An Adaptive Meeting Scheduling Agent

J. William Murdock, Ashok K. Goel
College of Computing, Georgia Institute of Technology
Atlanta, GA 30332
{murdock,goel}@cc.gatech.edu

Source: Proceedings of the 1st Asia-Pacific Conference on Intelligent Agent Technology (IAT'99), Hong Kong, December 15-17, 1999

An intelligent agent, such as a meeting scheduling system, has a set of constraints which characterizes what that agent can do. However, a dynamic environment may require that a system alter its constraints. If situation-specific feedback is available, a system may be able to adapt by reflecting on its own reasoning processes. Such reflection may be guided not only by explicit representation of the system's constraints but also by explicit representation of the functional role that those constraints play in the reasoning process. We present an operational computer program, SIRRINE2 which uses Task-Method-Knowledge models of a system to reason about traits such as system constraints. We further describe an experiment with SIRRINE2 in the domain of meeting scheduling.

1 Introduction

Meetings schedulers are examples of common software agents. Commercially available meeting schedulers typically address a simple and formal version of the problem. But the social dynamics of scheduling meetings in the real world often defies simple formalization. In addition, the design of commercially available meeting schedulers is static and rigid, and, thus, incapable of evolution. It is potentially desirable for a meeting scheduler to be able to modify itself, e.g., by learning new constraints. A meeting scheduler is likely to have a wide variety of constraints. Consider a meeting scheduler for a group of users who have a fixed constraint that meetings be held during normal business hours (9:00 AM to 5:00 PM on Mondays through Fridays). A system which only schedules meetings during these times will perform correctly in this environment. However, if the needs of the users change (for example, due to a change in the business practices of the organization which uses the meeting scheduler), it may be necessary for the meeting scheduling system to adapt to this change. If these users start wanting to hold meetings outside of normal business hours, it would be very useful for them to be able to provide *feedback* to the system and have it then *evolve* to satisfy these new constraints.

If the relationship between the constraints and the ultimate results produced by the system is represented explicitly (e.g., through a set of equations), it is generally possible absorb situation-specific feedback by logical deduction. If very many associations between constraint values and results are available,

it should be possible to infer appropriate constraint values for a given piece of feedback through inductive learning. However, if the relationship between the constraints and the results is relatively subtle and a large quantity of raw data is not available, more knowledge intensive techniques may be needed. In particular, if an agent has a functional model of its own reasoning process, it may be possible to include constraints within that model, and thus represent not only what the constraint values are but also *what functional role they play* in the agent's reasoning. Under these circumstances, model-based credit assignment may be used to identify which constraints are having what effect on a specific output; the combination of this information with situation-specific feedback can thus enable adjustment of the constraints.

The SIRRINE2 system is an agent architecture for implementing agents with self-knowledge. The language for specifying agents in SIRRINE2 is the Task-Method-Knowledge (TMK) language which provides functional, causal descriptions of intelligent reasoning mechanisms. One aspect of these descriptions can be the explicit representation of agent's constraints and the functional role they play in the agent's behavior.

2 TMK Models

Agents in SIRRINE2 are modeled using the Task-Method-Knowledge (TMK) language. Predecessors of this language have been used in a number of other research projects such as AUTOGNOSTIC [1]. The *TMK model* provided by the user is directly accessible to the evolutionary reasoning mechanism as declarative knowledge about the agent's processing. The work presented here extends the range of applications of TMK by focusing on the addition of new capabilities, rather than, for example, correcting failures in the original design, as in AUTOGNOSTIC.

Processes in TMK are divided into *tasks* and *methods*. A task is a unit of computation which produces a specified result. A description of a task answers the question: *what* does this piece of computation do? A method is a unit of computation which produces a result in a specified manner. A description of a method answers the question: *how* does this piece of computation work? Each task is associated with a set of methods, any of which can potentially accomplish it under certain circumstances. Each method has a set of subtasks which combine to form the operation of the method as a whole. These subtasks, in turn, may have methods which accomplish them, and those methods may have further subtasks, etc. At the bottom level, "primitive tasks" are defined which may not be further decomposed. Descriptions of knowledge in TMK is done through the specification of *domain concepts*, i.e., kinds of knowledge

and *task concepts*, i.e., elements of knowledge. The functional specification of a task refers to the task concepts that the task manipulates and the task concepts, in turn, refer to the domain concepts that their values embody.

TMK represents abstract knowledge of the kinds of constraints that exist in a domain as domain concepts. The connection of a particular set of constraints to a particular agent is then represented by a task concept. The representation of constraint knowledge in the meeting scheduling agent is explored in more detail in Section 3.

3 The Meeting Scheduler

Figure 1 presents the TMK model of the meeting scheduling agent. The top level task of the agent is the task of scheduling a meeting. It has one method which it uses, that of enumerating a set of slots and checking those slots against the schedules. This method sets up subtasks which chose a time slot to consider, and then check to see if that time slot fits the given schedules, and then (if necessary) goes on to the next time slot.

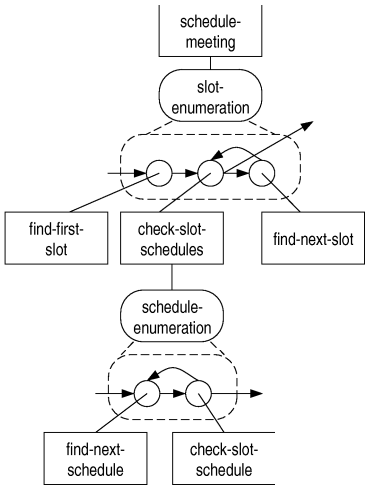

Figure 1: A TMK model of a simple meeting scheduling agent. Rectangular boxes represent tasks; round boxes represent methods. Transition diagrams underneath methods indicate the control imposed by methods on their subtasks.

In addition to the tasks and methods illustrated in Figure 1, the TMK model of the meeting scheduler also contains explicit representations of the knowledge contained by the meeting scheduler. The meeting scheduling agent

contains eight domain concepts, including a time-constraints concept which specifies when meetings can be held, typically Monday through Friday from 9:00 AM to 5:00 PM

In one experiment with this meeting scheduler, we attempted to schedule a 90 minute meeting for which every slot that it considered conflicted with at least one schedule; consequently, the meeting scheduler failed to generate a time for the meeting. Feedback then indicated that the meeting would be held on Tuesdays from 4:30 to 6:00 PM, i.e., the assumption that meetings must be held during standard business hours was violated. At this point, the system was able to do some self-adaptation so that it would have generated this answer in the first place.

There are many possible changes that can be made which would have lead to this result. The meeting scheduler could be changed to always schedule meetings on Tuesdays 4:30 to 6:00 PM. Alternatively, it could be made to schedule meetings on Tuesdays 4:30 to 6:00 PM only when it receives exactly the same set of schedules as it did in the experiment and to simply use its existing mechanisms in all other cases. A reasonable compromise would be to allow meetings to generally be scheduled until 6:00 PM. However, the current model of the meeting scheduling domain does not provide sufficient information to identify a reasonable compromise. Consequently, a simpler change was made in which the Tuesdays 4:30 to 6:00 slot is suggested whenever a 90 minute time slot is requested and no other time is available.

Figure 2 illustrates the revised meeting scheduler. The primitive find-next-slot task is modified to be a non-primitive task with two methods: find-next-slot-base-method simply invokes the existing primitive for finding slots, and find-next-slot-alternate-method always produces the Tuesdays from 4:30 to 6:00 slot. The applicability conditions of the alternate method indicates that it is to be run if and only if a 90 minute time slot is requested under knowledge conditions similar to the ones found here. Similarity, in this situation, is defined by the task concepts for that slot task, i.e., the alternate method is chosen whenever the values of the input task concepts exactly match their values in the earlier execution. The redesign strategy presented here is one example of a way to expand upon a set of constraints: by providing an alternative functional element with different constraints and establishing appropriate conditions for selecting that elements, the overall constraints of the model are expanded.

The key result of this experiment is that learning is enabled by the explicit representation of the constraints, *combined* with the connection between this representation and the functional descriptions of the computational units; the task concepts in the TMK models provide the integration of constraint representation and that representation's functional role. The learning of new

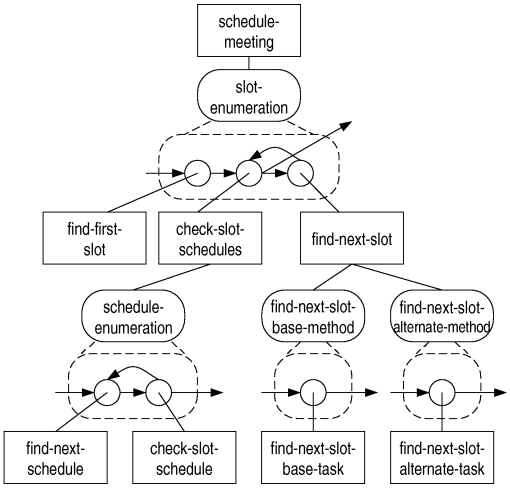

Figure 2: The revised meeting scheduling agent in which the find-next-slot task has been altered.

constraints here takes place with only a single trial; one problem is presented and one piece of feedback is received. Furthermore, it is done without any direct mapping from constraint knowledge to final results; the model simply indicates that the constraints data structure affects the find-first-slot and find-next-slot tasks. Credit assignment over the model and an execution trace is needed to determine how the constraints affected the results during the execution. The modification made is guided by this credit assignment and leads to an enhanced system whose constraints are consistent with the feedback provided by the user.

Acknowledgments

This work has benefited from discussions with our colleagues in the MORALE group, especially Spencer Rugaber. This effort has been sponsored by DARPA under agreement number F30602-96-2-0229.

References

1. Eleni Stroulia and Ashok K. Goel. Redesigning a problem-solver's operators to improve solution quality. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 562–567, San Francisco, August 23–29 1997. Morgan Kaufmann Publishers.