JAVA-BASED MOBILE ASSET LOCATION

Luca Deri
[1]
Finsiel S.p.A.
Pisa, Italy.

Owing to the way networks grow and the advent of mobile computing, the task of physically locating assets is becoming increasingly complex. Network management tools are usually not suitable for management of dynamically moving assets and provide almost no facilities for asset localisation. In addition, asset management products delegate to human operators the task to identify physical asset's location.
JLocator is a Java-based system that allows both static and dynamic assets to be physically localised. Based on SNMP and Java, it enables users to locate assets through a simple web interface and allows asset management systems to take advantage of asset location information provided by JLocator. Finally, its distributed architecture makes it scalable and completely platform-independent.

Keywords: Asset Location, Java, Web, Network Management, SNMP.

1. Introduction

The advent of mobile computing and home working combined with increasingly short computer lifetime has greatly complicated system administration and service provisioning. This is because IPv4 (Internet Protocol version 4) provides little support for mobility hence a mobile computer that is used in different sites usually needs to change its IP address. In addition, whenever the computer is attached to the network by means of dial-up services or BOOTP/DHCP [12] [13], there is definitively no guarantee that the computer's IP address (both numeric and symbolic) will be persistent across multiple connections. The consequence is that management applications might not be able to physically locate a network resource and access services provided by mobile computers when their IP addresses change.

Asset management systems enable network administrators to track company assets and provide them information such as asset manufacturer, installed applications, inventory ID and provided services. Typically, asset management systems assist the administrators for purchasing new equipment, detecting computing resources and allocating maintenance/asset cost for each group or division of the organisation. Unfortunately, commercial asset management systems [2] [3] [4] are usually not integrated with asset location systems hence they offer almost no facilities for automatically locate assets and keep track of their movements. Operators are then responsible to update the information concerning asset location -contained in the asset database- whenever assets change position. Owing to this half-automatic asset management procedure, it can happen that the asset database contains a mixture of fresh and outdated information that may confuse operators.

System administrators once in a while have to fight against malfunctioning computers that cause network-wide problems. In these cases, administrators cannot always solve the problem from remote but rather have to walk to the room where the malfunctioning computer is located and fix the problem on system's console. Modern management systems [20] [21] allow administrators to create network maps that group assets logically according to the network where they belong. Unfortunately, they lack facil-

1. Finsiel S.p.A., Via Matteucci 34/B, 56124 Pisa, Italy, Email: l.deri@finsiel.it, Ph. +39/050/968.639, Fax +39/050/968.626

ities for creating maps where assets are depicted according to their physical location. Especially in large and dynamic networks, computer location maps are difficult to keep updated by hand, and then an automatic asset location system may become indispensable.

Computer-based telephony applications are slowly substituting conventional telephony equipment. The basic idea is that each mobile user has a computer to which incoming telephone calls are diverted. Network servers are then responsible to transparently divert calls to the actual mobile host position. Those servers need to transparently track computer movements and to provide, for instance to secretaries, information such as room and building where the computer user is currently located in order to deliver him/her plain mail or other non-electronic documents.

All the above examples show that it is becoming increasingly important to create tools that keep track of the physical host position in order to:

- Locate malfunctioning computers connected to a large or dynamic network.
- Allow some applications such as computer-based telephony applications to operate.
- Find out at any moment how to contact the user of a mobile computer. For instance, an asset-tracking system should provide the number of the telephone closest to such user.

The SNMP [9] community attempted to address the problem of asset localisation by specifying the *sysLocation* variable inside the MIB-II MIB [8]. *sysLocation* contains the location of the asset in a unspecified form such as building, floor, and room. In order to use this solution for tracking assets:

- a SNMP agent must run on each asset;
- the asset should be active all time because the agent running on the asset needs to reply to SNMP queries;
- network administrators must manually guarantee that the value of *sysLocation* is kept updated whenever the asset changes position.

Because such update process is not automated, it is definitively possible that *sysLocation* contains an outdated value with respect to the actual asset position or that its format is not recognised. Due to this limitation, the use of *sysLocation* is not satisfactory because this solution still relies on human operators. A modern asset location system, once configured, must be fully independent without having to rely on humans or on particular software that has to constantly run on client computers. Some vendors offer expensive asset location systems targeted for telephone networks that unfortunately rely on ad-hoc network equipment. The lack of reliable, fully automated, software-only, asset location systems has motivated the development of JLocator.

2. JLocator Design Goals

JLocator is an asset location system written in pure Java [1] that allows assets to be dynamically localised. The design goals are:

- Ease of use from both the administrative and user's point of view.
  The ease of use is an important factor because JLocator may be used by administrative people and not by computer experts. These people are responsible for purchasing/allocating assets and could have limited computing experience. Due to this, JLocator offers a friendly user interface, usable by non-computer experts, that depicts the assets where they really are located and hides inner network details.
- Ability to track assets independently of their type, operating system and network protocol.

Due to the heterogeneity of computers present in large organisations, JLocator has to be able to track every asset independently of its type[2] without having to rely on any software running on the asset. This prerequisite is very important because:

- it guarantees that JLocator can operate even if the assets are not constantly attached to the network;
- it prevents JLocator to stop working whenever the client software is not operational or the software version is not the expected one.

- Minimal impact on performance, resource utilisation and network traffic.
  Following the same principle adopted by SNMP, JLocator has minimal impact on the performance of the host where it runs and, even more important, on the global network traffic (polling should be very limited, if any).
- Distributed architecture that allows the creation of a web of systems able to locate assets on multiple locations.
  Large organisations have different sites and branches. JLocator keeps track of all the assets present on the organisation. Instead of having a centralised asset tracking system, the distributed nature of asset data should be exploited. It is therefore preferable to have several JLocator instances that keep track of local assets and collaborate to perform global asset tracking.
- Web-based user interface that allows users to easily locate assets.
  JLocator should have a simple user interface accessible from a standard software application. Because most of the users (if not all) have a web browser installed, using JLocator from a web browser has several advantages:
  - there is no need to install any software application on client machines;
  - web user interface is well known and does not require any additional training;
  - JLocator's updates do not require any software update on the clients but just the installation of the new version on the host where JLocator runs.
- Portability at binary level.
  Especially in large organisations it may not be possible to speculate on the kind of operating system JLocator will run on. Portability at binary level allows JLocator to run unmodified on virtually any operating system. In order to achieve this degree of portability, JLocator has been written from the ground up using the Java language.
- Completely automatic asset tracking.
  Asset tracking has to be performed fully automatically in order to avoid dependency on human operators responsible for updating asset information. This avoids synchronisation problems regarding asset information and guarantees that such information is constantly updated as assets change position. In addition, asset tracking systems should not rely on a particular network/topology but must be able to operate on any point-to-point network.
- Ability for external applications to take advantage of asset location information.
  Applications such as asset management applications [2] [3] [4] can take advantage of the asset location information. In order to grant such functionality, JLocator offers an API that allows external remote applications to gain access to the asset location information database.

[2] The minimal asset granularity is limited to computer equipped with network boards. Dialup computers or palm devices are outside the scope of this work because they can be either located using authentication protocols such as Radius or because they have no network capabilities but rely on a host computer trackable by JLocator.

3. Inside Asset Location

Asset location is a very complex task and in general, it can be performed only if the underlying network provides some support. For instance in LANs based on 10Base2 ethernet or token ring it is not possible to know from the programming point of view whether two hosts are physically adjacent[3], hence their physical positions. Instead, point-to-point networks such as ATM (Asynchronous Transfer Mode) [14] or 10Base/T ethernet allow hosts to be physically located. In fact, once the physical location of network connector sockets is known[4] and stored in a database, it is possible to calculate which host is attached to a certain connector and thus to know the host location. Please note that if the underlying network is point-to-point, it is possible to locate hosts independently of the network protocol the hosts are using. This is because hosts are identified with a unique network address and not with a protocol specific address (e.g. IP address). However, because IP is by far the most used protocol, JLocator limits its scope only to IP and it does not discover information related to protocols other than IP.
The host location algorithm used by JLocator relies on the MAC (Media Access Control) address. The MAC address is the hardware address of a device attached to a network. It differs for various physical media but the network board manufacturer guarantees its uniqueness[5]. This property allows a computer to be uniquely identified through the MAC address independently of its IP address. Some hosts implement single link multi-homing, a mechanism that allows multiple IP addresses to be associated with the same hardware interface. This means that the association MAC address/IP address is not 1:1 but 1:N. This is not a problem for JLocator because the uniqueness of the MAC address is still guaranteed.
In the Internet world the ARP (Address Resolution Protocol) [15] protocol is used to dynamically correlate MAC and IP addresses. ARP is very important because network applications deal uniquely with IP addresses whereas the IP stack must deal with MAC addresses also. In fact, if a host has to communicate with another host, either the MAC address of the destination (if both hosts belong to the same IP subnet) or the router (if the hosts belong to different IP subnets) must be know. RFC 1157 [8] defines the `atTable`object that allows, through SNMP, cached ARP tables to be retrieved. Each `atTable`entry associates an IP address with its corresponding MAC address. Consider now figure [1]. Whenever two or more subnets/networks are interconnected, it is necessary to have a gateway/router for routing packets. Owing to the way IP works, the gateway is informed about MAC addresses of both subnets, therefore it is possible to conclude that the union of all the gateways/routers of a network contains the association IP address/MAC address for every hosts of an organisation. The drawback of the usage of ARP comes from the following facts:

- networked hosts that have never communicated with other peers are not present in the cache;
- cached entries expire, hence a host that has not communicated for a while may have its corresponding entry missing from the ARP table.

If the use of an ARP cache is considered unnecessary, JLocator provides a simple solution to the problem of refreshing the gateway's cache. Before walking the `atTable`table, a host (usually the one where JLocator runs) pings[6] all[7] the hosts that belong to the known networks. This operation is done in order to wake up all the hosts and hence propagate their MAC address and consequently refresh the ARP table. In addition to this mechanism, JLocator also keeps a persistent MAC address cache that is period-

3. Using some low level tools it is possible to discover hosts adjacency on token ring networks. However, this information is useless for the purpose of asset location because unused network sockets might have been short circuited hence two logically adjacent hosts might not be adjacent with respect to their sockets.

4. Please note that network administrators have to store the physical socket location into the database because they are the only ones who know where a certain socket is physically located inside a building.

5. Please note that in some peculiar cases the MAC address can be changed via software for addressing specific problems (e.g. hardware swap).

ically refreshed.
Once the association MAC address/IP address is known, it is necessary to associate the MAC address with the physical location of the network element identified by it. Modern hubs/switches contain a SNMP agent that implements the Repeater MIB [10]. Such MIB contains the *rptrAddrTrackPackage* package that associates a MAC address with a hub/switch port [8]. The intersection of the information retrieved from the *atTable* and *rptrAddrTrackPackage* (the correlation key is obviously the MAC address) allows a host -for instance its symbolic name- and port in the hub/switch/router to be uniquely associated. Once the position of a cable coming from a specified port is known (for instance the network administrator provided it), it is possible to (approximately) know where a host is located at a certain point in time because it is known to where the network socket is connected. This algorithm is used by JLocator to locate assets.

4. JLocator Architecture

An asset domain is a set of asset resources grouped using a logical relation. For instance, Finsiel's assets can be represented as an asset domain divided into further domains according to the site where the assets are physically located. Domain identification criteria are not unique among different organisations and their identification is usually left to asset managers or network administrators although in general domain granularity is much larger than an IP subnet. An instance of JLocator can control one or more asset domains. Each instance is responsible for managing asset information concerning the controlled domain(s). Different instances register each other in order to create a web of information instead of centralising the information on a single site.
JLocator allows computing assets to be located, listed, and managed. In particular, JLocator is able to discover network resources including -but not limited to- computers, hubs, and routers running the IP protocol. Asset information is stored in a relational database that is updated on operator's request or automatically according to a specified policy. Access to asset information is performed through a web interface. Such interface is enriched with applets whenever it is necessary to give the user interactive asset information access.

The architecture of JLocator is based on the client-server paradigm, where the client is a web browser and the server is JLocator itself. Multiple clients can connect concurrently to the same server. Asset information is fetched from the network by means of the SNMP (Simple Network Management Protocol) protocol [9] and then stored in a relational database. JLocator's main task is to collect asset location information, store it into the database, and serve client requests. Clients connect to the HTTPd (HTTP server) that returns HTML pages representing asset information, some of which containing Java applets part of JLocator. Those pages allow clients to navigate through the asset information, configure JLocator facilities and modify the way asset information is retrieved and displayed. Java applets transparently talk to JLocator by means of the Java RMI (Remote Method Invocation) [16] protocol, whereas there is no direct communication between JLocator and HTTPd. Using a web browser, system administrators can:

---

6. Ping is a tool for network testing, measurement and management based on the ICMP (Internet Control Message Protocol) protocol [7]. Using ICMP's *ECHO_REQUEST* datagram, it elicits an ICMP *ECHO_RESPONSE* from a host or a gateway. This allows network connectivity to be checked.

7. The list of the hosts to ping is calculated using the network address and the network netmask, contained in the gateway routing table. This table is read by JLocator via SNMP.

8. In the case of a hub switch or of multiple cascading hubs, the situation is slightly more complicated. If a set of hosts is attached to a switch port (for instance through a hub/switch) the IP address that corresponds to a certain port is not relevant. This is because such address does not identify a host/hub but a (sub) network.

- modify the policy for the collection of asset information;
- configure the hubs/switches to which assets are connected;
- use an asset editor that allows them to specify where network sockets are located hence the places where assets can be potentially located;
- locate assets using various criteria.

JLocator communicates with the asset database by means of the ODBC (Open DataBase Connectivity) protocol via the Java JDBC interface. Database information includes (but is not limited to) physical location, connectivity information (how the resource is attached to the network) and management information (SNMP), if any. Inventory information is stored in a different database accessed using various protocols such as ODBC or other database vendor-specific protocols[9]. The reason for keeping asset information separated from inventory information resides in its different nature. Asset information is fetched from the network using SNMP and other management protocols whereas inventory information is partially computed by investigating the asset itself and partially assigned by asset administrators. For instance, consider a host attached to a network. Host inventory information includes operating system and hardware type. This information can usually be retrieved from the network using specific protocols. Other information such as the estimated host value (in dollars), inventory ID and name of the current owner need to be provided by human operators. Asset information is rather dynamic especially when most of the assets are portable computers that move from one location to another, whereas inventory information is mostly static.

In order to avoid non-authorised users to perform administrative tasks, HTML pages and applets used for administration are stored in a protected area. If asset information is particularly sensitive, the HTTPd can be configured in a way that instead of using plain HTTP, S-HTTP (Secure HTTP) can be used.

5. JLocator Configuration

JLocator configuration is the only activity where human intervention is required. This is because asset administrators specify where network connectors are located inside the buildings hence the places where assets can be placed. In addition, administrators need to tune some additional parameters such as the hub/switches addresses and the policy used for reading asset information from the network.
As stated before, the only way to interact with JLocator is by means of a web browser. JLocator's main HTML page contains the links to the various components as well as the link to the asset search engine.

The preferences panel allows administrator to specify various JLocator parameters including:

- Database
  Administrators can specify the database entries lifetime (expressed in days). Each entry corresponds to an asset. Whenever an asset is detached from the network for more than X days, where X is the lifetime, the asset is considered no longer active. This check is necessary because some assets, although they are off and unused, are still attached to a network socket. Please note that some hosts, for instance portable machines, can be used while unplugged from the network. Due to this, administrators can selectively set the lifetime of such hosts to infinite in order to avoid JLocator to remove from the asset database after the specified lifetime.
- Network Scan
  Whenever asset information need to be refreshed, JLocator scans the specified network devices.

9. Currently JLocator has been integrated with [24] that allows external applications to access inventory information via ODBC.

The scan process, performed in background, takes some time and it updates the asset database. Thus is important to perform the network scan only when it is necessary. Administrators can scan the network manually or set JLocator to automatically scan the network at selected times, for instance every night.

In order to simplify the network configuration, JLocator comes with an application that automatically locates gateways/hubs. The algorithm used is the following. Network exploration starts the host where JLocator runs. Then the local subnet is explored. Whenever a gateway is located [10], all the subnets attached to it are recursively explored until all the subnets are explored. In order to limit the scope of the exploration, users can specify further parameters such as maximum number of hops from the initial subnet.

Once the network configuration has been specified, asset discovery can be started. While discovery is in progress, asset administrators can specify the structure of the sites where assets can be placed.

In order to enable JLocator to depict assets where they are actually located, administrators need to specify the structure of the sites where the asset can be located and the position of the network sockets. This operation is performed interactively by using the building configuration editor. The editor is used to:

- define the hierarchical structure of the sites where assets can be located;
- identify by means of hyperlinks how sites/buildings/rooms are physically interconnected.

Editor's main window is split in two parts: the containment tree and the containment editor. The containment tree defines how the information is hierarchically structured. For instance, a containment hierarchy is the following: a building contains floors, a floor contains rooms, and a room contains desks where computers sit. Each containment entry has some containment links towards the contained elements. The previous picture contains a simple hierarchy. The root node contains three nodes (Pisa, Rome and Naples) that correspond to three sites. Each site is divided into various floors and each floor contains several rooms. Containment hierarchy can be modified adding/removing nodes to/from the tree. Each node is mapped to an HTML page. Nodes can be leaves (rooms) or intermediate nodes (buildings for instance). Each node has a picture that represents it and is used as background map in the corresponding HTML page.

Once the containment hierarchy is created, links connecting the various nodes can be added. Links are either rectangle or circle shaped and they always link the current selected node with one of its direct children, i.e. a child one level below the selected node. In figure [4], the possible links for the Root node are Root–Pisa, Root-Rome and Root-Naples. The links and the background image are represented as sensitive HTML areas and mapped to the `IMAGEMAP`tag. When all the links have been added, the editor generates the HTML code corresponding to the specified hierarchy and place it in a directory accessible by JLocator.

Once buildings have been configured, it is necessary to specify where the assets can be located inside the rooms. Each room can contain one or more network sockets to which the assets are connected. Administrators need to specify where the sockets (hence the assets) are located in each room. Following the HTML links of the pages generated by the building configuration editor, administrators can place the network sockets to which assets are connected. This operation is performed interactively using a JLocator component called room editor applet. For each room, this applet shows the room image selected using the building editor and allows administrators to both place sockets and change some room characteristics.

---

[10] A gateway has the MIB II *ipForwarding* variable set to forwarding, whereas IP hosts have set it to *non-forwarding*. In addition a gateway and more than one interface -not comprising the loopback interface- to use for routing purposes.

Sockets can be added in two ways either ‘by socket name’ or ‘by example’. In the first case, the editor shows a window that contains the list of unplaced sockets sorted according to hub/group/port. In the other case, the editor shows the list of hosts that have been identified during the network discovery. By selecting one of these hosts, JLocator associates the network socket to which the selected host is currently attached with the room being edited. These two ways to place hosts have been conceived in order to make configuration task easier. In fact administrators who:

- have access to a socket map may prefer to place sockets using the socket name;
- do not have a socket map but just a host map, can easily place sockets by using the place ‘by example’ method. In other words using the second option, administrators instruct JLocator to place the socket to which the host is attached.

The socket placement was the last configuration step. JLocator has stored inside its database the location of the network sockets. Whenever the network is scanned, JLocator associates a hub/switch port with the corresponding socket. If a host changes the network socket to which its attached then it is automatically placed in the new location. For each host/room, JLocator keeps a history of movements that might be useful for administrator in order to track the position of an asset over the time.

6. JLocator at Work

JLocator’s main goal is to keep track of computing assets. From the user’s point of view, JLocator is able to answer two basic questions i.e. where the asset X is physically located, and what assets are contained in the place Y. JLocator’s search engine gives an answer to the first question. Navigation through the HTML pages to the second. These two orthogonal methods of asset navigation/location give the user the best way to locate assets depending on the input data. If the asset name is known, the search engine is the best choice. On the other hand, HTML navigation is used whenever it is necessary to know what assets are located at a given location.

The asset location applet, part of JLocator, provides a graphical user interface to the search engine. It allows users to locate an asset given its symbolic name, either symbolic or numeric IP address, or by means of other criteria such as department, computer type or serial number. Once the search criteria has been specified, the JLocator is contacted and the selected host is searched through the local database. In case the asset has not been successfully located or if it is currently detached from the network, an error message is prompted to the user. Please note that in case the host is not known locally, JLocator broadcasts the request to all the other JLocator instances running in distant sites [11]. In case the searched asset is successfully located, the web browser will automatically load a page showing the room where the asset is contained. If a remote JLocator server has located the asset, the page is retrieved directly by such server to whom all the future requests will be directed. The room is represented using the room editor applet started in read-only mode. The applet retrieves information about assets by contacting the JLocator server. Assets that have changed their position since the last network scan are highlighted. Users can access further information about the asset by clicking on the icons that represent them. If a SNMP agent runs on the asset, the room applet allows users to explore it using a generic SNMP browser. In case the asset runs a SNMP agent that implements further MIBs such as Finsiel’s proprietary MIBs [11], users can take advantage of a more specialised browser called JSugar as shown in figure [8].

JSugar is a user-extensible SNMP browser based on software components [19]. Each component handles a part of the MIB and is activated on demand if the host being explored supports that part of the MIB. JLocator provides an API that allows developers to create new JSugar components that can be

[11] The host addresses where the servers run are registered in a configuration file read by JLocator at start-up time. Future JLocator versions will likely be based on JavaGroups [23] that greatly simplifies group communication.

added to default ones. This facility makes the browser extensible by the final users without the need to have access to JLocator's source code.
External applications can have access to all information concerning asset location. JLocator provides two ways to access this information either by querying directly the database or remotely through some RMI-based Java interfaces.

7. Implementation Issues

Currently JLocator is used internally by Finsiel to track company assets. It has been installed in several sites and it controls several thousand of hosts. Because JLocator's location algorithm is based on the MAC address, JLocator can also keep track of:

- mobile hosts [5] [6] that are dynamically attached/detached to/from the network;
- hosts that implement single link multi-homing;
- hosts that dynamically change IP address (for instance using the DHCP [12] protocol).

The only drawback of this design choice is the assumption that the host MAC address never changes. Although this is a very rare situation, it might be that the network card of a host is replaced. In order to cover this situations, JLocator allows administrators to change the host's MAC address contained into the database. However, supposing that the administrator will not update the MAC address, JLocator will automatically locate the host with the new network board and put aside the old host. That old entry will be transparently purged from the database when the specified lifetime is elapsed.

The first JLocator prototype was based on JMAPI [18], Sun's network management API. However the current version is JMAPI-free because:

- JMAPI management architecture was overly complex and heavy for relatively slim applications such as JLocator;
- JMAPI proprietary user interface components have been replaced with standards JFC (Java Foundation Classes) [17] beans now part of Java JDK.

JLocator shows that Java can be successfully used to create distributed management applications. In addition, it demonstrates that network management systems can take advantage of Java. In particular:

- RMI has greatly simplified the server-server and server-web browser communications;
- JDBC made the server-database communication easy and independent of the database;
- Java interfaces allow dynamically loaded software components to be developed by independent developers without the need to have access to JLocator's source code;
- by exploiting Java built-in multithread support, JLocator can serve remote requests while updating the database and scanning the network devices;
- testing and debugging has been greatly simplified because problems typical of C/C++ based programs such as memory leaks or core dumps do not show up with Java.

As described previously, JLocator is able to operate if the following requirements are satisfied:

- the network type is point-to-point (for instance 10-100Base/T ethernet and ATM);
- network hub/switches have on-board a running SNMP agent implementing the Repeater MIB;
- gateways run a SNMP agent implementing the SNMP MIB-II [8].

Although these requirements may appear as limitations, it is worth noting that all of them are usually

satisfied in modern networks.

8. Final Remarks

In conclusion, JLocator is the first software-only system designed for asset location. Assets are tracked independently of their operating system or network protocol being used. It has been successfully used in large organisations with both static and dynamic host addresses. The use of Java and of the web interface makes it easy to use and administer from remote and it does not require any software to be installed in the client side. This demonstrates that Java and the web are mature technologies that can profitably replace established technologies based on C/C++ even on conservative fields such as network management.

9. References

[1] K. Arnold and J. Gosling, *The Java Programming Language*, ISBN 0-201-63455-4, Addison-Wesley, 1996.

[2] Hewlett-Packard, *HP AssetView: User Manual*, 1996.

[3] Apsylog S.A., *Asset Manager White Paper*, http://www.apsylog.fr/whitbook.htm, 1996.

[4] Microsoft Corporation, *System Management Server 1.2: User Guide*, 1997.

[5] A. Dixit, V. Gupta and B. Lancki, *Mobile-IP for Linux*, Dept. of Computer Science, State University of New York, November 1995.

[6] A. Bakre and B.R. Badrianath, *I-TCP: Indirect TCP for Mobile Hosts*, Technical report DCS-TR-314, Rutgers University, October 1994.

[7] J. Postel, *Internet Control Message Protocol*, RFC 792, September 1981.

[8] K. McCloghrie and M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets: MIB-II*, RFC 1213, March 1991.

[9] J. Case, M. Fedor, M. Schoffstall and C. Davin, *Simple Network Management Protocol (SNMP)*, RFC 1157, May 1990.

[10] D. McMaster, K. McCloghrie, *Definition of Managed Devices for IEEE 802.3 Repeater Devices*, RFC 1516, September 1993.

[11] Finsiel S.p.A., *Sugar Toolkit v.2.2: User's Guide*, 1996.

[12] R. Droms, *Dynamic Host Configuration Protocol*, RFC 1531, October 1993.

[13] B. Croft and J. Gilmore, *Bootstrap Protocol (BOOTP)*, RFC 951, September 1985.

[14] R. Handel, M. Huber and S. Schroder, *ATM Networks: Concepts, Protocols, Applications*, 2nd edition, ISBN 0-201-42274-3, Addison Wesley, 1994.

[15] D.C. Plummer, *An Ethernet Address Resolution Protocol*, RFC 826, November 1982.

[16] Sun Microsystems, *Java Remote Method Invocation: White Paper*, November 1997.

[17] Sun Microsystems, *Java Foundation Classes: Now and the Future*, White Paper, 1997.

[18] Sun Microsystems, *Java Management API: Programmer's Guide*, May 1997.

[19] O. Nierstrasz, S. Gibbs and D. Tsichritzis, *Component-Oriented Software Development*, Communications of the ACM, 35(9), September 1992.

[20] Hewlett-Packard Corporation, *HP OpenView Distributed Management: Platform Developer's Kit*,

October 1996.

[21] R. Strurm, *Working with Unicenter TNG*, ISBN 0-7897-1765-4, August 1998.

[22] D. Plummer, *An Ethernet Address Resolution Protocol*, RFC 826, November 1982.

[23] Bela Ban, *Design and Implementation of a Reliable Group Communication Toolkit for Java*, Cornell University, September 1998.

[24] Hewlett-Packard Corporation, *HP OpenView Desktop Administrator: User's Guide*, 1998.

10. List of Figures

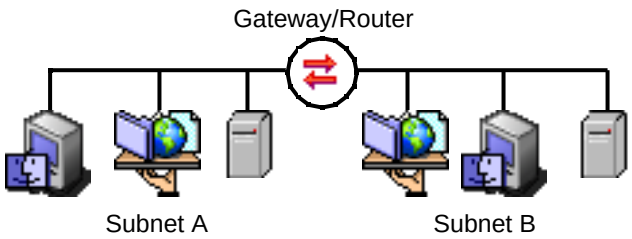

Figure 1 - IP Subnet Interconection

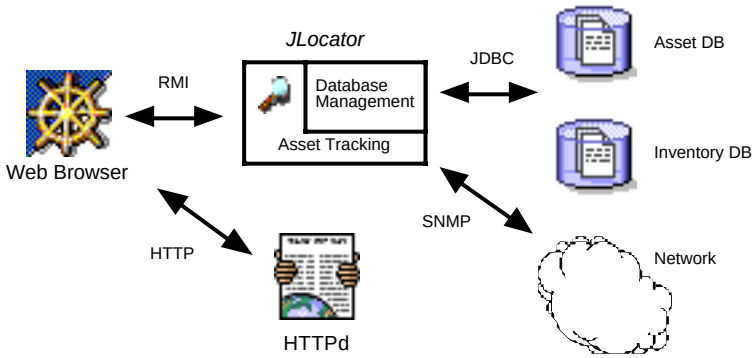

Figure 2 - JLocator Architecture

Figure 3 - JLocator Entry Page

Figure 4 - Building Configuration Editor

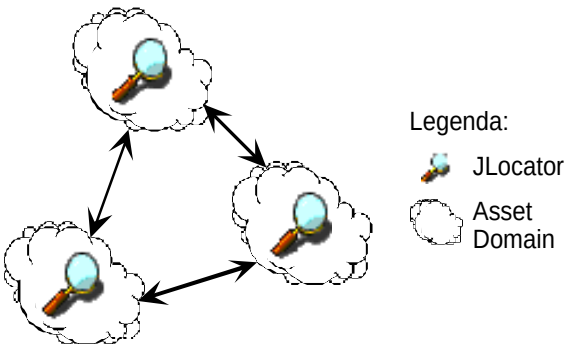

Figure 5 - Assets vs. JLocator

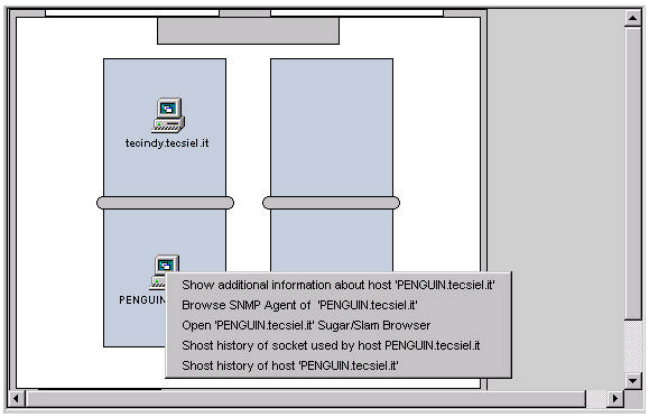

Figure 6 - Room Editor Applet

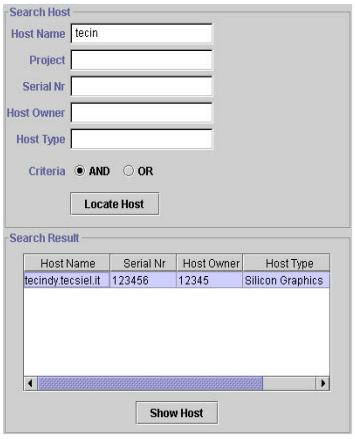

Figure 7 - Asset Search Engine

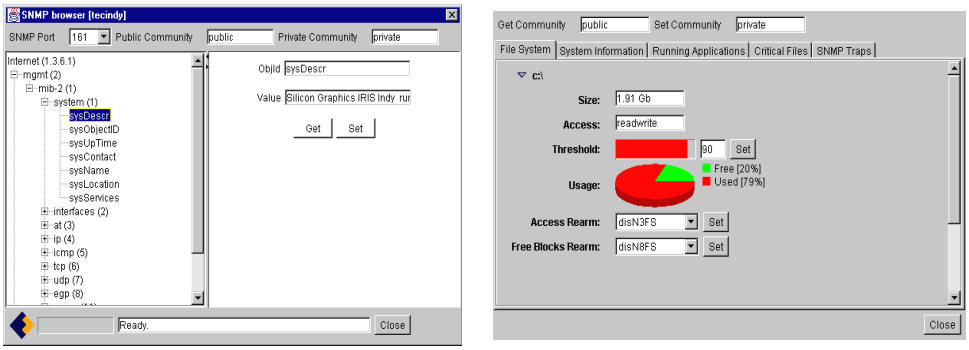

Figure 8 - Room Applet: Generic and Custom SNMP Browser