JIT '99, September 20-21, 1999, Duesseldorf, Germany

A Jini-based Gateway Architecture for Mobile Devices

Gerd Aschemann, Roger Kehr, and Andreas Zeidler

Darmstadt University of Technology, Department of Computer Science
{aschemann,kehr,az}@informatik.tu-darmstadt.de

Abstract. In the near future we expect a widespread deployment of mobile computational resources including network-enabled end devices like Laptops and PDAs. An interesting problem then and today is the transparent change of locality. Traditionally several manual actions are necessary to reconfigure the device and to rebind client-applications to services available in the host network. This paper presents the architecture of a Jini-based Application Level Gateway (ALGW) which avoids manual reconfiguration of a mobile device every time the user changes the host network. To do so, the ALGW makes use of key technologies provided by Jini. Jini is used for dynamic looking up and binding to services needed by the user and located in the host network. Moreover, Jini can be used for value-added services like our authenticating SMTP-Service.

1 Introduction

There is a rapidly growing market for intelligent mobile devices. Devices such as Laptops and Personal Digital Assistants (PDAs) are getting cheaper, smaller, and more powerful. We expect a fast and wide proliferation of intelligent mobile devices within the next few years.

Coupled to this development is a changing *pattern of usage* [13]. The user of a mobile device – in contrast to the user of a desktop system – is moving geographically, connecting the device often to different local networks in order to fulfill different tasks. A salesperson, for example, needs to connect to the Internet browsing for important information or use email for reporting the sales of the day to the company's sales department. To do so, he or she can dial up an Internet Service Provider (ISP) via modem but is hinged to the availability of a particular ISP at the current location. More convenient would be to connect directly to an available Local Area Network (LAN), e.g., of a customer or a hotel. The same solution is desirable for people often changing between a fixed number of different environments, e.g., the office and the home network.

Today, however, the change of network environments is linked to a number of *reconfiguration* tasks to be done manually (Fig. 1). There is an obvious *gap of transparency* between the need to change the network environments on the one hand and the methods to support this change in a transparent and convenient manner for the user on the other hand. One has to change various configuration

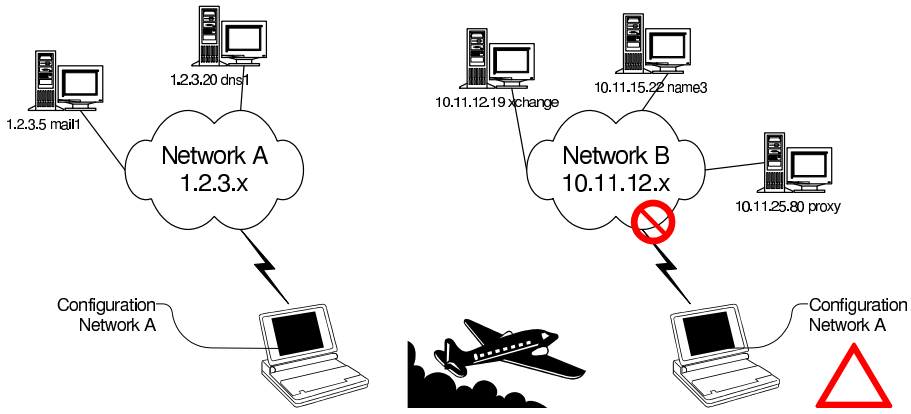

Fig. 1. Connecting to different networks

parameters such as the own IP address, the name of a local mailserver, or the WWW-proxy configuration in the web-browser. Moreover, in a public accessible network environment there usually exists a level of distrust between the user on the one hand and the provider of the local network on the other hand. Both have an interest in security and privacy of the data and the integrity of the services they use or provide. A scheme of authentication and legitimation is needed.

To alleviate the lack of flexibility and transparency of reconfiguration for the user when changing the network environment, we propose a Jini [17] based gateway architecture preventing the user from reconfiguring his or her mobile device. To do so we are using *Application Level Gateways* and Jini as enabling infrastructure. In short, an application level gateway is a mediator between a mobile device and the local network and its available services. The purpose is to provide a mechanism which allows the user to use a single static configuration for every network. The application level gateway takes care of mapping services needed locally to services available in the network. To ensure transparency and security for the user, we employ some key features of Jini which will be introduced in greater detail throughout the next section.

The overall architecture, including requirements and solutions, is described in Sect. 3, while Sect. 4 gives an example of an enhanced email service using our proposal. Related work can be found in Sect. 5, and we finish with a conclusion and description of future work.

2 Key Concepts of Jini

Jini is a recently released service-based network infrastructure from Sun Microsystems. Jini is deliverd in the form of a Java API and relies on features

available in the Java 2 platform only. It offers components we use in our architecture to enable our application level gateway to participate in a LAN without reconfiguration of the mobile device by the user. We give a short introduction of the key building-blocks of Jini.

Bootstrapping. Jini has a built-in bootstrapping mechanism called *Jini Discovery and Join Protocol* [18] based on multicast and unicast-communication. This set of protocols enables arbitrary Jini-enabled objects, i.e., hardware devices as well as software, to find the *Jini Lookup Service (LUS)* [19] and register their services and properties at the local *federation* of services. This allows for "spontaneous" networking of network components to find each other in a standardized manner.

Lookup Service. A Lookup Service acts as a central repository for services. Appropriate means are offered for clients to query and select registered services based on Java *interfaces* – describing the type of the service – and so-called *entries* – descriptive information or additional state information about a service in the form of serialized Java objects.

Additionally, clients can register interest in state changes of the lookup service – e.g., when a new service of a certain type registers or deregisters itself – to receive event notifications via remote listener objects.

Proxy Objects. Services upload serialized Java objects, called *service proxies*, to the lookup service. These objects can be downloaded to any client Java Virtual Machine (JVM) and invoked to access the service. The proxy acts as a mediator to the service itself, and may implement programmatic interfaces as well as graphical user front-ends for the service. The proxy encapsulates any protocol used for the actual communication between the proxy object and the service. This is a key feature when considering secure communication as we will see later.

Leases. Leases are time-based contracts between two objects within Jini. A lease grantor can bind a service to a lease holder for a certain amount of time. The lease holder can use the granted service within this period according to the contract made, but has to renew the interest for the service granted, i.e., renew the lease before it expires. Failing to do so automatically cancels the contract. Utilized appropriately we use leases for the detection of changes, e.g., the change of the network environment.

3 Architecture

The basic idea of our approach is to provide a generic application level gateway for all or most of the services a mobile device typically uses, e.g., WWW, printing, mail/pop etc. To put it simply, the application level gateway provides all the

services locally, by installing a respective proxy as a server. If the service is used, the gateway has to care for a transparent hand over to a real service in the host network. However, this – at first sight – simple approach leads to some new problems:

1. **Finding the real services.** The application level gateway has to find real servers on the host network, sometimes to choose an appropriate one if there are more than one and to establish a connection to it.
2. **Transfer of data and mapping of protocols.** The gateway has to transfer the data back and forth between the real server and the application. Sometimes the client and the server will not use the same protocol, even if the provided service is the same in an abstract sense, e.g., UNIX systems and MS Windows Systems usually access printing services with different protocols. Therefore a mapping between the different protocols must be performed.
3. **Detection of locality changes.** A change of locality, i.e., a change of the host network, must be detected and appropriate action must be performed, e.g., the respective servers on the new host network must be found and rebinding or reconnection must take place.
4. **Reconfiguration of lower level services.** Some of the network changes are beyond the scope of the application level gateway, i.e., basic IP connectivity and configuration of basic IP services, e.g., Naming Service (DNS).

Furthermore we want to achieve two additional purposes:

- Our solution aims at the reuse of most client applications and their server counterparts without any implementation changes. The client applications are configured once to use the application level gateway on the local host as a proxy server and talk to it with their respective protocol. Reuse of legacy server implementations and even existing proxy applications can be obtained by another mechanism we will describe further down (see Sect. 3.2).
- We want to enable value added service implementations in some cases, like making a service fault tolerant or making a service and its usage more secure in terms of authentication, encryption, privacy protection or non-repudiation. In our example (see Sect. 4) we outline such a value added service implementation.

3.1 The Jini-based Approach

Fortunately, with Jini we have a valuable technology to solve these problems. Figure 2 shows the basic architecture of the gateway and how it provides access to the services of the host network to local client applications. For simplicity we assume that all service implementations are Jini-enabled, i.e., they are registered with the Jini Lookup Service and provide a proxy to plug them into the gateway.

1. **Bootstrapping and trading.** If the mobile device enters a network it uses the Jini bootstrapping mechanism to find a suitable LUS and then uses the Jini trading facility to obtain appropriate services.

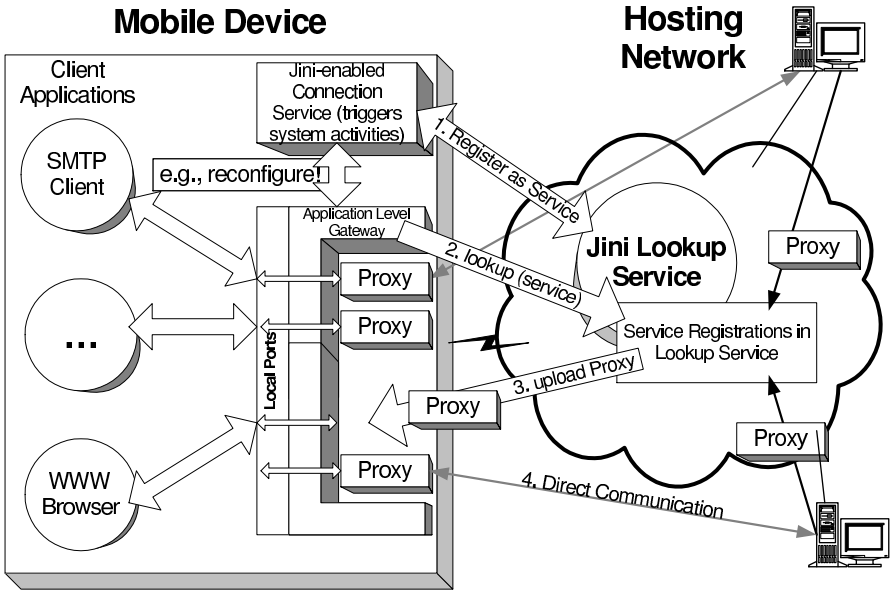

Fig. 2. Architectural overview of the gateway components

2. **Binding of services and bridging of data.** The respective proxies are loaded into the application level gateway and the client applications are able to use them. If the client application and the current service use the same protocol, the proxy functionality is reduced to forwarding of data which arrives at the local port to the host and port of the service. If they use different protocols the proxy must play the role of a protocol converter. Within the gateway the proxy can be plugged in in two different ways. If simple forwarding is desirable, the gateway simply hands over the data to a streaming interface of the proxy and accordingly expects the streaming interface to return data which are sent back by the real server. If protocol conversion is necessary, a programmatic interface of the proxy with certain operations due to the semantics of the provided service is used.
3. **Change Detection.** The Jini lease mechanism is used to immediately detect a locality change due to the fact that the user may spontaneously leave the network. Part of the gateway architecture is a connection service responsible for the detection of locality changes. It requests leases with a short duration, e.g., one minute, from the LUS. This requires regular short term renewal of the lease, i.e., some sort of a heartbeat. If the connection service fails to renew the lease, it may try to find another LUS. If this also goes wrong it is very likely that the device has left the host network and the connection service informs the application level gateway about this event. This information can be forwarded to currently plugged in proxies, which may take advantage of

it. Additionally it starts to periodically check for new Jini connectivity with the help of lower level services such as DHCP (see below) and also forwards this event to the application level gateway as soon as it has found a new LUS.

4. **Network Reconfiguration.** To obtain basic network connectivity, usually native services of the underlying operating system must be used. In the area of mobile devices DHCP (Dynamic Host Configuration Protocol) is often used to get an IP address and other bootstrap parameters from the network but such configuration is normally only performed at boot time. Since we can easily detect loss of network connectivity by the connection service, it periodically triggers the DHCP client application to check for new IP connectivity and reconfiguration of low level parameters, e.g., IP gateways, net-masks, naming services etc. If the connection service gets a positive return from the DHCP facility about a successful rebinding on the network level, it can go on to re-establish Jini connectivity as outlined above.

3.2 Legacy integration and enhanced features

If we want to reuse server implementations, it does not make sense to encapsulate each server within a Java/Jini proxy on its own. Moreover we provide a generic proxy which implements the streaming interface and is only parameterized with the actual values, i.e., host name and port of the real service. However, these proxies must be registered with the Jini Lookup Service. Since proper parameterization is necessary we have left this task to an extended version of our SCOT configuration repository [2], which is used to maintain service location within a network. It could easily be replaced by another mechanism, ranging from a simple service using file based configuration information up to an arbitrary complex service using relational databases or other repositories.

For some services we can reuse existing application level gateways, e.g., from firewall toolkits like DeleGate [7], classical daemons like the lpd printer spooler or standard WWW proxy servers like squid[1] [16]. This possibility is enabled by the application level gateway in terms of rewriting the respective configuration files and databases and restarting the application level gateways. Jini is only used to obtain the appropriate parameters from the host network.

4 Example – An Authenticating SMTP Service for Mobile Users

To illustrate the idea of a Jini-based application level gateway consider a mobile user who usually writes her mail off-line and in case she joins a host network, e.g., in a hotel, at a customer, or her bureau, she wants to have her mail delivered

[1] It is obvious that laptops running standard open operating systems like Linux which incorporate such gateways in standard distributions are much easier to integrate into the future open networks than devices using proprietary operating systems.

immediately to the Internet via the host network. This assumes that the mobile user has some trust into the host network infrastructure, otherwise she would refuse sending mail from a network until she joins a trusted network again.

For various reasons such as mail spamming, mail relaying, and billing for the transmission the host network itself is likely to not allow anonymous sending of mail from arbitrary mobile users that have joined the network. This is a major difference to current practice in companies, universities, and other institutions where the machines connected to the network are mostly under control of some system administrator who is responsible for proper setup of the machines. Users authenticate to the system at login time and are afterwards "known" to the system. A mobile device, though, must first be considered to be not trustworthy since it is unknown who is the responsible principal for that device.

Though the details of such authentication schemes for mobile users are topics of current research, we can for the sake of simplicity assume that a mobile user authenticates himself with a certificate that is publicly known or signed by some authority of the host network that grants access to the services offered. For example, a hotel might sign certificates of all the guests that check-in at the reception. These certificates can be considered as "tickets" for using services, or might itself issue certificates, e.g., in the form of smart-cards.

Institutions that provide a public SMTP (Simple Mail Transfer Protocol [14]) service will probably introduce another authentication scheme for mobile users that are about to use the SMTP service. Surely, each institution is free to offer enhanced services such as an *authenticating* SMTP (ASMTP) service that extends the SMTP protocol with some form of authentication. As an additional requirement we are interested in providing a solution that enables to reuse existing components such as mail user agents (MUA) and mail transfer agents (MTA) to offer a smooth migration path into a future world of Jini-based services.

The general architecture of the ASMTP infrastructure is shown in Fig. 3. The client sends an e-mail to the application level gateway listening on the standard SMTP port. After the appropriate Jini mail-proxy has been downloaded, the ASMTP proxy uses the security API to digitally sign the mail before transmission. The digitally signed mail is then transferred together with the client's certificate to the ASMTP service.

The ASMTP server checks the digital signature with the client's certificate and additionally checks, by using some authorization database, whether the client is allowed to use the ASMTP service. After authorization was successful, the mail is handed over to the standard legacy SMTP service which then routes the mail into the Internet.

One advantage of our approach is that on both ends we can simply reuse existing applications such as the mail user agent on the client side and the SMTP service on the server side. We basically introduce another intermediate layer based on Jini that manages dynamic trading and implements additional features such as authenticated communication.

Another advantage is that we explicitly make use of the code shipping features of Java. The computation of the signature is initiated by the ASMTP proxy

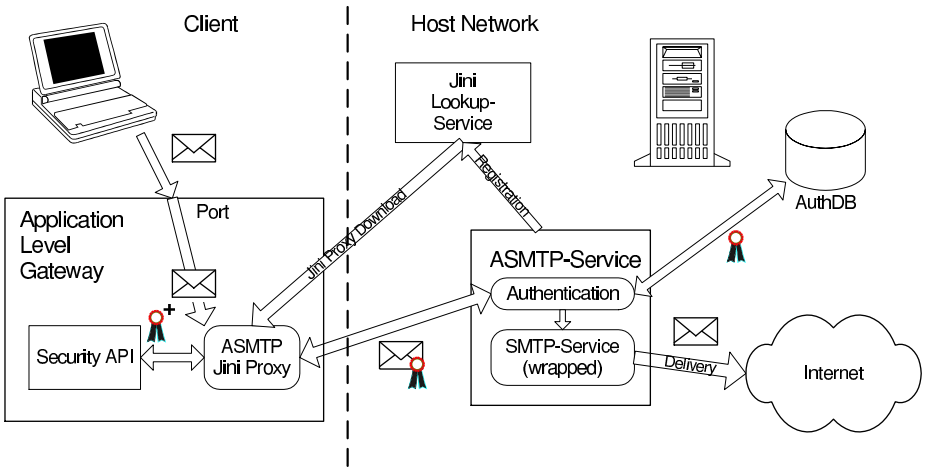

Fig. 3. Architecture for a Jini-based authenticating SMTP service

object but occurs entirely in the "trusted" environment of the client – seen from the client's perspective. Though the details of what degree of security can be achieved is beyond the scope of this paper, new interesting applications of proxy activities on the client side might arise.

5 Related Work

Recently, some new technologies were emerging aiming at the flexible and transparent integration of mobile devices and consumer appliances into different networked scenarios. Besides Jini there is ongoing work in the field of application level integration such like Ninja [11], or Millenium [15] but also in the field of network technologies, like Bluetooth [3], powerline networking [5, 9], or HAVi [4], for instance. Being in their infancy, we believe there are several migration steps in between – like ours – from todays configuration scheme to a fully automated integration promised by some of the technologies mentioned.

There are different approaches for service trading which could be used in general. We will briefly discuss selected examples:

- **Service Location Protocol**
  SLP [21] is a protocol which aims at the location of arbitrary services within an IP network. Its bootstrapping and trading facilities can be compared to Jini but it only provides information in the form of name-value lists instead of objects. In contrast to Jini, SLP can even be used in the absence of a central service registry.
- **Secure Directory Service (SDS)**
  SDS [6] is part of the ICEBERG project [8] located at the University of

Berkeley. Similar to the Lookup Service we use as part of Jini, the SDS acts as a distributed and fault tolerant repository for service announcements. XML is used for describing service properties and the matching of services in a SDS lookup call.

- **Universal Plug and Play (UPnP)**
  UPnP [20] is a recently announced service trading infrastructure built on top of HTTP-based multicast-protocols. Services register their URL with a central Simple Service Discovery Server together with an IETF-standardized type description. Clients query this server to obtain URLs of the desired type. Similarly to SLP, UPnP defines means for service trading without a central registry.
- **CORBA Trading Object Service**
  The CORBA Trading Object Service specified by the OMG [12] allows to register and find particular CORBA objects by specifying an arbitrary set of properties, such as type, name, location, costs etc. Compared to Jini, matching of properties is not restricted to exact and static values but allows for value ranges and dynamic property changes. A special language is defined for the specification of requests. Additionally a distributed implementation of the trading service is possible, which currently is not in the main scope of Jini. This CORBA service seems to be a more powerful mechanism than the Jini Lookup Service, especially with regard to scalability. However, in contrast to Jini, CORBA lacks a convenient or even portable bootstrapping mechanism, i.e., it is hard to obtain an initial reference of anything at all when a client enters an unknown network.

6 Conclusion and Future Work

We have shown that Jini is a valuable technology for the integration of mobile devices into arbitrary host networks, at least for stateless services or services which do not depend on long term network connections such as telnet or ftp. With our application level gateway it is not only possible to unburden the user of a mobile device from the awkward and error-prone reconfiguration of client applications, but also to allow for value added extension of certain services. A main advantage of our solution is to reuse both client and service implementations and to enable a seamless migration to future distributed infrastructures.

In the future we are planning to define a standard API for secure services, e.g., by using upcoming smartcard technologies like the Java Card API [10] for authentication purposes. Currently we have a focus on the transparent integration of a mobile device into its host network. To gain location-awareness it is possible to extend the connection service, e.g., in order to take full advantage of available services or contents. The big picture is an open network infrastructure integrating legacy devices and services (see [1]) as well as future component-oriented distributed applications. We believe that Jini is a well-suited enabling technology for our vision.

Acknowledgements

We would like to thank Friedemann Mattern for discussing an early draft of this paper.

References

[1] Gerd Aschemann, Svetlana Domnitcheva, Peer Hasselmeyer, Roger Kehr, and Andreas Zeidler. A Framework for the Integration of Legacy Devices into a Jini Management Federation. In *Proceedings of Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM'99)*, October 1999.

[2] Gerd Aschemann and Roger Kehr. Towards a Requirements-based Information Model for Configuration Management. In *Proceedings of 4th International Conference on Configurable Distributed Systems (ICCDS'98)*, pages 181–189. IEEE Computer Society Press, May 1998.

[3] Bluetooth Consortium. The Bluetooth Project. `http://www.bluetooth.com/`, 1999.

[4] HAVi Consortium. *The HAVi Specification: Specification of the Home Audio/Video Interoperability Architecture Version 1.0 beta*, November 1998.

[5] Nor.Web Consortium. Nor.web's digital powerline solution. `http://www.nor.webdpl.com/`, 1999.

[6] Steven Czerwinski, Ben Y. Zhao, Todd Hodes, Anthony Joseph, and Randy Katz. An Architecture for a Secure Service Discovery Service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCom'99), Seattle, WA*, August 1999. Draft version, accepted for publication.

[7] DeleGate Home Page. `http://www.delegate.org/`.

[8] ICEBERG Project Home Page. `http://iceberg.cs.berkeley.edu/`.

[9] Siemens Inc. Powerline Communication. `http://www.siemens.de/`, 1999.

[10] Java Card Technology. `http://java.sun.com/products/javacard/`.

[11] Ninja Project Home Page. `http://ninja.cs.berkeley.edu/`.

[12] OMG. *CORBA Object Trader Service*, December 1997. Available at `http://www.omg.org/`.

[13] Charles E. Perkins. Mobile networking in the internet. *Mobile Networks and Applications*, 3(4):319–334, 1998.

[14] J. Postel. Simple Mail Transfer Protocol. Internet RFC 821, August 1982.

[15] Microsoft Research. The Millenium Research Project. `http://research.microsoft.com/sn/Millenium/`, 1998.

[16] Squid Internet Object Cache. `http://squid.nlanr.net/Squid/`.

[17] Sun Microsystems Inc. *Jini Architecure Specification – Revision 1.0*, January 1999.

[18] Sun Microsystems Inc. *Jini Discovery and Join Specification – Revision 1.0*, January 1999.

[19] Sun Microsystems Inc. *Jini Lookup Service Specification – Revision 1.0*, January 1999.

[20] Universal Plug and Play Homepage. `http://www.upnp.org/`, 1999.

[21] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol (SLP). Internet RFC 2165, June 1997.