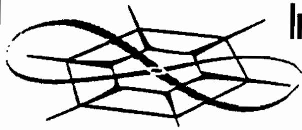
The University of Kansas



Technical Report

Capturing Network/Host State in MAGIC-II

Yulia Wijata
Douglas Niehaus

ITTC-FY98-TR-12161-04

February 1999

Project Sponsor:
Information Technology Office
of the
Defense Advanced Research Projects Agency

Copyright @ 1999:
The University of Kansas Center for Research, Inc.
2291 Irving Hill Road, Lawrence, KS 66045-2969
All rights reserved.

Capturing Network/Host State in MAGIC-II

Yulia Wijata
MAGIC-II Project

Table of Contents

	Page #
1. Overview	1
1.1 Motivation	1
1.2 Objectives	2
2. Design Criteria	2
3. Approach	2
3.1 Using Netspec to Monitor a Wide Area Network	3
3.2 JATLite Package	3
4. Functional Overview	4
5. Implementation of NSAgent	6
5.1 NSAgent Architecture	6
5.2 Knowledge Representation	7
5.2.1 Resources	8
5.2.2 Database	9
5.3 Monitoring Task	9
5.4 KQML Messages	10
6. Implementation of VisAgent	10
6.1 GenMap Package	11
6.2 Visualization Layers	11
6.3 Visual Element Mapping	12
6.4 KQML Messages	12
7. Demonstration Overview	13
7.1 NSAgent Configuration	13
7.1.1 Monitoring Link Quality Between DPSS Client and Servers	14
7.1.2 Monitoring Connectivity in the Network	14

7.1.3 Monitoring Transfer Capacity of the Network 15
7.1.4 Monitoring Network element Status 15
7.2 VisAgent Configuration 15
8. Conclusion and Future Work 20

Appendix A. KQML messages Implemented by NSAgent 23
Appendix B. KQML Messages Implemented by VisAgent 26
Appendix C. Mapping Configuraiton File 28
References 29

1 Overview

This technical report will describe the software architecture and implementation of the monitoring agents developed for the MAGIC-II project [1]. The design and capabilities of the agents will be discussed. This report also documents the configuration of the monitoring agents demonstrated at the MAGIC-II quarterly meeting at University of Kansas on July 14, 1998.

1.1 Motivation

Recent development in the area of high speed networking stimulates the development of large-scale distributed applications. Currently, these applications rely on the best-effort service offered by the network and sometimes suffer low performance when the underlying network behaves unexpectedly because most applications are generally oblivious to variation in network conditions. A network-aware application attempts to alleviate this problem by capturing the state of the network and using this information to adapt to the changing conditions of the network.

To support such adaptation, a network-aware application needs to maintain a view of the network state which is generally dynamic, transient and sometimes tightly coupled with the semantics of the application. An application's view of the network may comprise its topology, availability of resources, and quality or performance of the network elements.

Numerous efforts have been devoted to monitor and probe the network for the purpose of network management or performance evaluation. However, very few are targeted to help applications to make intelligent decisions about utilizing the resources in the network given the extra knowledge about the network. There is yet a need for a support layer between the application and the underlying network through which an application can express its requirements on the characteristics of the network and maintain its view of the network state.

Keeping track of the relevant aspect of the network state is a very challenging task because it deals with a vast amount of information from a large number of network elements which may span multiple administrative boundaries. Each piece of information needs to be collected using an appropriate measurement methodology and should be organized systematically to ensure timely retrieval and meaningful interpretation. To address these issues, we approach this problem by deploying a collection of software agents to provide integrated control of monitoring elements and collected information.

In the context of this report, an agent is defined as an autonomous entity whose responsibility is to *automate* one or more of the following tasks:

- Continuous monitoring of application components and network characteristics
- Creation and control of network testing and measurement
- Collection and storage of performance data
- Correlation and presentation of performance data to application and/or user

In particular, we use KQML[1][2] based agents which allows one to "wrap" legacy tools – in this case, network monitoring and measurement tools – with software and enable them to communicate via a common agent protocol such as KQML. With this approach, the complex task of monitoring a large, distributed system can be decomposed structurally into some domain-specific tasks while maintaining the common goals.

In the MAGIC-II context, the agent-based monitoring system can be used to dynamically configure the Distributed Parallel Storage System (DPSS) [2] and at the same time, monitor the *health* of the MAGIC-II network. In order to understand the interaction between the distributed application and the underlying network, we also need a tool which can provide graphical representation of the state of the distributed system. In a sense, the visualization tool can provide a front end interface to the information collected and maintained by the agents.

1.2 Objectives

The objectives of the work described in this report is twofold:

- To develop a monitoring agent which collects and maintains data about network state by doing testing and measurement on network elements
- To develop a tool that provides geographical display of application and network components and measurement data in a wide-are distributed application and network.

2 Design Criteria

The following design guidelines have been adopted to achieve the design objectives:

- Modularity:
  The software agent framework must promote modular design which clearly separate policy from the mechanism in measurement.
- Portability:
  Since a distributed system most likely comprises heterogeneous components and systems, the agents should be easily portable to different architectures.
- Distributed:
  The system must be capable of monitoring network elements in more than one administrative domain.
- Extensibility:
  The capabilities of the agents in the system should be easily extended to support new types of measurement or testing.

3 Approach

This section describes the two major tools used in the development of the monitoring system: NetSpec, a distributed network performance evaluation tool from University of Kansas, and JATLite, a Java Agent Toolkit package from Stanford University.

[1] Knowledge and Query Manipulation Language

3.1 Using NetSpec to Monitor a Wide Area Network

NetSpec[3] will be used as the main control entity of network testing and measurement because of its capabilities to perform distributed network testing in an integrated and extensible manner. This allows the system to be distributed and extensible at the same time. Moreover, since NetSpec has been ported to several major platforms, portability is not an issue. Basically, NetSpec framework permits two types of daemons/probes to collect data from the network elements:

- Test daemon generates traffics with different types of characteristics and measures the achievable throughput.
- Custom measurement daemon performs specific measurement on the network element.

Figure 1 shows the general architecture of the NetSpec framework. NetSpec controller distributes measurement tasks to several points inside the network based on the testing topology described in the script. In response, the measurement daemons perform the testing or data collection and produce performance reports.

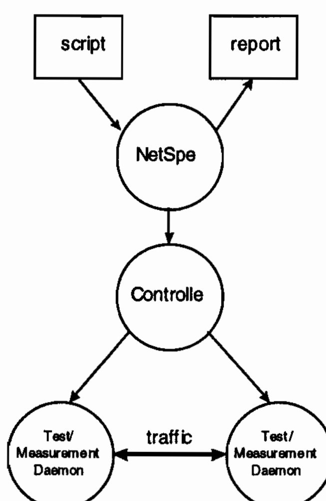


Figure 1 NetSpec Architecture

The role of the Java agent is to create and schedule NetSpec experiments. This can easily be done by specifying the test parameters in the NetSpec script and passing it to the NetSpec interface. The Java agent also needs to collect and organize the performance data reports generated by the test/measurement daemons.

3.2 JATLite Package

JATLite (Java Agent Template, Lite) [4] is package of programs written in the Java [5] language that allows users to quickly create new software agents that communicate robustly over the Internet. JATLite provides a basic infrastructure for agents' communication based upon TCP/IP and KQML messages. The use of Java language allows the agents to be run on heterogeneous platforms and thus, ensures portability. Its modular construction consists of a hierarchy of increasingly specialized layers which may

be customized to fit the specific requirements of a given system. Figure 2 shows how the hierarchy of layers are organized in JATLite.

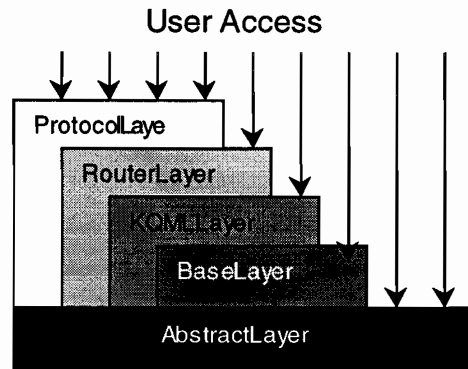


Figure 2 JATLite Layers

The *Abstract Layer* provides the collection of abstract classes necessary for JATLite implementation. The *Base Layer* is built on top of the abstract layer and provides basic communication based on TCP/IP. The *KQML Layer* provides storage and parsing routines for KQML messages. The *Router Layer* provides name registration and message routing and queuing for agents. Finally, the *Protocol Layer* supports diverse standard internet services such as SMPT, FTP or HTTP.

One important concept in JATLite framework is the *Agent Message Router (AMR)* (also referred to as a *router*). It provides name registration and message routing or queuing for agents. In this scheme, agents can operate in disconnected mode and still receive the messages addressed for them. Another advantage is that the existence of an agent is transparent to the other agents in the system. An agent can send a message to another agent in the system by indicating the registered name of that agent in the destination field of the message and then sending the message to the router. The router then will forward the message to its intended recipient as long as it has registered itself with the router.

4 Functional Overview

This section provides the high level architecture of the monitoring system developed in MAGIC-II testbed. In particular, it will define the different types of monitoring agents exist in the system and how they relate to each other. The two main organizations involved in developing the distributed monitoring system are KU and LBNL. KU's contribution mainly involves monitoring at the network level while LBNL is particularly interested in monitoring at the application level. Both efforts are aimed at dynamic reconfiguration of the DPSS and also performance tuning and optimization of the distributed application.

Figure 3 shows the functional overview of the components existing in the system. The MAGIC-II testbed cloud represents the wide-area ATM network and also the distributed application being monitored by the agents.

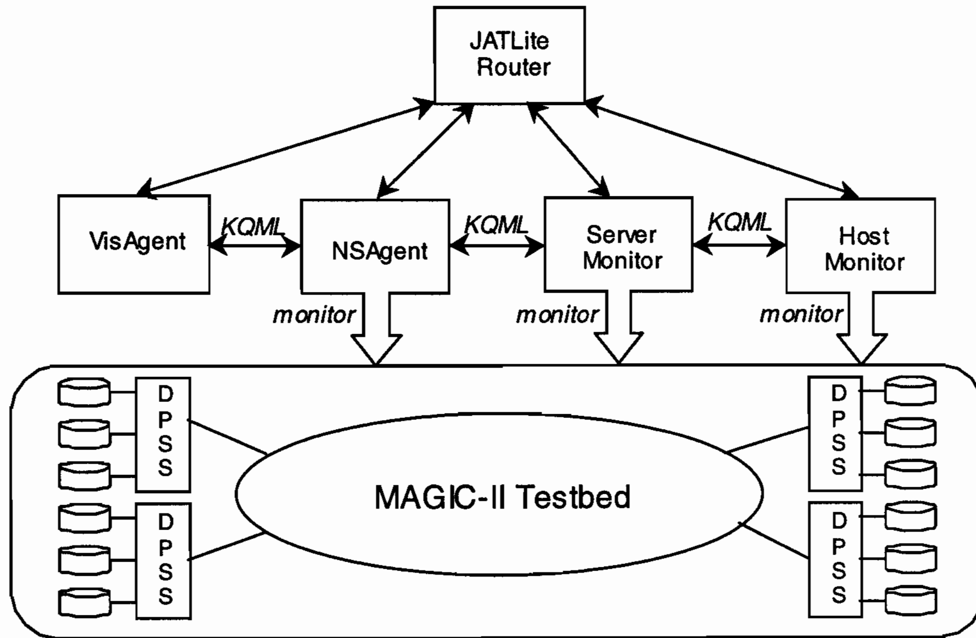


Figure 3 Functional Overview

There are four types of agents that we can identify from the picture:

- ***NSAgent*** is a JATLite agent which creates and schedules NetSpec experiments and organizes the performance reports. The type and parameters of the experiment can be loaded dynamically. NSAgent collects the information about the DPSS system from the ServerMonitor. This is developed by KU and will be described in more detail in this report.
- ***VisAgent*** is a JATLite agent with a front-end applet which visualizes the state of the network and distributed application and the agents configuration. The information is collected from the NSAgent and ServerMonitor. This is also developed by KU.
- ***ServerMonitor*** is a JATLite agent which monitors the status and configuration of a DPSS syste [2]. This agent is developed by the LBNL.
- ***HostMonitor*** is also a JATLite agent which keeps track of the status of the currently connected DPSS clients.

Each of these agents registers itself with the router when it starts up. It exchanges KQML messages with other agents in the system via the router.

[2] A DPSS system consists of a DPSS master and one or more DPSS server.

5 Implementation of NSAgent

The main responsibility of the NSAgent is to capture the state of some of the network characteristics. It does that by performing the appropriate test and measurement in the network. The result of monitoring and measuring the network characteristics can be used for different purposes. In the MAGIC-II testbed, the main objective is to use the knowledge about the current condition in the network to dynamically select the best server in DPSS system. The characteristics of particular interest are the load of the network which can be represented by the available link bandwidth and round trip time. The NSAgent is also used to perform general network monitoring such as connectivity or throughput test..

Thus, the main focus of the NSAgent implementation is to design a extensible framework which can accommodate future types of network measurement or testing. Although the main objective in this project is to support the dynamic reconfiguration of the DPSS system, NSAgent should fulfill the ultimate goal of capturing the network state.

5.1 NSAgent Architecture

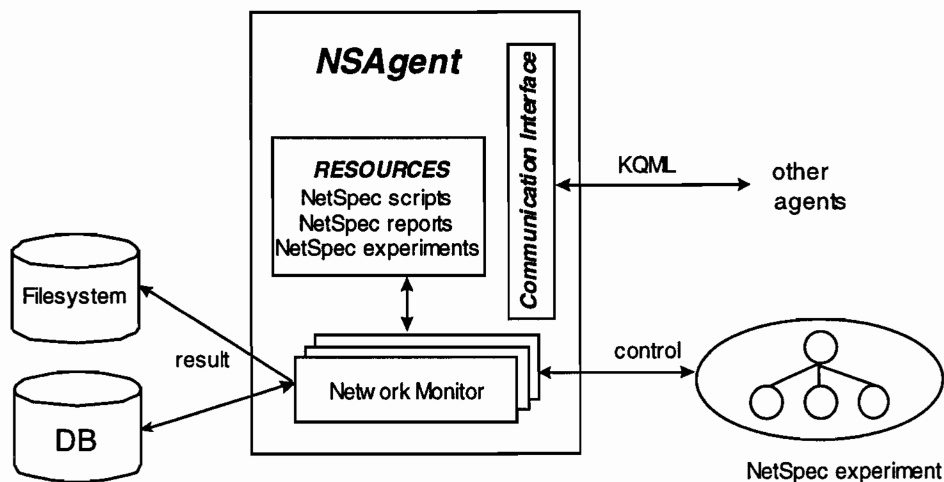


Figure 4 NSAgent Architecture and External Components

Figure 4 shows the architecture of the NSAgent. The JATLite's `RouterClient-Action` class provides the basic communication interface based on TCP/IP socket via the router for receiving and sending KQML messages. The NSAgent has a collection of templates for NetSpec scripts and reports and for creating NetSpec experiments. It usually receives a monitoring task from other agents and creates the appropriate network monitor. The results of the NetSpec experiments can be stored in a database or the filesystem.

The actions for the NSAgent is defined in the `NSAgentAction` class which is a subclass of the JATLite `RouterClientAction` class. Figure 5 shows the complete class hierarchy. If we refer back to the definition of these layers in section 3.2, we can see clearly that the NSAgent can handle KQML messages and can use the services of the router such as naming service and message routing.

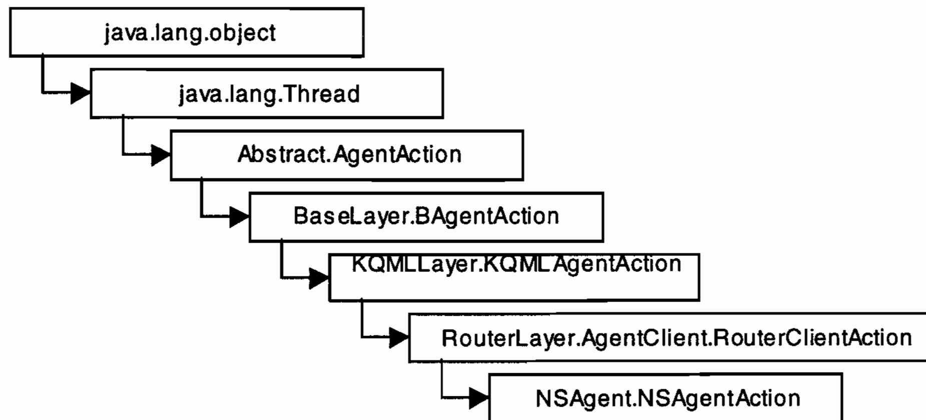


Figure 6 NSAgentAction Class Hierarchy

The message handling of a RouterClientAction agent is quite simple. When a KQML message destined for an agent arrives at the router, it stores the message in the incoming message box for that agent and then it will notify the agent. The agent is responsible of retrieving its own message and deleting them afterward. Figure 6 shows the pseudo-code of the `Act()` method in the `NSAgentAction` class.

```
class NSAgentAction extends RouterClientAction {
      . . .
      public boolean Act (Object o) {
            //create KQMLmail
            KQMLmail mail = new KQMLmail ((String)o, 0);

            //extract KQMLmessage
            KQMLmessage kqml = mail.getKQMLmessage();

            // parse and interpret message
            . . .
            // delete message
            addToDeleteBuffer(0);
      }
}
```

Figure 5 Pseudo-code for the Act() method in NSAgentAction

5.2 Knowledge Representation

One of the main components in a software agent is its knowledge base (KB). It provides the context of agent execution and the knowledge about its environment. The representation of knowledge can vary according to its purpose. NSAgent has two types of

knowledge representation. The *procedural representation* uses program functions or methods to represent the data and the operation associated with an object. This type of representation is particularly useful in defining the capabilities of an agent. The second type of knowledge representation used in NSAgent is a relational data base. It usuall serves as a back-end storage of static or run-time data. Relational data base provides a convenient and structured access and manipulation of data.

In NSAgent, the procedural representation is also called a *resource*. Section 5.2.1 will explain the types of resources which have been defined in NSAgent. Section 5.2.2 will describe the structure and configuration of the database.

5.2.1 Resources

NSAgent mainly uses resources to store the knowledge needed for running network experiments. Figure 7 shows the conceptual process of creating a network experiment. Given a task to monitor a specific network characteristic, NSAgent will create the appropriate *network monitor* object. The monitor object will create the *NetSpec script* that will invoke the proper NetSpec measurement daemon for the task. Then it will need to parse the performance *report* generated by the NetSpec daemon and interpret the result.

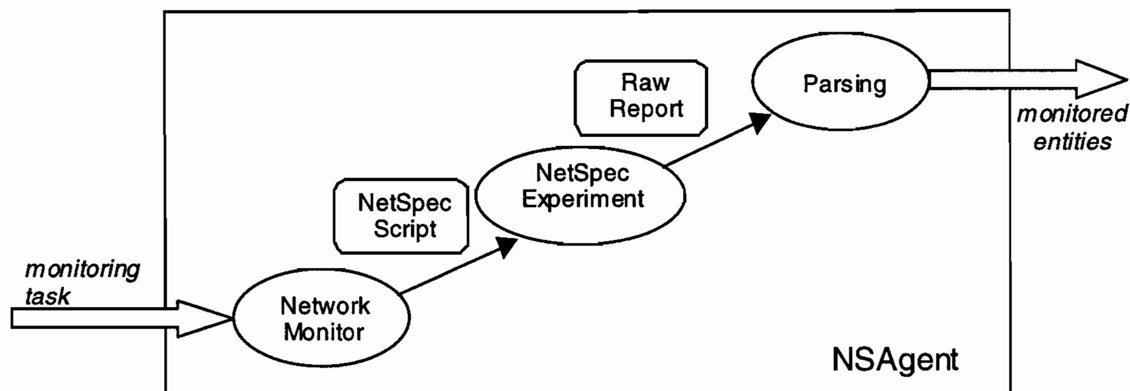


Figure 7 Network Monitoring Process

NSAgent defines the following resource abstractions:

- Network Monitor:
  It represents the object that controls NetSpec experiment. Every network monitor object encodes the *type* and *topology* of NetSpec experiment. The type is associated with the kind of NetSpec measurement daemon which needs to be executed. The topology defines the execution construct and the participants of the experiment. A monitoring task must specify the type of monitor object to create. It can also specify the duration of the experiment and storage method.
  For example, we can define a monitor object which performs a full mesh throughput experiment among N nodes. In this case, the type of the experiment (i.e. throughput) indicates that we have to use NetSpec Test Daemons. The full mesh topology will creates a script of N^2 pair of end to end experiment between each two nodes in the set

of N given nodes.
Table 1 summaries the types of network monitors that have been implemented so far.

Name	Topology	Daemon type
FullMeshThroughput	N-to-N (full mesh)	*nstestd*
PointToMultiPointThroughput	1-to-N (star)	*nstestd*
EndToEndThroughput	1-to-1	*nstestd*
FullMeshDelay	N-to-N	*nspingd*
PointToMultiPointDelay	1-to-N	*nspingd*
EndToEndDelay	1-to-1	*nspingd*
HostMonitor	1-to-N	*nssnmpd*

- NetSpec Script:
  A NetSpec script object contains the parameters and options of a NetSpec daemon and the methods to generate the script. A class must be defined for each NetSpec daemon since each NetSpec daemon has different parameters and format.
  For example, a NetSpec script class of the NetSpec Test Daemon defines the test parameters such as type of traffic, protocol and end-nodes. It also needs to implement the method that generates a block structure describing a this test.
  At the writing of this report, three types of NetSpec script classes have been defined: NSTestd, NSPingd, and NSSnmpd.

- NetSpec Report:
  Since there is no standard report format defined in NetSpec, each daemon generates varying types of report which makes parsing and interpretation of the report particularly difficult. To handle this problem, NSAgent defines a NetSpec report class for each type of daemon. Each class provides methods to parse and store the result.

5.2.2 Database

For this project, we have decided to use the *msql* (mini SQL) [6], a light-weight relational database based on SQL as the back-end storage. The NSAgent uses the database to store static configuration of the network (such as names and addresses of end hosts or switches) and the configuration of the distributed application being monitored, in this case, the DPSS. The results of network experiments can also be optionally stored in the database for further retrieval by other agents. Since the goal is to capture the state of the network, only the most recent result is kept in the database. Historical result can optionally be cached in the filesystem.

5.3 Monitoring Task

NSAgent's main capability is to accept monitoring task and perform the monitoring action. Monitoring task is encapsulated in a KQML message and can be submitted by other agents in the system. NSAgent also provides a convenient way of loading tasks from a file during the agent start up. Routine monitoring tasks can be loaded from this initialization file.

A monitoring task must identify the following attributes:

- The type of network monitor.
- The frequency of the experiment.
- The storage method (to database or filesystem)
- The parameters of the experiment (specific to the type of network monitor)

5.4 KQML Messages

This section will describes the KQML messages implemented in the NSAgent. The types of messages can be broadly categorized into three domains according to the audience of the messages:

- DPSS
  Messages in this domain deal with the configuration of the DPSS and the queries about the network status from the DPSS.
- NetSpec
  These messages provide interface to create or terminate NetSpec experiment, including the loading of monitoring task.
- Visualization
  The NSAgent also serves as the information provider for a visualization agent

Appendix A contains the detail description for the KQML messages.

6 Implementation of VisAgent

VisAgent is a front-end interface to the monitoring systems described in this report. It aims at providing geographical display of the state of a distributed application and its underlying wide-area network. The logical approach is to make the visualization tool itself an agent that communicates with the monitoring agents and hence, the name VisAgent.

VisAgent is implemented as a Java applet which can be loaded from a web browser or launched as a Java application. This approach provides the convenient access for a thin client to access the visualization tool from various location or environment. JATLite's router mechanism plays an important role especially for this type of agent because the only information needed by the VisAgent to communicate with the rest of the system is the address of the router. It also solves the security problem imposed by web browser on Java applet which only permits applet to create sockets to the same host where the web server resides. As long as the JATLite's router and the web server are configured to run on the same hosts, the applet can be loaded from anywhere.

VisAgent uses both polling and event-driven mechanism in updating the display. Polling is mainly used to collect information which are supplied by other agents, while event driven is used to collect information from the database. This strategy is used to achieve the reactiveness of the visualization tool. Query to an agent usually takes significantly more time than query to a database. Therefore, user's action usually only triggers query to a database and updated view.

6.1 GenMap Package

The main goal of VisAgent is to display information based on their *geographical* position. Therefore, we need a user interface with a map overlaid with visual symbols to represent state of the system. The GenMap package [7] provides a very nice starting point to achieve this goal. The package consists of a set of Java classes which provides the basic functionality for geographical network visualization. It implements the classes to draw the background map, nodes and lines and methods to zoom in and out the map.

Considerable amount of time has been devoted to adapt the GenMap package for this project. GenMap is really specific as to the format and size of background map used. The original package uses a flat map of the whole world which then can be zoomed in to a particular continent. If the resolution of the base map is not good enough, the zoomed version of a continent will be of very poor quality. Since this project is particularly interested in providing visualization for the United States region, we want to start with a US map as the base map. The solution is to modify the `ImageProducer` class which supplies the pixels to be drawn on the screen so that we can start with a map of a particular region bounded by a rectangle of the given latitude-longitude pairs.

Another addition to the GenMap package is the thumbnail map which shows a rectangle bounding the current display on the base map. This feature is particularly useful if the base map does not have details such as state lines or city names. Users can always refer to the thumbnail image to figure out which part of the map they are looking at.

6.2 Visualization Layers

The VisAgent collects information from various sources and tries to aggregate them to form a unified view of a distributed application and its underlying wide-area network. The best way to organize the data is to group them by the source of information. Visually we can provide the display as viewed from a specific layer. By separating the information in layers, we can potentially display many characteristics of the application and the network in one convenient visualization tool.

VisAgent provides three layers of visualization. The first layer is for the distributed application, the second for the network and the third is for the monitoring agents. The next few paragraphs will describe each layer in detail and shows the screen shot.

Application layer shows the location of the components of the distributed application and the status of each component. In the MAGIC-II context, each node represents either one of the DPSS master, server or client. The lines represent the active connection from a client to the server(s). This layer provides the information about the number of servers, the location and configuration of each and identifies the location and status of the client.

Network layer shows the configuration of the network and the results of the measurement done on the network. Each node represents a site in the testbed. Lines represent the physical connection or the network characteristics. The width of the lines usually reflects the value of the network characteristics they represent. This layer also provides the detail configuration of network elements (e.g. switch) and end-hosts in each site.

Agent layer shows the configuration of the monitoring system, i.e. the geographical location and the address of the agents. It also shows the topology and status of the active

network experiments. This layer can provide useful information to understand the components and interaction between elements in the monitoring system.

6.3 Visual Element Mapping

For each visualization layer, each node and line can represent different entity and value. A node can be varied in terms of shape, size and color. Line can only vary in size and color. The mapping of the attributes of the nodes and lines can either be hard coded in the program or dynamically reconfigurable at run time. To make our tool as general and flexible as possible, we choose the second approach.

For each layer we define a set of mapping for the nodes and lines and provide an efficient API for the programmer. The GenMap's base class for node and line have been modified so that the attribute binding is done as late as possible, for example just before the node and line are displayed on the screen. The mapping can be defined in a configuration file which is loaded initially. The attributes of a node or a line are assigned by consulting the rules defined in the configuration. The configuration is also used to create legend for each visualization layer that is updated every time the view changes to another layer.

For each layer, two lines describing the node and line's attributes, respectively, must be specified in the configuration file. Table 2 summarizes the attributes for the node and line and acceptable values for each attribute.

Element	Attribute	Value Syntax	Description
Node/line	LABEL	*Name*	The type of entity this node/line represent
	UNI	*Name*	The unit for the value represented by this node/line
	COLOR	FIX (*color*)	The hex value of the color for all nodes/lines
		RANGE (*min, max*)	The color of this node/line can vary according to the value which lies between *min* and *max*
		LIST ((*label1,color1*), (*label2, color2*) ...)	Node/line with *label1* is colored *color1*, and so on
	SIZE	FIX (*size*)	All nodes/lines have the same size
		RANGE (*min, max*)	The size of nodes/lines vary according to the value which lies between *min* and *max*
		LIST ((*label1, size1*), (*label2, size2*) ...)	Node/line with *label1* is of size *size1*, and so on
Node	SHAPE	FIX (*shape*)	All nodes have the same shape
		LIST ((*label1, shape1*), (*label2, shape2*) ...)	Node with *label1* is of shape *shape1*, and so on

6.4 KQML Messages

The KQML messages implemented by VisAgent mainly deal with messages that encapsulate the queries to other agents or database. Some KQML messages contains update notification from other agents. For example, NSAgent can send a KQML message to tell the VisAgent that a NetSpec experiment with certain topology and involving some nodes has just started.

Appendix B describes all the KQML messages implemented by VisAgent.

7 Demonstration Overvie

This section will describe the configuration and capabilities demonstrated during the MAGIC-II quarterly meeting on July 14, 1998 at the University of Kansas.

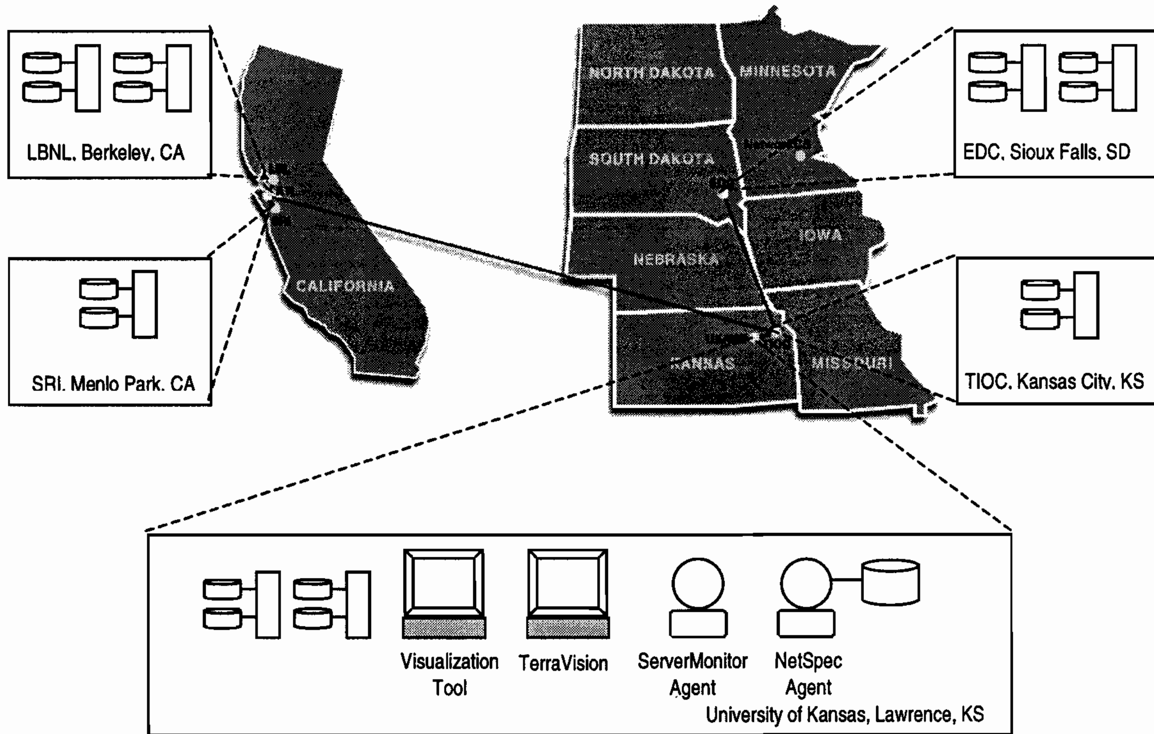


Figure 8 Demonstration Configuration

Figure 8 shows the configuration of the demonstration. We had 8 DPSS servers distributed across the MAGIC sites. The agents, the JATLite router, and the database server and the web server are running on *faraday* (a Sun Ultra Sparc running Solaris 2.6) at KU. NetSpec is installed at all DPSS hosts and at least one machine at each site.

7.1 NSAgent Configuration

For the demonstration, NSAgent was configured to schedule and run a number of network experiments by using NetSpec. Each network measurement was encapsulated in a monitoring task presented to the NSAgent during its initialization. By default, NSAgent loads a series of KQML messages from an initialization file which defines the agent's initial behavior. Appendix A describes the KQML messages implemented by the NSAgent.

The following subsections will describe each monitoring task and its significance as well as some example results obtained from the measurement when appropriate.

7.1.1 Monitoring Link Quality between DPSS Client and Servers

The objective of this activity is to determine the link quality between a DPSS client and the servers configured in the system in order to select the server with the best connectivity to the client. The link quality is defined in term of round trip delay and transfer capacity of the link.

When a new DPSS client comes up in the system, the NSAgent must be notified about its existence. The entity that must register the client to the NSAgent can be the client itself or a DPSS monitor agent which continually keeps track of the emergence of a new client. Since neither of these approaches had been implemented by the demonstration date, a static configuration was used instead. The following KQML message was included in the initialization file to register a DPSS client named *tv-client* with only one network interface, *terravision.ukans.magic.net*, which will potentially talk to one of the eight DPSS servers.

```
(evaluate :sender NSAgent :receiver NSAgent
          :content (tell-resource :type client :name tv-client
                        :pos (38.963 –95.233)
                        :interface (terravision.ukans.magic.net 198.207.143.157)
                        :server (lbl-server4 sri-server1 tioc-server1 tioc-server2
                              lbl-server3 ku-server1 edc-server1 edc-server2)))
```

Upon receiving this message, the NSAgent instantiates two types of network monitors (see Table 1):

1. A *PointToMultiPointThroughput* monitor measures the transfer capacity (throughput) of the links between the client and the servers. If either the client or the server has more than one network interface, each interface needs to be tested. This experiment is done once every 30 minutes. The results of the experiment is stored in the database.
2. A *PointToMultiPointDelay* monitor measures the round trip time between the client and the servers. Since this type of experiments does not produce a significant disturbance to the network, it can be done more frequently (once every 15 minutes) without consuming too much network resources.

7.1.2 Monitoring Connectivity in the Network

The objective of this task is to monitor the connectivity in the network. Sometimes it is difficult to identify the cause of the problem when an "Unreachable destination" message is received because the network has many potential points of failure. This is especially true in an experimental testbed such as MAGIC-II where the configuration of the network may change frequently. By observing the MAGIC's ping-pages in various sites, man time we found that the reachable machines/sites varied greatly from one machine to the next in the network.

The objective of the measurement activity described in this section is to provide a better understanding about how the sites in the network are connected to each other and what kind of connectivity exists currently. Every 15 minutes, NSAgent schedules a *FullMeshDelay* experiment among the major hosts in each site in the network. Since

*FullMeshDelay*network monitor uses the ICMP_ECHO mechanism to measure the round trip time, it can be utilized to test the connectivity from one point in the network to several other points.

7.1.3 Monitoring Transfer Capacity of the Network

Besides monitoring the connectivity, another important metrics in assessing the general health of a network is to test the transfer capacity (throughput) of the links in the network. Variation in throughput is generally affected by the amount of traffic and the presence of bottleneck links in the network. For this purpose, NSAgent creates a *FullMeshThroughput* experiment among the major hosts in each site. Since this type of experiment introduces a large amount of test traffic to network, it should not be done too frequently. However it is important to note that to study the variation in throughput during the course of a day, it should be done at different times during the day. In the demonstration, NSAgent schedules this experiment once every 3 hours.

7.1.4 Monitoring Network Element Status

While the measurement activity described in Section 7.1.2 is aimed at providing information about connectivity between sites in the network, it does not provide detail information about each machine or other network element within a site. This type of monitoring can be achieved by doing a *PointToMultiPointDelay* experiment between a host in a site to the remaining hosts and switches in each site. The database contains the static information about the configuration in each site. NSAgent uses this information to create experiment that sends ICMP_ECHO message every 15 minutes from a designated host to the rest of the network element and hosts in its site.

7.2 VisAgent Configuration

In the demonstration, the VisAgent is started by loading the Java applet for the visualization from a Java capable browser. VisAgent collects the performance data and static configuration from three sources: the NSAgent, the database server and the ServerMonitor agent. KQLM messages are used in communicating with the agates, while standard SQL messages are used to interact with the database server. Appendix B describes the KQML messages which are used by the VisAgent to interact with the other agents in the system. As described in Section 6.2, VisAgent provides three layered views:

- Application Layer
  Color-coded nodes represent the location of DPSS master, server and clients. The placement of the nodes indicates the geographical location of the corresponding DPSS components. Lines represent the active connection between a client and some DPSS servers. Figure 9 shows the screen capture of the visualization tool displaying the status of the DPSS. In this figure, a DPSS client, *ku_client1*, is accessing data set

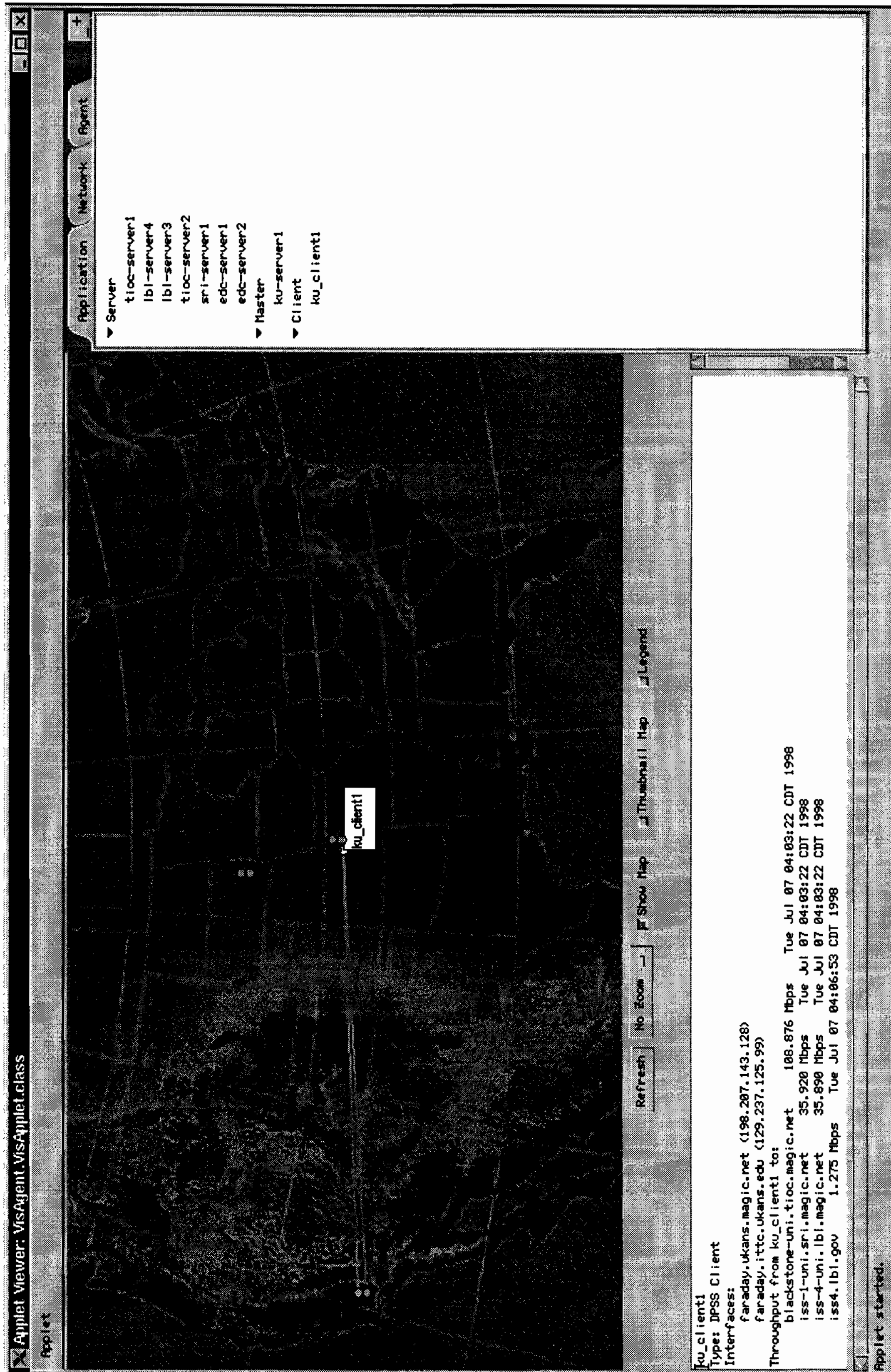


Figure 9 Screenshot of VisAgent at the Application Layer. Nodes represent DPSS master, servers and clients. Lines represent active connection between a DPSS client and servers.

from two DPSS servers as shown by the lines connecting the client and the servers. The data panel at the bottom left corner shows the detail information about ku_client1, such as its network interfaces and the results of the throughput measurements. The panel on the right side displays the names of the DPSS master(s), server(s) and client(s) that are currently registered with the monitoring system.

Actions on this layers include [3]

- Clicking on a DPSS master or server's node will send a *GetHostInformation* query about that node to the ServerMonitor.
- Clicking on a client's node will send a *LinkInformation* query to the NSAgent about the latest throughput and delay numbers from that client to the servers. The following example shows the information displayed about a client:

```
Name: ku_client1
Interfaces:
        faraday.ukans.magic.net (198.207.143.128)
        faraday.ittc.ukans.edu (129.237.125.99)
Throughput from ku_client1 to:
        blackstone-uni.tioc.magic.net  108.876 Mbps  Tue Jul 07 04:03:22 CDT 1998
        iss-1-uni.sri.magic.net         35.920 Mbps  Tue Jul 07 04:03:22 CDT 1998
        iss-4-uni.lbl.magic.net         35.090 Mbps  Tue Jul 07 04:03:22 CDT 1998
        iss4.lbl.gov                    1.275Mbps   Tue Jul 07 04:06:53 CDT 1998
```

- VisAgent periodically polls the ServerMonitor to collect the information about the currently connected client by sending *GetMasterInformation.* The information typically includes the user name, the program name and the data sets accessed by the client. When a new user is detected, the view is updated with lines representing connection from the user to the corresponding DPSS servers.

- Network Layer
  Each node at the network layer represent a site in the MAGIC-II testbed. The network layer is further divided into 3 sublayers.
  - The topology sublayer shows the physical configuration of the testbed. The width of the lines represent the physical link's capacity (i.e. DS-3, OC-3, OC-12). Clicking on a node will bring up another window which shows the detail configuration and status of the network elements in each site. Elements which are down are colored differently from elements which are up. Figure 10 shows the screenshot of the VisAgent displaying the topology sublayer. The smaller windows shows the detail configuration at KU site. The panel on the right displays the names of the network elemtns and hosts at each site. Green color is used for network elements and blue color is used for hosts. Any element or host which is not responding to a PING message is indicated by red color.
  - The connectivity sublayer shows the result of the connectivity test from a point of view of a site. The color of the lines coming out from a site to another site represents the round-trip-time in millisecond. Unreachable site will not be connected by a line. Figure 11 shows the screenshot of the VisAgent displaying the connectivity as seen from EDC site. The color of the lines represent the round

[3] Please refer to Appendices A and B for detail information about each KQML queries

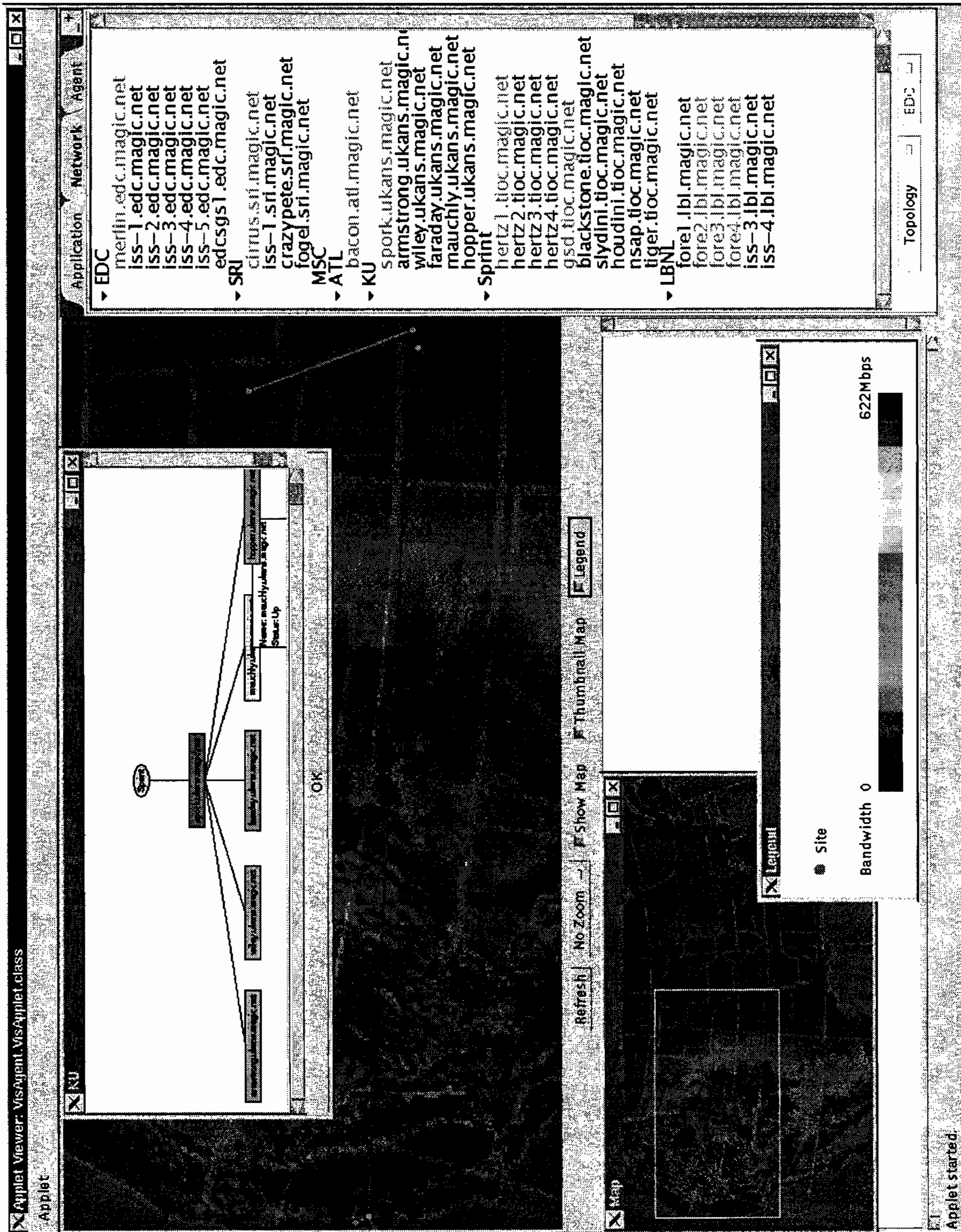


Figure 10 Screenshot of VisAgent at the Network Topology Sublayer. Nodes represent sites (organizations). Lines represent the bandwidth of the physical connection between sites.

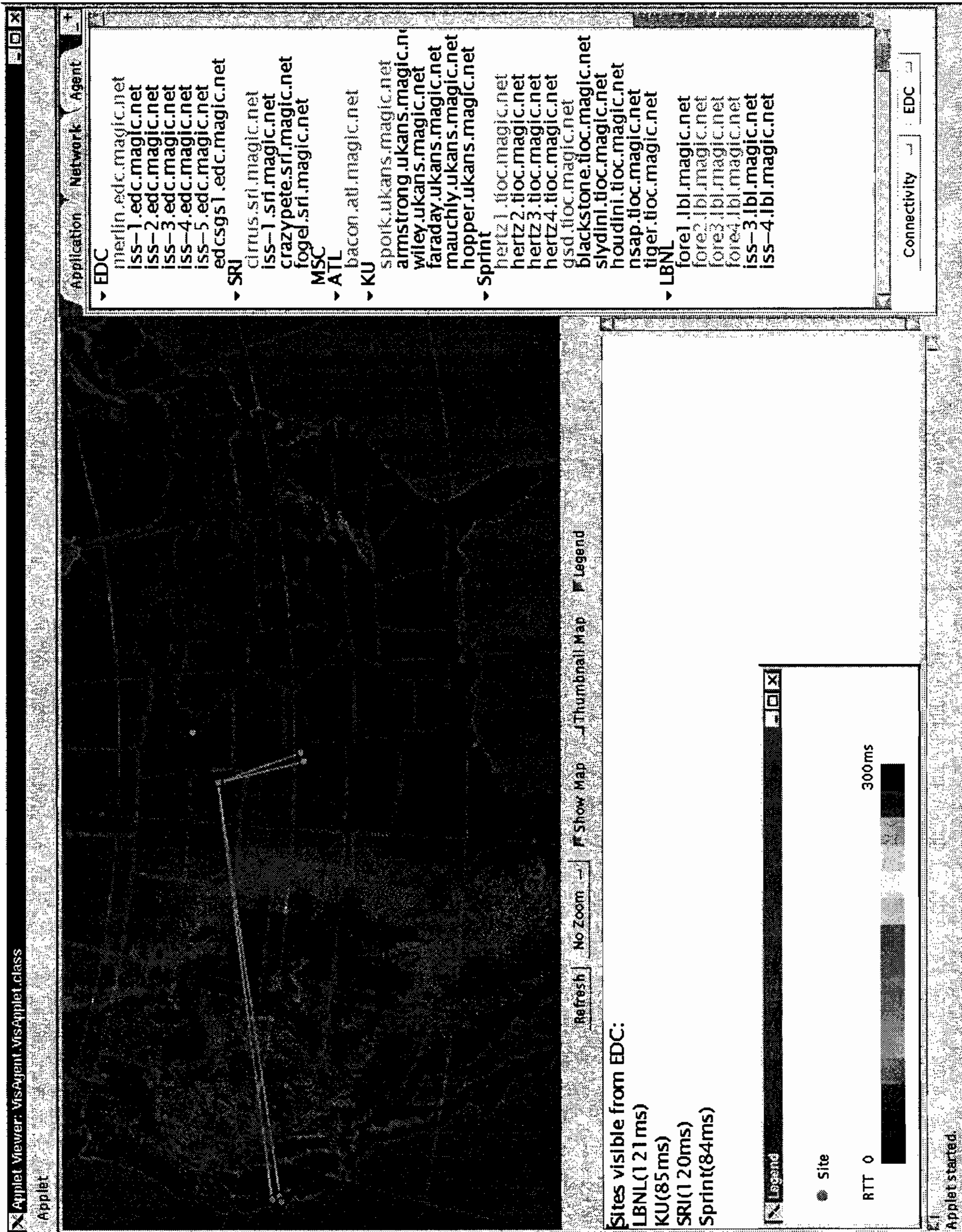


Figure 11 Screenshot of VisAgent at the Network Connectivity Sublayer. Nodes represent sites (organizations). Lines represent existing connectivity and their color represent the round trip

trip time values as indicated by the color spectrum in the legend window. The numerical values of the round trip time are also displayed in the data panel.

- The maximum bandwidth sublayer shows the result of the throughput test from a point of view of a site. The color of the lines represents the achievable throughput in Mbps.

- Agent Layer
  Each node represents either a JATLite agent or a NetSpec daemon. The color of the node indicates the types of agent or NetSpec daemon. Lines connecting NetSpec daemon's nodes indicate the topology of the experiment. The view of the agent display is automatically updated when new experiment begins and terminates. For example, Figure 13 shows the visualization at the agent layer when a full mesh delay experiment is taking place. The blue nodes represent the NetSpec Ping Daemons involved in the experiment. The lines between the nodes represent the topology of the experiment.

The script used to configure the visual element mapping is given in Appendix C.

8 Conclusion and Future Work

This report has described the design and implementation of NSAgent and VisAgent which are two of the agents in the MAGIC-II monitoring system. Both of the agents are implemented in Java, use the JATLite framework and communicate using KQML.

NSAgent's main objective is to perform measurement and test on the network in order to capture some characteristics of the network. NSAgent relies on NetSpec to perform the distributed network experiment. The result of the experiment is either stored in the database or in the filesystem.

VisAgent aggregates and correlates the information from various agents to provide a unified visualization of the distributed application and the network. VisAgent is implemented as a Java applet that can be loaded from any web browser.

During the implementation of the system and experimentation, several areas of improvement can be identified:

- As mentioned in section 7.1, the list of candidate clients must be maintained and updated dynamically because NSAgent requires this information to determine which experiment needs to be run. Another possibility which may require more serious research is to identify the type of measurement that can determine the best server in less amount of time.

- The current implementation of NSAgent has some method and messages specific to the DPSS domain which may not be applicable to other areas of application.

- The ease of implementation of network monitors can be greatly improved if the topology information is separated from daemon-specific parameters/information. Since the topology of experiment is usually applicable to different type of NetSpec daemons, most of the topological abstraction can be reused.

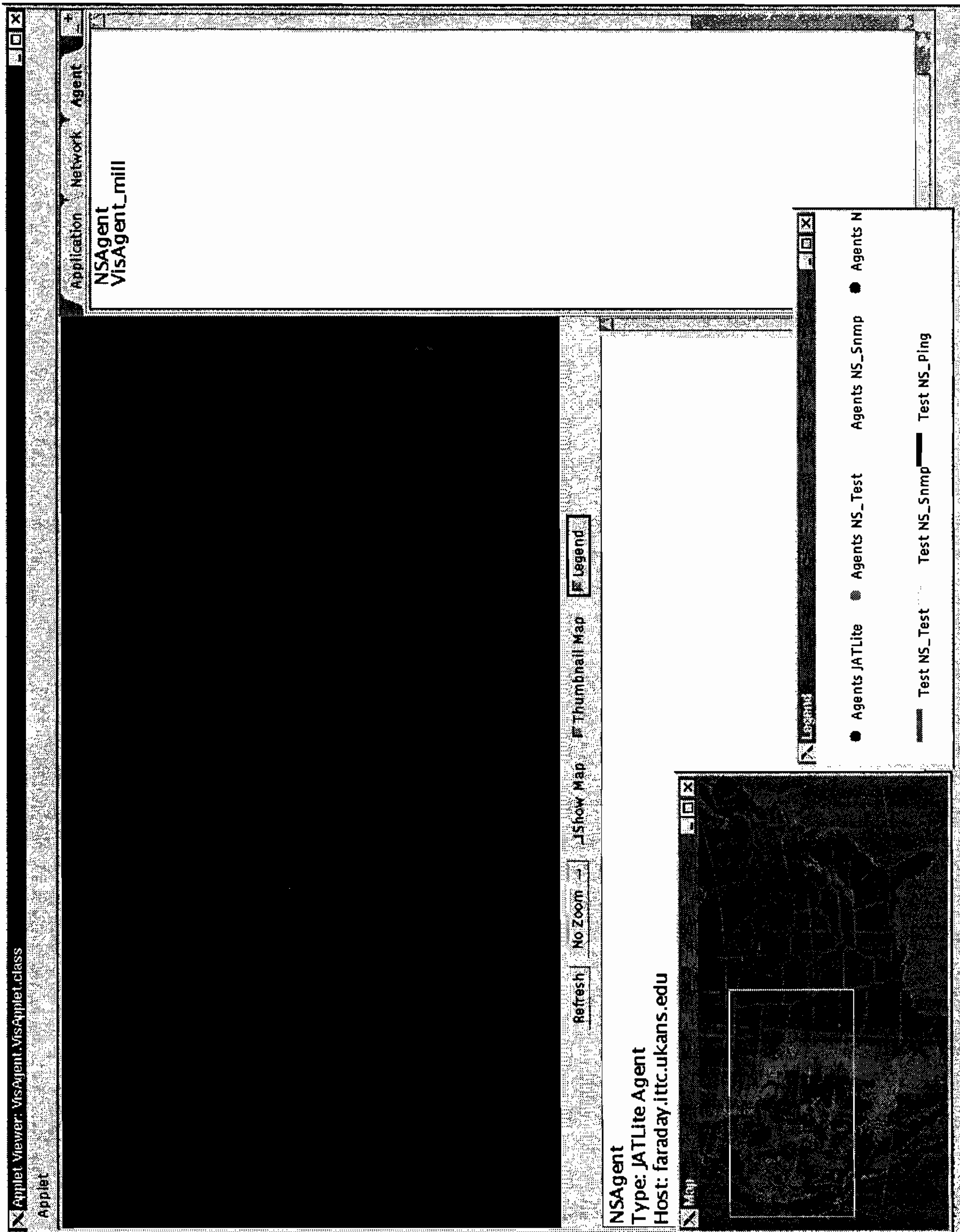


Figure 12 Screenshot of VisAgent at the Agent Layer. Nodes represent JATLite agents or NetSpec daemons. Lines connecting NetSpec daemon nodes represent NetSpec experiment's

- NSAgent should be made more general by creating an abstraction of a domain that consists of an interpreter, domain-specific data structure and configuration and tasks. Domain can be implemented as a Java class which can be loaded dynamically.
- A monitoring system is not complete without allowing continuous and real-time monitoring. For example, a user can specify some points in the network for on-the-fly data collection.
- The organization of configuration and historical data should be more standardized.
- The VisAgent currently does not provide an interface to browse the historical data.

Several of these issues will be explored and implemented in [10]; particularly, the issue of making the framework more general and the provision of continuous monitoring.

APPENDIX A. KQML Messages Implemented by NSAgent

- (evaluate :sender *xxx* :receiver NSAgent
  :content (tell-resource :type client :name *clientName*
  :pos (*lat long*)
  :interface (*iface1 ipaddr1 iface2 ipaddr2* ...)
  :server (*server1 server2* ...)))

  Register a new DPSS client. NSAgent will monitor the links connecting to the given set of servers for each network interfaces and store the information in the database.

- (evaluate :sender *xxx* :receiver NSAgent
  :content (invalidate-resource :type client :name *clientName*))

  Unregister DPSS client. NSAgent will stop monitoring the links connected to this client and remove information from the database.

- (ask-one :sender *xxx* :receiver NSAgent
  :content (GetLinkInformation ClientName *client* ServerName *server*))

  Get the throughput and delay values for the link(s) between *client* and *server*.

  (reply :sender *NSAgent* :receiver *xxx*
  :content (LinkInformation ClientName *client* ServerName *server*
  (Interface *iface1* Throughput *t1* Delay *d1*)
  (Interface *iface2* Throughput *t2* Delay *d2*) ...))

  Reply for the GetLinkInformation query. Throughput and delay values for ever network interface are returned.

- (ask-one :sender *xxx* :receiver NSAgent
  :content (GetKnownDPSS

  Get the names of all DPSS servers and clients known to NSAgent.

  (reply :sender *NSAgent* :receiver *xxx*
  :content (KnownDPSS ClientNames *client1 client2* ...
  ServerNames *server1 server2*...))

  Return the names of DPSS clients and servers known to NSAgent.

- (ask-one :sender *xxx* :receiver NSAgent
  :content (GetKnownClients)

  Get the names and addresses of all DPSS clients known to NSAgent.

  (reply :sender *NSAgent* :receiver *xxx*
  :content (KnownClients (ClientNam *client1* Interfaces *iface1* ...)
  (ClientName *client2* Interfaces *iface1* ...)

  Returns the names and addresses of all DPSS clients known.

- (ask-about :sender VisAgent :receiver NSAgent
  :content (dpss-clients))

  Get the information about the DPSS clients registered with NSAgent: name, position, servers.

  (reply :sender *NSAgent* :receiver *xxx*
  :content (dpss-clients :value '(*name lat long* (*server1 server2* ...))
  '(*name lat long* (*server1 server2* ...)))

  Returns the information about registered clients.

- (ask-about :sender VisAgent :receiver NSAgent
  :content (netspec-experiments))

  Get the information about active NetSpec experiments.

  (netspec-experiments :sender *NSAgent* :receiver *xxx*
  :content ((*NetMonName NetmonType NumHosts host1 host2* ...)
  (*NetMonName NetmonType NumHosts host1 host2* ...) ...)

  Returns the information about active NetSpec experiments.

- (ask-about :sender VisAgent :receiver NSAgent
  :content (network-monitors))

  Get the information about the status of network monitors created by NSAgent.

  (netspec-experiments :sender *NSAgent* :receiver VisAgent
  :content ((*NetmonName status*)
  (*NetmonName status*) ...)

  Returns the names and status (idle or running) of network monitors.

- (register-visagent :sender VisAgent :receiver NSAgent
  :name *VisAgentNam*)

  Register a VisAgent with NSAgent. A registered VisAgent will get updates about experiment results and status.

  (unregister-visagent :sender VisAgent :receiver NSAgent
  :name *VisAgentName*)

  Unregister a VisAgent.

- (create-experiment :sender *xxx* :receiver NSAgent
  :type *NetMonType* :param (*NetMonName :key1 value1 :key2 :value2* ...)
  :period *timeInSec* :saveData *boolean*)

  Create a network monitor with the given type, parameters, frequency and storage options.

  (delete-experiment :sender *NSAgent* :receiver *xxx*
  :name *NetMonName*)

  Delete a network monitor named *NetMonName*

- (terminate-agent :sender *xxx* :receiver NSAgent)

Terminate NSAgent. It will stop all network monitors and send an acknowledgement message to the sender.

(agent-terminated :sender NSAgent :receiver *xxx*)

Notify sender that NSAgent has terminated.

- (new-netspec-experiments :sender NSAgent :receiver VisAgent
  :content (*NetMonName NetMonType NumHosts host1 host2 ...*))

Notify VisAgent that a new NetSpec experiment is just started.

(remove-netspec-experiments :sender NSAgent :receiver VisAgent
:content (*NetMonName*)

Notify VisAgent that a NetSpec experiment has terminated.

Appendix B. KQML Messages Implemented by VisAgent

- (new-client :sender *xxx* :receiver VisAgent
  :name *ClientName* :pos (*lat long*) :servers (*server1 server2 ...*))

  Tell VisAgent that a new DPSS client is connected. VisAgent will update the view by adding the client node and connections to the specified servers.

  (remove-client :sender *xxx* :receiver VisAgent
  :name *ClientName*)
  Remove a DPSS client from the view.

- (registered-agents :sender router :receiver VisAgent
  :content ((*AgentName host port status*)
  (*AgentName host port status*) ...)

  Tell VisAgent about the names and status of all agents registered with the router.

- (new-netspec-experiments :sender NSAgent :receiver VisAgent
  :content (*NetMonName NetMonType NumHosts host1 host2 ...*))

  Notify VisAgent that a new NetSpec experiment is just started.

  (remove-netspec-experiments :sender NSAgent :receiver VisAgent
  :content (*NetMonName*)

  Notify VisAgent that a NetSpec experiment has terminated.

In addition, VisAgent also sends KQML messages to the ServerMonitor [8] to collect information about a DPSS host and the whole DPSS systems.

- (ask-one :sender *xxx* :receiver ServerMonitor
  :content (GetMasterInformation SystemName *systemName*))

  Get information about the DPSS master and servers in a system.

  (reply :sender ServerMonitor: receiver *xxx*
  :content (MasterInformation SystemName *systemName* isUp
  NumberOfUsers *numUser* NumberOfDataSets *numDataSet*
  TotalMemory *totalMem* TotalMemoryUsed *totalMemUsed*
  (ServerName *server1* TotalMemory *serverMem* TotalMemoryUsed
  serverMemUsed)...
  (UserName *userName* ProgramName *progName*
  IP_Address *ipAddr* HostName *host* SessionID *session*
  DataSetI *dataSetID* DataSetNam *dataSetName*
  NumberOfServersUsed *numServer*
  NamesOfServersUsed *server1* ...)...))

  Information about the DPSS hosts in the system. VisAgent uses the information about the users/clients to keep an updated view.

- (ask-one :sender *xxx* :receiver ServerMonitor
  :content (GetHostInformation SystemNam *systemName* NodeName *nodeName*))

Appendix C. Mapping Configuration File

The following mapping configuration file was used to configure the visual element mapping in the VisAgent.

```
#Mapping Configuration for VisAgent
#AppLayer
LABEL = DPSS; COLOR = LIST((MASTER, ff0000), (SERVER, 00ff00), (CLIENT, ffff00), (END));
SIZE = FIX(3); SHAPE = FIX(1);
LABEL = Connection; COLOR = RANGE(0,4) ; SIZE = FIX(2);
#NetTopLayer
LABEL = Site; COLOR = FIX(ff00) ; SIZE = FIX(3); SHAPE = FIX(1);
LABEL = Bandwidth; UNIT = Mbps; COLOR = RANGE(0,622) ; SIZE = FIX(2);
#NetConnLayer
LABEL = Site; COLOR = FIX(ff00) ; SIZE = FIX(3); SHAPE = FIX(1);
LABEL = RTT; UNIT = ms; COLOR = RANGE(0,300) ; SIZE = FIX(2);
#NetBwLayer
LABEL = Site; COLOR = FIX(ff00) ; SIZE = FIX(3); SHAPE = FIX(1);
LABEL = Throughput; UNIT = Mbps; COLOR = RANGE(10,155) ; SIZE = FIX(2);
#AgentLayer
LABEL = Agents; COLOR = LIST((JATLite, ff0000), (NS_Test, 00ff00), (NS_Snmp, ffff00),
(NS_Ping, 0000ff), (END)); SIZE = FIX(3); SHAPE = FIX(1);
LABEL = Test; COLOR = LIST((NS_Test, 00ff00), (NS_Snmp, ffff00), (NS_Ping, 0000ff), (END));
SIZE = FIX(2);
```

References

1. The MAGIC-II Project, http://www.magic.net
2. T. Finin et. al., *DRAFT Specification of the KQML Agent Communication Language*, unpublished draft, 1993, http://www.cs.umbc.edu/kqml
3. B. Tierney, et al. *An Overview of the Distributed Parallel Storage System (DPSS)*, http://www-didc.lbl.gov/DPSS/Overview/DPSS.handout.fm.html
4. R. Jonkman, D. Niehaus, J. Evans, V. Frost. *NetSpec: A Network Performance Evaluation Tool*, submitted to SIGCOMM'96, February 1996.
5. Center for Design Research University of Stanford, *JATLite: The Java Agent Template*, http://java.stanford.edu
6. J. Gosling, H. McGilton. *The Java Language Environment: A White Paper*, Sun Microsystems, 1995.
7. D. Hughes. *Mini SQL: A Lightweight Database Server*, Bond University, Australia, http://www.hughes.com.au/library/msql1/manual/
8. The Cooperative Association for Internet Data Analysis (CAIDA). *The Genmap Package.* http://www.caida.org/Tools/Genmap/
9. B. Crowley, *KQML Messages in the DPSS Agent Monitoring System*, http://www-itg.lbl.gov/~crowley/kqml.html
10. Y. Wijata, *Implementation of A Scalable Agent-based Network Measurement Infrastructure to Improve the Performance of Distributed Application*, Master Thesis, University of Kansas, November 1998