MATE: An experiment while you sleep test bed

C. Mason
AI Research Branch
NASA Ames Research Center
Moffett Field CA 94035
USA

Collaboration among design systems is rapidly becoming a focus of research in automated design (Klein 1992) (Lander and Lesser 1992) (Maher et. al. 1993). The construction of collaborative design systems involves issues relating to the integration of heterogeneous databases and tools, multi-user interfaces, and design documentation, as well as issues of interagent communication and coordination. Here we focus on tools for exploring issues related to communication and coordination during the design, construction, and evaluation of collaborative design systems. The Multi-Agent Test Environment, MATE, is an "experiment while you sleep," test environment that performs the task of network resource allocation, experiment initiation, monitoring and termination, and archival of experimental results. It was developed over the course of three DAI projects, and has proven to be invaluable in the experimental development and evaluation of collaborative problem solving systems.

The extended abstract is organized as follows. We first present some background on MATE - how it evolved, the motivations, and philosophy. Next, a general description of MATE is presented. The remaining sections present a brief description of the three basic steps to running experiments with MATE: experiment configuration, network resource allocation, and experiment execution and recording.

1. BACKGROUND

MATE has evolved over the construction of three DAI systems, NETSEA II - a collaborative interpretation system for global seismic monitoring for a Comprehensive Test Ban Treaty, EXPLORE-TEAM - a network of hybrid-architecture T-1 robots for planetary exploration, and most currently, GNAT - a collaborative scheduling system for Globally Networked Automatic Telescopes. One generalization that can be made from the experience with these three widely varying, collaborative systems is that experimentation with communication and coordination strategies is critical to the correct and effective functioning of collaborative problem solving systems. The appropriateness of a particular strategy for negotiation or cooperation behavior varies according to the structure of the application, and in general, the usefulness of a particular strategy in an application domain may be situation dependent. In short, the construction of collaborative problem solving

* This research was supported by a National Research Council Associateship through the National Academy of Sciences.

systems requires a great deal of experimentation to sort out exactly what type of communication and coordination strategies are most effective, and when, or even if they work at all[1]. MATE was created to address this side of collaborative system design.

2. MATE - The Multi-Agent Test Environment

The Multi-Agent Test Environment, MATE, is a collection of experiment management tools for assisting in the design, testing, and evaluation of collaborative distributed problem solving systems. MATE provides tools for automatic experiment execution, monitoring, and archiving of results, thus relieving much of the work involved in monitoring and organizing the execution of a collaborative problem solving system. Using MATE, the program designer can run a series of experimental collaborative problem solving scenarios without human intervention by finding a set of eligible machines and executing pre configured experimental trial descriptors. Alternatively, an individual experiment may be run manually, allowing for interaction between the program designer and one or more single agent debugging environments. MATE thus provides a platform for the collaborative program designer to investigate alternative representation and reasoning strategies, protocols for communication and negotiation, or other coordination issues.

The role of MATE in solving a particular problem solving instance can be described by envisioning the collaborative problem solving software in terms of three layers, the agent, the communications or network support layer, and MATE. An "agent" is composed of an interactive process, such as a lisp interpreter, a problem solving module, such as a KBS, procedure, or other representation, the data set(s), and a communication interface.

The communications layer provides four services to the agents: 1) point-to-point, multi-cast, and broadcast message delivery, 2) transparent translation between symbolic agent name and physical workstation address, 3) experiment start synchronization, and 4) experiment termination notification.

MATE's primary control tool is "AUTO-MATE," which, without human intervention, monitors each experimental trial for network, machine, or software failure, and restarts the trial if necessary and possible. The results of the experiment may be reviewed at the time it is run, or much later, as each experimental trial is archived for later analysis. These logs are used not only for gathering performance information, but in debugging the collaborative distributed problem solver as well.

There are three phases to running experiments: 1) Experiment configuration, 2) network resource allocation, and 3) experiment execution and recording. Each of these phases is described below.

2.1 Experiment Configuration

Each experimental scenario must be configured according to the number of agents (and therefore the number of machines to be used), communication or negotiation policy, problem solving module, and data sets. The configuration of experiments is done manually, by specifying a collection of invocation parameters to MATE tools. For example, in our distributed scheduling system, negotiation strategies may be selected: 1) direct - agent to agent, 2) indirect - where a third agent monitors the progress of the two

[1] In this sense, programming collaborative problem solving systems is no different than any other type of programming activity.

agents involved in the negotiation, and 3) arbitration - where each of the two agents communicates with an arbitration agent, and abides by the decision of the arbitration agent. We also have parameters for selecting among inference engines in our problem solving module, which is represented by a knowledge based system: 1) single - for debugging a single agent with no communication, 2) nontms - for debugging multiple agents without use of a TMS system and 3) full-engine - for running multiple agents with the use of a TMS system.

2.2 Network Resource Allocation

In the network resource allocation phase workstations in the local area network are selected for the experiments by the program, "NOSE." NOSE is called either from AUTO-MATE, or run manually. NOSE consults a list of known workstations on the local area network, and selects a machine for each agent. NOSE tests each workstation to determine if : 1) it has the proper file system and account access to run the experiment, 2) it has the memory resources to run the agent, 3) it has sufficient CPU power, and 4) the system is not heavily loaded by other users (the load threshold being adjustable). NOSE produces a list of suitable workstations once per experiment, or whenever a negative change in usage patterns of the workstations is detected during the runs. This was found to be quite sufficient, and much less expensive than testing the availability of each workstation more frequently as would occur in a WORM program, for example.

2.3 Experiment Execution and Recording

Once the experiment configuration is in place, and a list of capable machines is created, the collaborative problem solver may be initiated through the AUTO-MATE control and monitoring program (implemented as a UNIX csh program), or manually via menu selections. Typically, manual invocation of the system is performed while debugging new problem solver modules, for example, as the experimenter is given finer control over execution and can interact with the single-agent debugging environment, running single-step, watching problem solver actions, manipulations of data base objects, and so on.

The main purpose of AUTO-MATE is to execute a collaborative problem solving system over a variety of problem solving scenarios, making sure everything from agent creation to logging of results goes smoothly. This includes the following activities: 1) ensuring all agents specified for the problem scenario are up and running - restarting agents whenever error conditions allow, 2) detecting error conditions, 3) ensuring agent logs are written properly, 4) verifying that no agent terminates prematurely, and 5) detecting when an experiment is over.

For each experimental trial, AUTO-MATE begins by releasing a number of remote shell commands that, after killing off any other stray agent processes, create agents on each of n workstations, copying the proper load files, data sets, communication rules, and so on (specified by invokation parameters) for this particular trial. AUTO-MATE then runs the communications server, watching to ensure it starts successfully. As each agent process and server is started, a log file is created. This part of experiment execution is subject to several types of failures, so AUTO-MATE closely monitors the log files for the agents. If necessary, AUTO-MATE will try to restart one or more agents that did not start, and may re-run NOSE looking for other available machines.

During problem solving, agents write many intermediate results and run-time statistics (such as size of working memory, number of messages sent or received and so on) to the log file as each step of the processing proceeds. A recognizable message is written to the log file when problem solving has been exhausted, and agents await further messages. When all agents are finished processing and are waiting for further messages, the AUTO-MATE monitoring process detects the trial is complete (by observing the distinguishing end message at the end of each log file for a fixed period of time) and terminates the communication server and agents. The log files for each of the agents and the server are then copied into the proper place in the directory structure specified for that experimental trial.

2.4 IMPLEMENTATION

MATE tools use several UNIX-based workstations networked together via ethernet in a large local-area network. All of the MATE software is programmed in C and UNIX shell programs. A complete listing of MATE functions may be found in (Mason 1992).

3. SUMMARY

This abstract addresses the issues of agent communication and cooperation from an experimentalist perspective, and emphasizes the need for tools to aid in the design, construction, and evaluation of collaborative problem solving systems. The Multi-Agent Test Environment, MATE, provides a no-hands experimentation testbed, freeing the collaborative program designer from the tedious chores of resource allocation, archival of results, and experiment initiation, monitoring, and termination.

REFERENCES

Klein, Mark (1992). Supporting Conflict Management in Cooperative Design Teams, *AAAI-92 Workshop Notes on Cooperation Among Heterogeneous Intelligent Systems*, San Jose.

Landers, S. and Lesser, V. (1992). Negotiated Search: Cooperative Search Among Heterogeneous Expert Agents, *AAAI-92 Workshop Notes on Cooperation Among Heterogeneous Intelligent Systems*, San Jose.

Maher, M.,. Gero, J., and Saad, M.(1993). *CAAD Futures '93*, Elsevier, New York, Eds. U. Flemming and S. Van Wyle.

Mason, C. (1992). MATE - A Multi-Agent Test Environment, *Technical Report FIA-93-09*, NASA Ames Research Center, AI Research Branch.