**NOMADS Software Demonstration**

Niranjan Suri and Paul Groth  
Institute for Human and Machine Cognition  
University of West Florida  
{nsuri,pgroth}@ai.uwf.edu

**Introduction**

NOMADS is a Java-based mobile agent system that provides strong mobility and fine-grained resource control. The NOMADS system is built on top of the Aroma Virtual Machine (VM), a Java-compatible VM that has been designed and developed specifically to support two key capabilities: capturing the execution state of a Java process and controlling the resources used by a Java process. This demonstration consists of three different scenarios to showcase the capabilities of the NOMADS system.

The NOMADS system consists of several components as shown in the figure below. The Oasis agent execution environment encapsulates the Aroma VM and handles the execution of agents. The Oasis console and administration program is the front-end to the Oasis execution environment and allows users to configure the execution environment and to interact with agents.

**Resource Control**

The resource control scenario demonstrates the ability of NOMADS to control the resources consumed by agents running on a host. NOMADS provides dynamic and fine-grained resource control limits for both disk and network resources. These limits include the rate (KB/sec) at which disk and network access is allowed, the total number of bytes that are allowed to be read and written to the disk and network, and the disk space used. These limits may be established through policy files and also dynamically changed on a per agent basis. The resource control
scenario will demonstrate the Oasis Console and Administration program being used to
dynamically change the network write rate and the disk write rate for different agents.

**Scram**

The scram scenario demonstrates the ability of NOMADS to forcibly move agents from one host
to another. The Aroma VM allows the execution state of a running process to be captured,
moved to a new system, and restarted using the captured state information. This state capture
mechanism is used to provide both transparent mobility and forced mobility. Transparent
mobility is the ability for an agent to call go() anywhere in its code and have its execution
resume immediately after the go() method on the new host. Forced mobility is the ability of the
system to move an agent from one host to another and have the agent continue its execution
exactly where it was stopped on the first host. Depending on the circumstances, the agent need
not even be aware of the fact that they have been moved.

The forced migration in NOMADS transparently supports multiple, concurrently running
threads. In addition, since the NOMADS agent system is based on Java and Java byte-codes are
platform independent it is possible to checkpoint and therefore scram an agent in a machine
independent way. Currently, NOMADS has been ported to Windows NT (on x86), Solaris (on
SPARC), and Linux (on x86). This allows for an agent to be scrammed to a system running on any
of these environments.

**WYA Distributed System**

The WYA (While You’re Away) scenario demonstrates the application of NOMADS towards
distributed systems. The WYA system’s goal is to take advantage of the resources in idle
workstations. Currently there is a central server responsible for distributing processes over the
network of idle computers. Computations are stored in a queue and are distributed one at a time in
the following fashion. A computer that is part of the WYA system runs a screen saver and an Oasis
service in the background. When the screen saver turns on it contacts the server and informs the
server that the machine can receive computations to process. The server sends the Oasis on the
computer a computation and the computer begins processing it. When the computer stops being
idle the screen saver communicates with the Oasis and all computations are sent back to the server
where the state is stored until another idle computer requests a computation. Since the Aroma VM
can capture the execution state, the computations are restarted exactly at the same point where they
stopped execution. A computation will be processed by the distributed system until it is complete
after which it returns to the originating host.