NASA-TM-108114

135326

p/12

MATE: The Multi-Agent Test Environment

Cindy L. Mason
Artificial Intelligence Research Branch
MS 269-2
NASA Ames Research Center
Moffett Field, CA 94035-1000

(NASA-TM-108114) MATE: THE MULTI-AGENT TEST ENVIRONMENT (NASA) 12 p

N93-15292

Unclas

G3/63 0135326

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA-92-25

June, 1992

MATE: The Multi-Agent Test Environment

Cindy L. Mason

AI Research Branch, Mail Stop: 269-2
NASA Ames Research Center
Moffett Field, CA 94035 U.S.A.

mason@chaos.arc.nasa.gov

Abstract

In this report we present the Multi-Agent Test Environment, MATE. MATE is a collection of experiment management tools for assisting in the design, testing, and evaluation of distributed problem-solvers. It provides the experimenter with an automated tool for executing and monitoring experiments choosing among rule bases, number of agents, communication strategies, and inference engines. Using MATE the experimenter can run a series of distributed problem-solving experiments without human intervention.

MATE: The Multi-Agent Test Environment

Cindy L. Mason

1 Introduction

The MATE (Multi-Agent Test Environment) testbed is a collection of experiment management tools for the design, testing, and evaluation of distributed problem-solving experiments. Using MATE the experimenter can run a series of distributed problem-solving experiments without human intervention. MATE tools use several UNIX-based workstations networked together via ethernet in a large local-area network. Each workstation runs a single agent that communicates via the local-area network, except for one workstation that serves as the MATE control center and communication server (see figure 1). Using MATE, the experimenter creates a problem-solving setting by choosing the rule bases, number of agents, inference engine, and communication strategy. Thus, MATE provides a platform for an experimenter to investigate alternative reasoning, communication and control strategies.

The role of MATE in solving a particular problem-solving instance is depicted in figure 2. The multi-agent software can be viewed in terms of 3 layers, the agent,

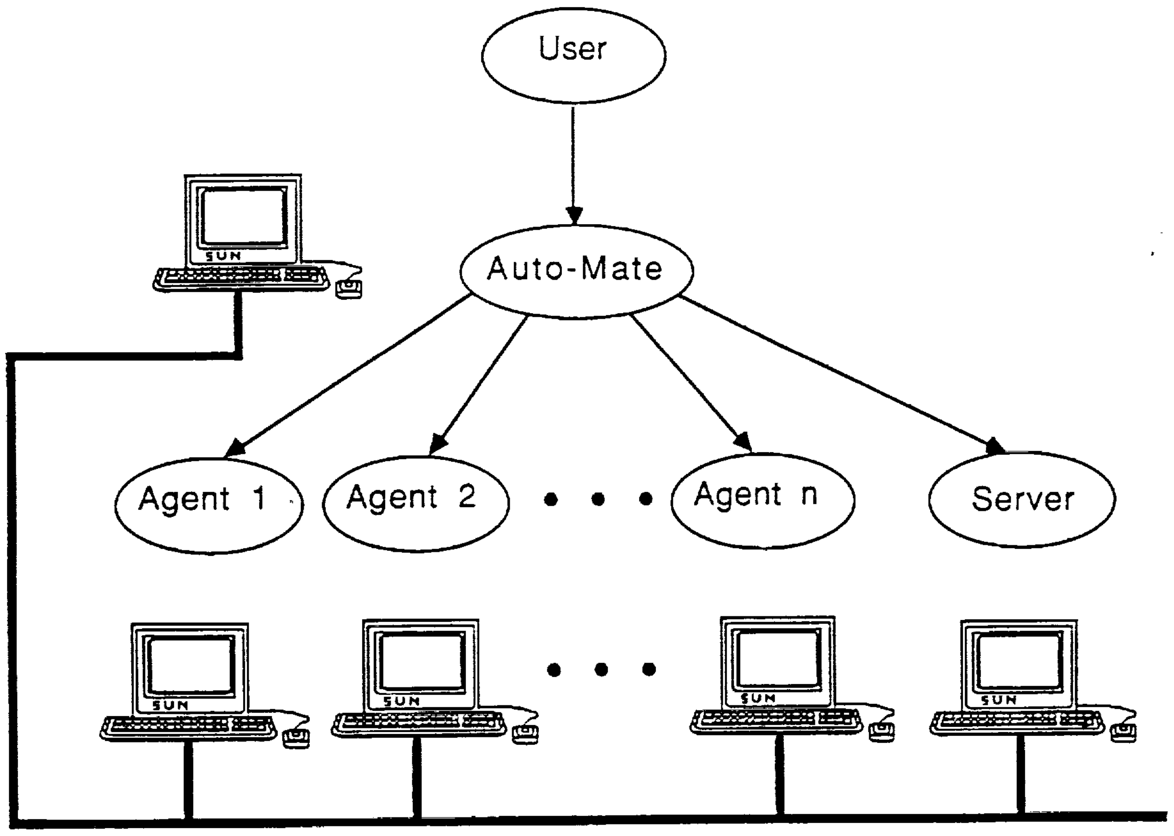


Figure 1: The MATE testbed architecture consists of the control and monitoring program, AUTO-MATE, the communications server, and one or more agents, interconnected by an ethernet.

the communications layer, and MATE. An "agent" is composed of a lisp interpreter, inference engine, rule base, data set, and a communications interface.

The communication 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.

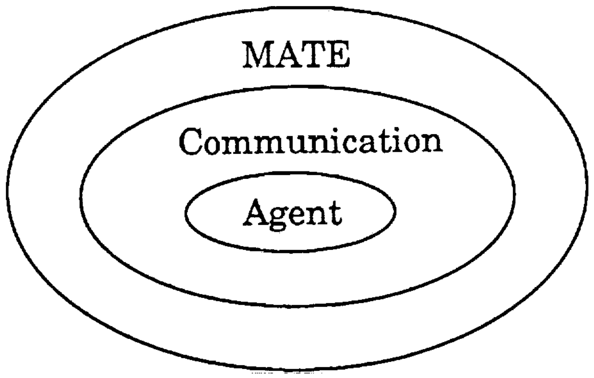
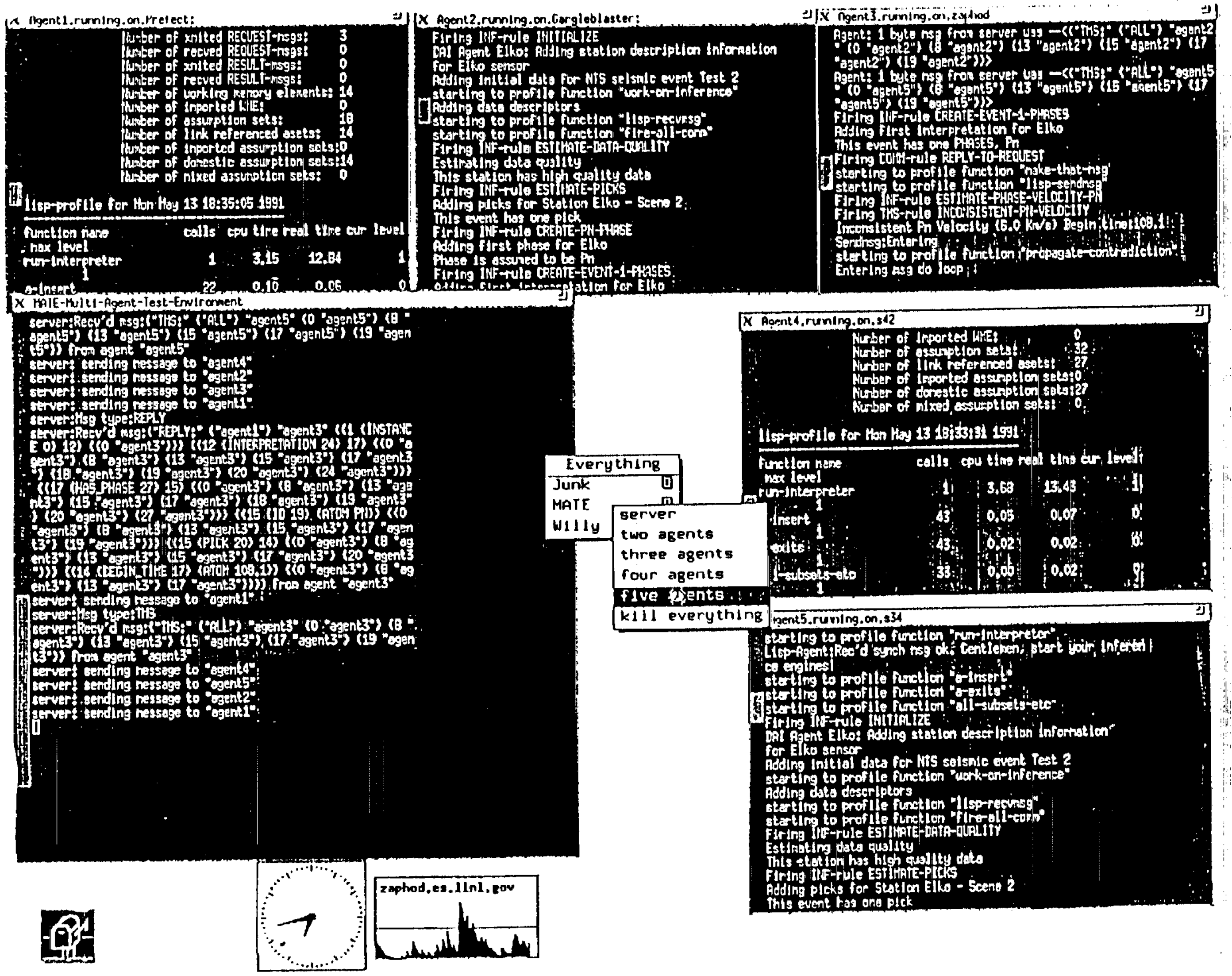


Figure 2: An agent consists of the individual Agent layer, a Communications layer, and the MATE control layer.

2 MATE Tools

MATE tools automate much of the work involved in monitoring and organizing the execution of the distributed problem solver. MATE's primary control tool is "AUTO-MATE" which finds a set of eligible machines and executes preconfigured experimental trials. Alternatively, experiments may be run manually via the program "LAUNCH-MATE", which may be selected from a menu in the X-windows user interface (see figure 3), or simply typing the program name after the UNIX command prompt.

Using AUTO-MATE, each experimental trial is automatically monitored for network, machine, or software failure and is restarted if necessary and possible. The results may be reviewed at the time the experiments are run or much later, as each trial log is automatically archived for later analysis. These logs are used not only for gathering performance information but in debugging the distributed problem solver as well.



```
               Number of xmited REQUEST-msgs:      3
               Number of recved REQUEST-msgs:      0
               Number of xmited RESULT-msgs:       0
               Number of recved RESULT-msgs:       0
               Number of working memory elements: 14
               Number of imported WME:             0
               Number of assumption sets:         18
               Number of link referenced asets:   14
               Number of imported assumption sets:0
               Number of domestic assumption sets:14
               Number of mixed assumption sets:    0

lisp-profile for Mon May 13 18:35:05 1991
function name            calls  cpu time real time cur level
 max level
run-interpreter              1     3.15     12.84         1
          1
a-insert                    22     0.10      0.06         0
```

```
Firing INF-rule INITIALIZE
DAI Agent Elko: Adding station description information
for Elko sensor
Adding initial data for NTS seismic event Test 2
starting to profile function "work-on-inference"
Adding data descriptors
starting to profile function "lisp-recvmsg"
starting to profile function "fire-all-comm"
Firing INF-rule ESTIMATE-DATA-QUALITY
Estimating data quality
This station has high quality data
Firing INF-rule ESTIMATE-PICKS
Adding picks for Station Elko - Scene 2
This event has one pick
Firing INF-rule CREATE-PN-PHASE
Adding first phase for Elko
Phase is assumed to be Pn
Firing INF-rule CREATE-EVENT-1-PHASES
Adding first interpretation for Elko
```

```
Agent: 1 byte msg from server was —(("THS:" ("ALL") "agent2
" (0 "agent2") (8 "agent2") (13 "agent2") (15 "agent2") (17
"agent2") (19 "agent2")))
Agent: 1 byte msg from server was —(("THS:" ("ALL") "agent5
" (0 "agent5") (8 "agent5") (13 "agent5") (15 "agent5") (17
"agent5") (19 "agent5")))
Firing INF-rule CREATE-EVENT-1-PHASES
Adding first interpretation for Elko
This event has one PHASES, Pn
Firing COMM-rule REPLY-TO-REQUEST
starting to profile function "make-that-msg"
starting to profile function "lisp-sendmsg"
Firing INF-rule ESTIMATE-PHASE-VELOCITY-PN
Firing THS-rule INCONSISTENT-PN-VELOCITY
Inconsistent Pn Velocity (6.0 Km/s) Begin time:108.1
Sendmsg:Entering
starting to profile function "propagate-contradiction"
Entering msg do loop
```

```
server:Recv'd msg:("THS:" ("ALL") "agent5" (0 "agent5") (8 "
agent5") (13 "agent5") (15 "agent5") (17 "agent5") (19 "agen
t5")) from agent "agent5"
server: sending message to "agent4"
server: sending message to "agent2"
server: sending message to "agent3"
server: sending message to "agent1"
server:Msg type:REPLY
server:Recv'd msg:("REPLY:" ("agent1") "agent3" ((1 (INSTANC
E 0) 12) ((0 "agent3"))) ((12 (INTERPRETATION 24) 17) ((0 "a
gent3") (8 "agent3") (13 "agent3") (15 "agent3") (17 "agent3
") (18 "agent3") (19 "agent3") (20 "agent3") (24 "agent3")))
 ((17 (HAS_PHASE 27) 15) ((0 "agent3") (8 "agent3") (13 "age
nt3") (15 "agent3") (17 "agent3") (18 "agent3") (19 "agent3"
) (20 "agent3") (27 "agent3"))) ((15 (ID 19) (ATOM PN)) ((0
"agent3") (8 "agent3") (13 "agent3") (15 "agent3") (17 "agen
t3") (19 "agent3"))) ((15 (PICK 20) 14) ((0 "agent3") (8 "ag
ent3") (13 "agent3") (15 "agent3") (17 "agent3") (20 "agent3
"))) ((14 (BEGIN_TIME 17) (ATOM 108.1)) ((0 "agent3") (8 "ag
ent3") (13 "agent3") (17 "agent3")))) from agent "agent3"
server: sending message to "agent1"
server:Msg type:THS
server:Recv'd msg:("THS:" ("ALL") "agent3" (0 "agent3") (8 "
agent3") (13 "agent3") (15 "agent3") (17 "agent3") (19 "agen
t3")) from agent "agent3"
server: sending message to "agent4"
server: sending message to "agent5"
server: sending message to "agent2"
server: sending message to "agent1"
```

```
               Number of imported WME:             0
               Number of assumption sets:         32
               Number of link referenced asets:   27
               Number of imported assumption sets:0
               Number of domestic assumption sets:27
               Number of mixed assumption sets:    0

lisp-profile for Mon May 13 18:33:31 1991
function name            calls  cpu time real time cur level
 max level
run-interpreter              1     3.68     13.43         1
          1
insert                      43     0.05      0.07         0
          1
exits                       43     0.02      0.02         0
          1
l-subsets-etc               33     0.00      0.02         0
          1
```

```
starting to profile function "run-interpreter"
Lisp-Agent:Rec'd synch msg ok: Gentlemen, start your inferen
ce engines!
starting to profile function "a-insert"
starting to profile function "a-exits"
starting to profile function "all-subsets-etc"
Firing INF-rule INITIALIZE
DAI Agent Elko: Adding station description information"
for Elko sensor
Adding initial data for NTS seismic event Test 2
starting to profile function "work-on-inference"
Adding data descriptors
starting to profile function "lisp-recvmsg"
starting to profile function "fire-all-comm"
Firing INF-rule ESTIMATE-DATA-QUALITY
Estimating data quality
This station has high quality data
Firing INF-rule ESTIMATE-PICKS
Adding picks for Station Elko - Scene 2
This event has one pick
```

Figure 3: The Manual MATE User Interface

ORIGINAL PAGE IS
OF POOR QUALITY

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 Experimental Configuration

Each experiment must be configured according to number of agents (and therefore the number of machines to be used), communication policy, inference engine, rule bases and data sets. The configuration of experiments is done manually, by specifying a collection of invocation parameters to MATE tools. Currently, communication policies may be selected as broadcast, undirected-request, and directed-request. Three inference engines may be selected: 1) **single** - for debugging a single agent with no communication 2) **non-tms** - for debugging multiple agents without truth maintenance systems 3) **full-engine** - for running multiple agents with truth maintenance systems. Any number of rule bases and data sets may be created and specified for an experimental trial. In addition, new inference engines and communication policies may be added to the system by simply building the new software and appending the list of engine types (or communication policies) in the appropriate MATE tools.

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 it: 1)has the proper file system and account access to run the experiments,[1] 2) has the memory resources to run the lisp interpreter, 3) has sufficient CPU power and 4) the system is not heavily loaded by other users (the load threshold is adjustable.) NOSE produces a list of suitable workstations once per experimental session 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 for workstation availability before each trial.

2.3 Experiment Execution and Recording

Once the experiment configuration is in place, and a list of capable machines is created, the distributed problem solver may be initiated through the AUTO-MATE control and monitoring program (implemented as a UNIX csh program), or manually via menu selections (see figure 5.3). Typically, manual invocation of the system is

[1]In this implementation we use NFS, the UNIX-based network file system, for setting up, controlling, and recording the experiments. Other implementation choices are possible and will not affect the experimental results. The agents communicate between themselves via the communication server which is independent of NFS.

performed while debugging new rule sets, inference engines, etc., as the experimenter is given finer control over execution and can interact with the lisp interpreter, run single step, and watch rule tracings and assumptions data base creation, etc.

The main purpose of AUTO-MATE is to execute a problem-solving system over a variety of problem scenarios, making sure everything goes smoothly. This includes:

- ensuring agent logs are written properly
- ensuring that all agents are up and running - restarting agents whenever error conditions allow
- verifying that no agent terminates prematurely
- detecting when an experiment is over
- detecting error conditions

For each experimental trial, AUTO-MATE begins by copying the proper inference engine initialization file and rule-sets (specified by invocation parameters) for this particular trial into place. AUTO-MATE then runs the communication server, watching to ensure it starts successfully[2]. Should the server have any errors, AUTO-MATE will persistently re-start the server until it runs successfully. At this point each agent is then initiated on the remote workstations using the UNIX remote shell execution command, rsh. As a result of the rsh, a UNIX csh script is executed on each machine that, after killing off any other agents found on the machine, will start

[2]The communications server is needed for instrumentating the experiment, and will not be necessary in a deployed system. However, agents must either address the problem of how to synchronize problem solving, or have the ability to maintain results on multiple problem instances.

up common lisp, thus loading in and executing the agent process. This part of the experiment system is subject to several types of failures (e.g. a machine has crashed, the network went down, there is not enough memory available to start lisp) so AUTO-MATE closely monitors 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.

The UNIX csh script invoked on each workstation starts the process to build and execute an agent. The remote shell script first outputs a recognizable welcome message to the log file. This informs the AUTO-MATE monitoring process that agent initialization has begun successfully. After the message is output, the remote script invokes the lisp interpreter and reads the initialization file, loading the inference engine and rule and data-set, and begins execution of the remote agent. At this point the lisp agent makes a connection with the communication server and waits for a synchronization message to start processing inferences. This synchronization message signals that all agents are ready to begin. During problem solving, agents write many intermediate results and runtime statistics (such as size of working memory, number of messages sent or received, CPU times, and so on) to the log file as each step of the processing proceeds. When the agent can make no further inferences with its current data base, it enters a loop and waits for any messages that could trigger further inferences. A recognizable message is written to the log file when this loop is entered.

Since the communication server must be prepared to process broadcast messages, and the names and addresses of the agents are not known to the server until a connection is made, all agents are required to connect with the server before any messages are delivered. The communications server establishes port connections for agents, using streams, and waits for a connection message from each agent. As connections are made, the server builds up its name translations table. Once all connections are made, the server sends a "Gentlemen start your inference engines" synchronization message to all agents. This synchronization procedure prevents fast agents from queueing up large numbers of incoming messages in the server before they can be delivered to the agents. Without this synchronization, a single slow agent will delay the delivery of messages to all agents.

When all agents are finished processing and are waiting for further messages, the AUTO-MATE monitoring process detects that the trial is complete (by recognizing that this pattern is at the end of each log file for a fixed time period) and terminates the communication server and the agents. The log files for each of the agents and the communication server are copied into the proper place in the directory structure for that experimental trial, and the next trial is started.

3 Summary

The MATE (Multi-Agent Test Environment) testbed is a collection of experiment management tools for the design, testing, and evaluation of distributed problem-solving experiments. These tools use several UNIX-based workstations networked together via ethernet in a large local-area network. MATE can be used to run distributed experiments automatically and monitors experiment execution and completion.