1996] C. Kwok and D. Weld. Planning to gather information. In *Proceedings of the 13th National Conference on Artificial Intelligence (AAAI-96)*, 1996.

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

[Williamson *et al.*, 1996] M. Williamson, K. Decker, and K. Sycara. Unified information and control flow in hierarchical task networks. In *Proceedings of the AAAI-96 Workshop on Theories of Planning, Action, and Control*, 1996.