JLOCATOR: A WEB-BASED ASSET LOCATION SYSTEM

Luca Deri
Finsiel S.p.A.[1]
Centro SERRA[2]*, University of Pisa*
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.
This paper covers the design and implementation of JLocator, a Java-based system that allows assets to be dynamically localised. Users can locate assets through a web interface, and external applications such as asset management systems can take advantage of asset location information provided by JLocator. Finally, JLocator's 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 needs to change its IP address. The consequence is that management applications are usually not able to physically locate a network resource and cannot access services provided by mobile computers when their IP addresses change.
Asset management systems allow 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] provide 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this half-automatic asset management procedure, it can happen that the asset database contains a mixture of fresh and outdated information that may confuse operators.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. 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.

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 network.

1. Finsiel S.p.A., Via Matteucci 34/B, 56124 Pisa, Italy, l.deri@finsiel.it.
2. Centro SERRA, Lungarno Pacinotti 43, 56100 Pisa, Italy, deri@unipi.it, http://jake.unipi.it/~deri/.

- 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:

- a SNMP agent must run on each asset;
- in order to retrieve asset positions, assets should be active because the SNMP agent running on the assets needs to reply to queries;
- network administrators need to keep *sysLocation* updated whenever the asset changes position.

Because such update process is not automated, it is clearly possible that *sysLocation* contains an outdated value with respect to the actual asset position. Due to this limitation, the use of *sysLocation* is not satisfactory because this solution still relies on human operators. Some companies offer expensive asset location systems 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 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 bus Ethernet (10-100Base/2) or token ring it is not possible to know from the programming point of view whether two hosts are physically adjacent, hence their physical positions. Instead, point-to-point networks such as ATM (Asynchronous Transfer Mode) [13] or 10Base/T Ethernet allow JLocator to localise hosts. In fact, once the physical location of network connectors is known[3] 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. 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 algorithm used by JLocator to locate hosts 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. 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 allow 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.
In the Internet world the ARP (Address Resolution Protocol) [14] protocol is used to dynamically map correlate MAC and IP addresses. RFC 1157 [8] defines the object type `atTable`that allows, through SNMP, cached ARP tables to be retrieved. Each `atTable`entry associates an IP address with its corresponding MAC address. Consider now the following picture.

3. 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.

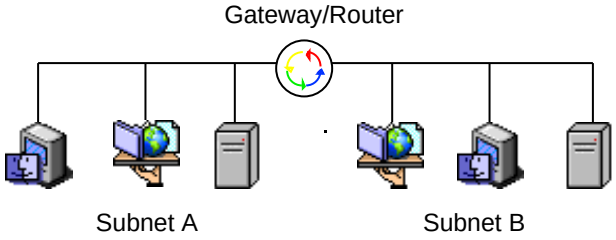

Figure 1 - IP Subnet Interconection

Whenever two or more subnets/networks are interconnected, it is necessary to put a gateway/router in the middle to route packets properly. 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 a 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.

A way to refresh the cache and make sure that all the active host entries are present in the cache is the following. Before walking the ARP table, a host (usually the one where JLocator runs) pings[4] all[5] 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 periodically refreshed.
When 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 10Base/T 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[6]. 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. This is the algorithm used by JLocator to locate assets.

3 JLocator Design Goals

JLocator is a Java-based [1] system 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 managing assets and have lim-

4. Ping is a tool for network testing, measurement and management based on the ICMP (Internet Control Message Protocol) protocol. 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.
5. 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.
6. In the case of a hub switch or of multiple cascading hubs, the situation is slightly more complicated. In case 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.

ited computing experience. Due to this, JLocator should offer a friendly user interface, usable by non-computer experts, that depicts the assets where they really are located and hides 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 should be able to track every network asset independently of its type.
- Minimal impact on performance, resource utilisation and network traffic.
  Following the same principle adopted by SNMP, JLocator should have 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 allows the creation of a web of systems capable to locate assets on multiple locations.
  Large organisations have different sites and offices. JLocator must be able to keep track of all the assets. 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 that collaborate to perform global asset tracking.
- Web-based user interface that allows users to easily locate assets.
  JLocator should have a simple user interface that has minimal impact on client hosts. Because most of the users have a web browser installed, using JLocator from a web browser has the following advantages:
  - there is no need to install any software on client machines;
  - web user interface is well known and does not require any additional training;
  - JLocator's updates do not require any installation on the client hosts 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 should be performed fully automatically in order to avoid dependency on human operators responsible to update asset information. This avoids synchronisation problems regarding asset information and guarantees that asset information is constantly updated as assets change position. In addition, asset tracking systems should not rely on specialised network in order 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.

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 subnetwork.

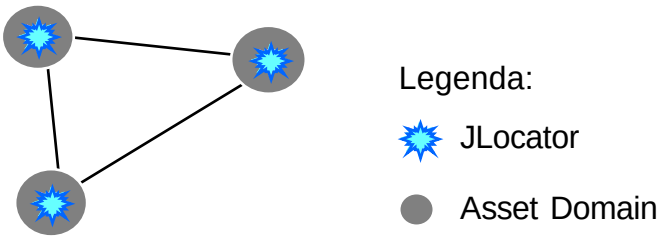

Figure 2 - Assets vs. JLocator

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. The advantages of this solution are:

- Sensitive information is not spread around but it is kept locally in order to restrict access only to authorised users.
- There is no need to keep JLocator instances synchronised because there is no asset information flowing among the various instances.
- Scalability: as soon as the number of asset to track grows, administrators can decide to split the assets in further asset domains and activate additional JLocator servers for those domains.
- The architecture is very similar to the web architecture making it simple to exploit web tools to wander around assets.

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 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.

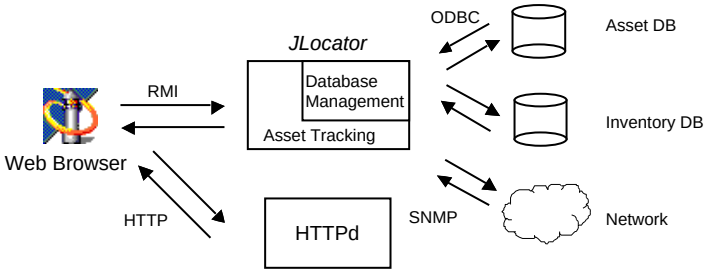

Figure 3 - JLocator Architecture

Asset information is fetched from the network by means of the SNMP (Simple Network Management Protocol) protocol [9] and then stored in a persistent 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) [15] protocol, whereas there is no direct communication between JLocator and HTTPd. Using a web browser, administrators can:

- change the way asset information is collected;

- configure the IP subnets to be explored;
- use an asset editor that allows them to create HTML pages, used for asset navigation, that depict 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. Information stored into the asset database includes (but is not limited to) physical location, connectivity information (how the resource is attached to the network) and management information (SNMP), if present. Inventory information is stored in a different database accessed using various protocols such as ODBC or other database vendor-specific protocols. 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 used for administration are stored in a protected area. The consequence of this is that whenever a user attempts to access such pages, the web browser prompts a dialog where the user has to specify the login and the password. This is in order to limit the access only to authorised users (usually the administrators). Instead, non-privileged users can only navigate through the assets that are meant to be public. User authentication is performed using basic HTTP authentication. In case the 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's main HTML page contains the links to the various components as well as the link to the asset search engine.

Figure 4 - JLocator's Main Page

The preference panel allows administrator to specify various JLocator parameters:

- Database
  Administrators can specify the lifetime (expressed in days) of database entries. 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 them by mistake from the asset database.

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

As far as the network configuration is concerned, JLocator is able to automatically locate gateways/hubs. The algorithm used is the following. The network exploration starts from the local subnet. Once a gateway is located[7], all the subnets attached to it are recursive explored until all the subnets are explored[8].

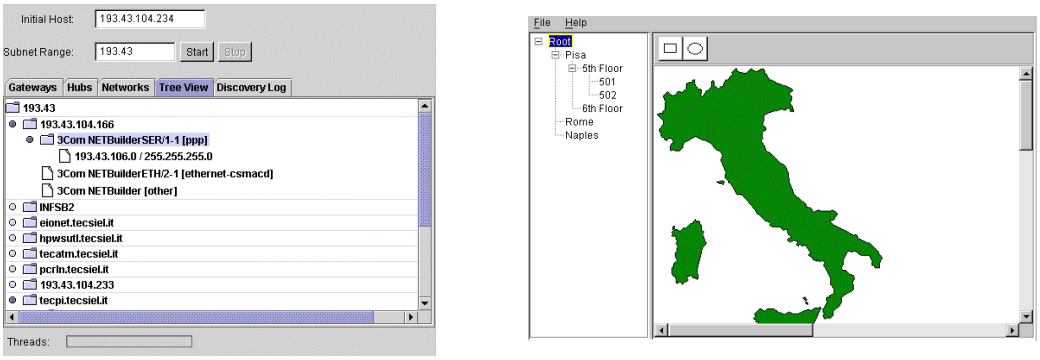

Figure 5 - Network Discovery and Building Configuration Editor Applet

In order to represent the asset location properly, 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 allows administrators to define the asset information structure by defining and linking together the various places where such asset can reside. For instance, a way to structure asset information is by dividing it according to the geographical location and other criteria such as company, building, and floor. 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 that is used as background map in the corresponding HTML page.
Once the containment hierarchy is created, links among 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 3, 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 HTML code is generated and placed automatically in a place that can be accessed from

7. 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.
8. Whenever it is necessary to limit the scope of the exploration, some further parameters (for instance max. number of hops) can be specified.

JLocator's main page.

After the building configuration, it is necessary to specify where the assets can be located inside the rooms. Each room contains one or more network sockets to which the assets can be 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 access and edit rooms and the network sockets to which assets are attached. This edit step is performed using a JLocator component called room editor applet. Sockets can be added in two ways either 'by socket name' or 'by example'. This flexibility allows administrators to select the host to which the socket (to place) is attached. 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.

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 one. 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 network name, either symbolic or numeric IP address. Once the user has specified the name, the JLocator server 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[9]. 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 an applet that depicts the assets where they really are. The applet retrieves the 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 Sugar/Slam [11], users can take

9. The host addresses where the servers run are registered in a configuration file read by JLocator at start-up time.

advantage of a more specialised browser as shown in the following figure.

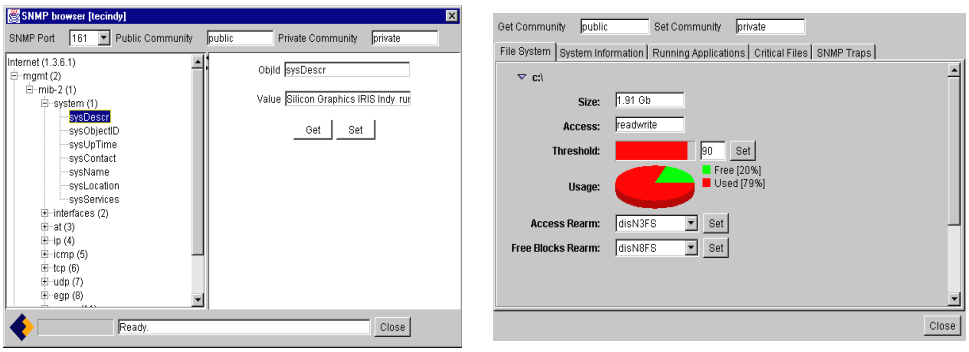

Figure 6 - Room Applet: Generic and Custom SNMP Browser

The custom browser is based on software components [18] that are loaded on demand and make it extensible. JLocator provides an API that allows developers to create new components that can be added to the 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. The use of these interfaces allows JLocator users to develop new applets/applications that can access remote JLocator servers.

7 Final Remarks

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] and of hosts that dynamically get their IP address from a server (for instance using the DHCP [12] protocol).
The first JLocator prototype was based on JMAPI [17], 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) [16] 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;
- 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 a running SNMP agent implementing the Repeater MIB;
- gateways have running a SNMP agent implementing the SNMP standard MIB.

Although these requirements may appear as limitations, it is worth noting that all of them are usually satisfied in modern networks.

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++.

8 Acknowledgments

The author acknowledges R. Burlon (r.burlon@finsiel.it) for his studies about network localisation algorithms that have been adopted by JLocator. In addition, he would like to thank D. Manikis (Jimmy.Manikis@telecom.ece.ntua.gr), E. Mattei (e.mattei@finsiel.it) and M. Robertini (robertini@sns.it) for their suggestions and valuable discussions during the design and development of JLocator.

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] S. Deering, *ICMP Router Discovery Messages*, RFC 1256, September 1991.

[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, *Interoperation between DHCP and BOOTP*, October 1993.

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

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

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

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

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

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