Design of a Contact Service in a Jini-based Spontaneous Network

Jan Bäckström[*a] and Simin Nadjm-Tehrani[**b]

[a]Erda Technology AB; [b]Dept. of Computer & Information Science, Linköping University

ABSTRACT

In this paper we report on a study of the Jini technology for design of spontaneous networks offering services dynamically. We identify important design criteria such as flexibility, efficiency, dependability and transparency in the architectural design of spontaneous networks and illustrate how these criteria affect the specific design of a contact service offered within a Jini network. We explain the architectural design of the contact service, the lessons learned, and the future directions for research in extending the Quality of Service (QoS) and dependability enforcement in Jini networks. The report also includes a brief comparison with other current technologies for similar networks.

Keywords: Jini networks, spontaneous networks, search services, GUI architectures, quality of service (QOS)

1. INTRODUCTION

A spontaneous network is a network taking care of itself. Members are automatically connected to the network, establish their required channels and go about their businesses, with the underlying mechanisms being transparent for the user of services offered within the network. In this paper we report on work performed at a Swedish company Erda Technology (formerly Framfab), on implementation of a contact service in a spontaneous network. The work was in partial fulfillment of a masters thesis, and examined the use of the Jini technology[1] to achieve this. We present the basic factors behind the choice of Jini in comparison with other competing technologies, the chosen architecture, and some aspects of the detailed design and implementation. The paper can be seen as a condensed version of a masters thesis (written in Swedish) on the same topic[2].

Consider a scenario wherein the user of a contact service needs the contact details of a particular person while in movement (e.g. in an airport, wanting to make a phone call and needing the phone number). The chances are that the most convenient mode of connection to an instantaneous network from which this information can be obtained is via the user's handheld device. Now, the user should be able to connect to the network with minimal knowledge about the underlying mechanisms, irrespective of which type of handheld device he has, and where the interesting information is stored. The contact service should, moreover, be able to use heuristics in identifying the most likely source of information for the current user profile, and perform the search using these guidelines. The goal of our project is to explore architectures in which the user is helped to exploit the resources in an spontaneous network, irrespective of the mode of connection, and for any form of information retrieval. This should moreover be done efficiently, flexibly, in a dependable manner and with full transparency. Efficiency means that the adopted technology should be mature and scalable, working for potentially large (spontaneous) networks, and for small devices with limited resources. Flexibility implies that the user can be served in different styles based on the current profile, e.g. the type of connected device, the type of query, and the current network traffic. Dependability implies that the contact service should explore alternative network resources to obtain the required information transparently, even if there are certain node failures or link failures. Ideally, the user should be able to provide a Quality of Service (QoS) profile and get a response generated

[*] jan.backstrom@erda.se; phone +46 13 37 72 13; Erda Technology AB, St Larsgatan 12, S-582 24 Linköping, Sweden
[**] simin@ida.liu.se; phone +46 13 28 24 11; Fax +46 13 28 40 20; Dept. of Computer & Information Science, Linköping University, S-581 83 Linköping, Sweden

based on principles of adjustable autonomy – the system dealing with as many decisions as possible, and only when needed ask for user's priorities and additional information[3]. Transparency is the cornerstone of instantaneous networks and will be discussed in more detail below.

2. BASIC TECHNOLOGIES

With pervasive computing as a likely future scenario, application design is taking a step from the individual desktop or embedded device level to the net-centric. As Bruce Power Douglass explains, the basic technologies for achieving computation, interaction and integration exist, but the advent of pervasive computing requires that the isolated devices be robustly linked, transparently reconfigured and have a usable user interface[4].

> *What prevents the pervasive computing era from becoming a viable, flourishing ecosystem instead of a fantasy? Simply put, the massive amount of software it takes to create individual devices, link them together, and provide the distributed systems with the capability required to make everything work together.*
>
> B. P. Douglass

In order to enable a hand-held computer, a telephone, a fax or any other device connect itself to an existing net, identify the required resources, make connections and perform its tasks, every node must be identified with an IP address. In addition we require a distributed system technology for the connection, and the maintenance of the quality of the service. The basic technologies for implementing such a system are in place. In this paper we make a brief comparison between the commercial platforms of E-speak[5], Universal Plug and Play[6], and Jini. We then go on to describe how our system was designed based on Jini, and our conclusions based on this experience.

Each of the technologies mentioned below are promoted to achieve at least a subset of the criteria listed in the introduction. However, the forces acting on the market are subject to complex dynamics. Thus, most vendors are primarily keen to capture a large portion of the initial and emerging markets, as opposed to fully evaluate and test their products on large-scale studies. Hence, the comparative information here is not as comprehensive as it would be in a fully benchmarked study. It should only be seen as a starting step in this direction. Much of our initial judgment was based on the claims of the developers of the technologies and a matching of the promoted qualities with our requirements.

2.1 E-Speak

E-Speak is a commercial product developed by Hewlett Packard. It is a technology promoted to develop electronic services on the Internet, and includes

- ✓ Service interface on which it is possible to build other components
- ✓ Support for communication with underlying systems such as databases and enterprise networks

There is a kernel that manages traffic between all units. In order to connect to the kernel, the unit must implement a E-Speak Service Interface (ESI). There is an ESI for most programming languages, and the technology can thus be considered as language-independent. The information used by the kernel is the general information available about resources – e.g. if the resource is a file, it includes the attributes of the file and how to read it. The actual reading of the file is dealt with through the local resource manager. That is, a node that connects to the network does so by connecting its local resource manager to the E-Speak network, and sending its attributes (meta-data) to the kernel. It can then be used once the kernel receives a message from a potential user of the resource, and passed on to the appropriate manager via the meta-data. This flow of information is illustrated in Figure 1.

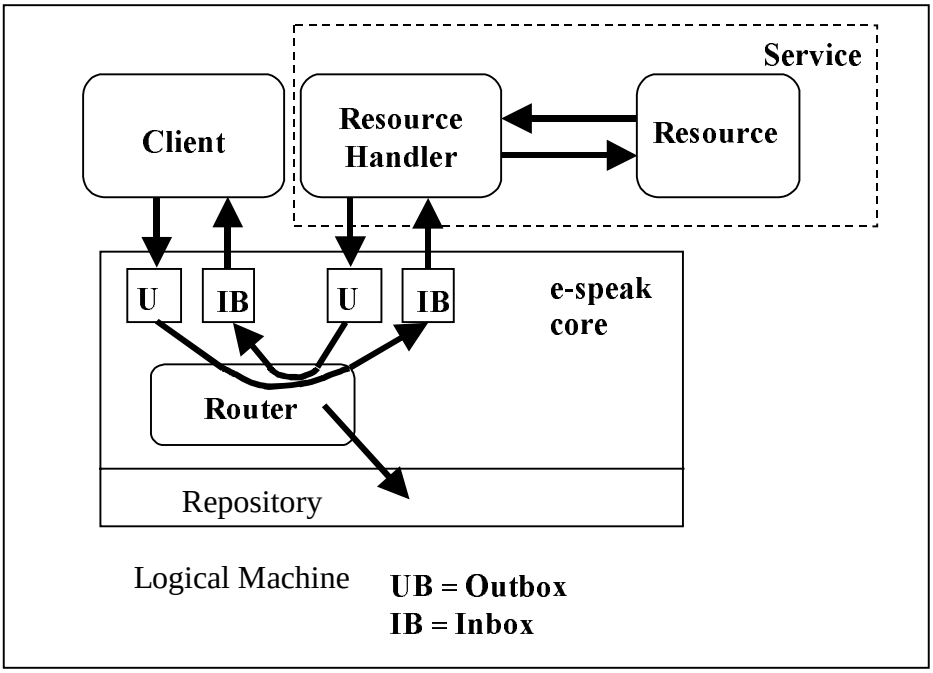

Figure 1: The E-Speak architecture and call patterns

2.2 Universal Plug and Play (UPnP)

UPnP is developed by Microsoft and is promoted to achieve transparency as a main goal. To a user of UPnP there is no difference between a local and a remote computation and the design includes protocols at the lower network layers (below network layer). Different devices coming on a network announce their presence via the low-level device interfaces available and UPnP's "Simple Service Discovery Protocol". Each connected device eventually gets an IP address, announces its presence, and listens to the special types of messages seeking its use. Seen from a user (client) perspective, the information about device characteristics and use are stored in an XML database.

2.3 Jini

This technology was developed by Sun Microsystems with the vision that new units should be added to a network as simply as we add telephones today. Jini is implemented in Java and makes connection of any device (unit) to the network possible irrespective of the operating system. A unit communicates with the Jini network by moving a Java program (an object), typically using Remote Method Invocation (RMI).

When a resource is connected to the network it first locates the networks Look-up service. It then presents the services it can provide within the network, the information to be stored by the Look-up service. When a client is looking for such a service, it in turn finds it via the Look-up service, loads the code for the object that facilitates communication with the sought service, and from then on, is in direct contact with the service provider. The duration that a service is available on the network is governed by the leasing contract initiated when the service provider unit joined the network.

2. 4 The selected technique

One goal of our project was to try one of the current technologies in the design of a spontaneous system in order to evaluate its general applicability, and to test it in the context of an application. The initial survey of the above technologies led to the choice of Jini due to the following reasons:

- ✓ It is a Java-based technique, which is a modern and popular programming language.
- ✓ It seemed easier to program than the other alternatives.

- ✓ It allows direct communication between the resources and is not dependent on a middle-node like E-Speak - this should lead to higher efficiency.
- ✓ It seems to be more applicable to the scenario we sketched above, whereas UPnP appears more suitable for connection of physical devices, and E-Speak was formed around e-commerce needs.
- ✓ It was freely available and easy to get hold of.

2. 5 Other related works

Jini provides mechanisms for service construction, Look-up, communication, and use in a distributed system. It can be used to implement a service in mobile systems, but does not require the software agents to be mobile within the network. Thus, the system we have developed does not directly fall into the mobile agents categories described by Ciancarini et al.[7]. It is perhaps most similar in functionality to the Voyager ORB, classified in the same report, wherein the migration object can be a program as well as a Java object[8].

3. DESIGN OF A CONTACT SERVICE

The design of our contact service essentially consisted of the following stages:

- ✓ Design of the architecture, taking account of the need for:
  - ✓ flexible GUI, based on requirements for each network element
  - ✓ placement of client's search requirements (i.e. the logic behind which Look-up service to choose), based on the available memory in the client

- ✓ Design of the search service, taking account of the need for:
  - ✓ filtering and organizing the presentation of returned solutions
  - ✓ expression of search criteria (narrowing down the search)

3.1 Architecture design

Two alternatives were considered in the design of the GUI:

- ✓ To download the GUI into the client together with the other programs to be used in the contact service
- ✓ To download only a start script for the contact service in the client. When the user starts the script the GUI is downloaded from a server that contains a suitable GUI for that user (device).

The first alternative means that the client stores the interface in memory, whereas in alternative two the GUI is stored in a server. The first option is easier to implement and quicker to start up, since no server is needed. The second option is more attractive if no memory is to be tied up at the client side and at the same time gives more flexibility for updates. The most recent GUI is always available to any client who connects to the network, and no local updates are necessary for changes to the GUI over time.

We chose the second alternative, both due to the mentioned flexibility and due to the attraction of thin clients. This choice gives the most efficient use of memory in small devices, makes it possible that even a small device can be connected to a large screen and uses a more appropriate GUI for a particular session. The characteristics of the current screen can then be sent with the start up data so that the Look-up service can find the most fancy GUI based on the size and other details. The idea is that there should be one GUI service for every type of unit connected to the Jini network – a special one for the PDA, another version for the telephone, a third one for the desktop, etc.

Figure 2 shows the architecture to load the GUI. The single arrows in Figures 2 and 3 represent calls to the resources and the double arrow represents direct communication between the two nodes. The GUI service provides the user with a graphical interface to be used as a portal to the search service. When a client node runs its start script it will search for all available Look-up services and registers itself. The registered proxy uses RMI to communicate with the object that implements the GUI service at the server side. That object receives the commands from the client, via the proxy, and returns a graphical interface. The object is activated when a client uses the GUI service.

For another type of service where the user in fact wishes to choose the required GUI, as opposed to the above automatic filtering scheme, the GUI service can be altered so that a list of available GUI types is presented to the user, and the user actively chooses. This solution thus provides the flexibility between active choice of GUI and an automatic filtering version for different applications.

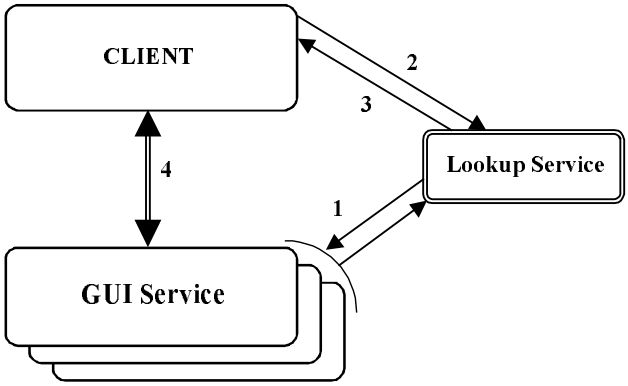

Figure 2: The client locates and loads the appropriate user interface

The numbers in Figure 2 explain the call sequence:

1. Each GUI type publishes towards all adjacent Look-up services in the Jini network.
2. The client, e.g. a mobile phone or a desktop looks for a suitable GUI (for itself) via the Look-up service.
3. When the client finds an appropriate GUI service, it downloads the service object.
4. Now the client can directly communicate with the GUI service and call the method that returns the GUI.

3.2 Design of search service

In the context of the search service, three factors were considered important: robustness, scalability, and adaptation to quality of service (QoS) specifications. Robustness is required to take care of failure scenarios; e.g. to deal with the cases where the client or the search server disappear from the network before all the information has been locally delivered. Scalability was important since in the current stage there are no small enough JVMs for several small hand-held units. It was therefore envisaged that both solutions based on “one JVM per network-unit” and “local gateway” solutions for connecting units to the Jini network would work. QoS requirements are a wider topic not entirely dealt with in our system. However, we have considered how our system can be extended to be able to deal with a scenario whereby the user states how efficient the search should be for a particular activation of a service.

When designing the architecture of the search service, it was important to consider how the client would contact different search services in the Jini network. The logic behind the choice of search service could either be placed directly in the client, or it could be placed in the GUI. The first alternative has all the already mentioned disadvantages, i.e. memory would be bound up in the client for the constant storage of search criteria. Also updates would be problematic. However, if the logic is placed in the GUI object, these problems are overcome. In the second alternative the client would always get access to the most recent search service. This is why we chose the second option here, as depicted in Figure 3.

The search service is responsible for finding the right service provider. The downloaded GUI object that includes the search logic collects the user input for a particular search session. From the values collected a search query object is formed, which is used in the search process all the way down to the resource where the search is performed, e.g. a database. To avoid overloading the GUI object by all hit results, the service provider signals with a ready event when the search is completed. Then the GUI object can fetch as many hits as required and discard the rest.

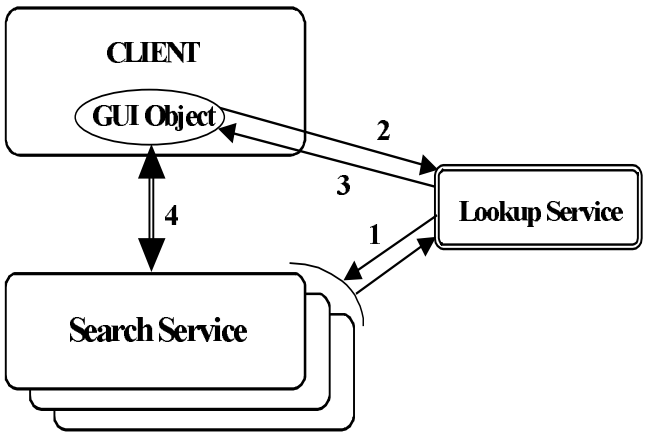

Figure 3: The client looks for a search service via the GUI object

The numbers in Figure 3 explain the call sequence:

1. Service providers publish their services with the nearest Look-up service.
2. GUI object searches for the search services using the Look-up service and a given search template.
3. Look-up service returns a service object for the matching services.
4. GUI object now has a number of handles for the search services it has found via the Look-up service. It then performs a direct search in every search service, and shows the result in the GUI.

The search activity consists of at least two filtering levels. One level is the restriction of the search to look only for interesting search services. The next level is to take care of the results in a good manner. The first restriction requires that the GUI is built up in a way that an advanced filtering can be formulated. For example, to make a simple filtering the GUI can be formed to contain a number of simple fields to include search criteria. In searching for a telephone number one can have a field for a company name that leads to searching in that company's databases in the first instance, and if left empty, implies searching in the private catalogues. Using other keywords could restrict the search for the search service to other domains, e.g. "golf" for golf clubs. Thus, it is clear that the design of a filter can be as sophisticated as desired.

As far as taking care of the results, as mentioned before, we have chosen to let the client specify the number of hits to fetch at any time. This makes it possible to deliver the results in different forms for different units. A small unit can then differentiate between the number of hits fetched compared to a larger unit, and the client can stop the delivery when a satisfactory hit is found.

3.3 Case study

In this section we describe the application of the above general architecture to the specific case of a client trying to find contact details for a particular person. In this case, first the network connection is set-up according to which device is connected to the network, in accordance with Figure 2. Then, the connected device approaches the network Look-up services in order to find nodes that can provide answers to its query, in accordance with Figure 3. Once such a node is found, the interaction between the client and the search server will be carried out as described earlier using the class diagram of Figure 4.

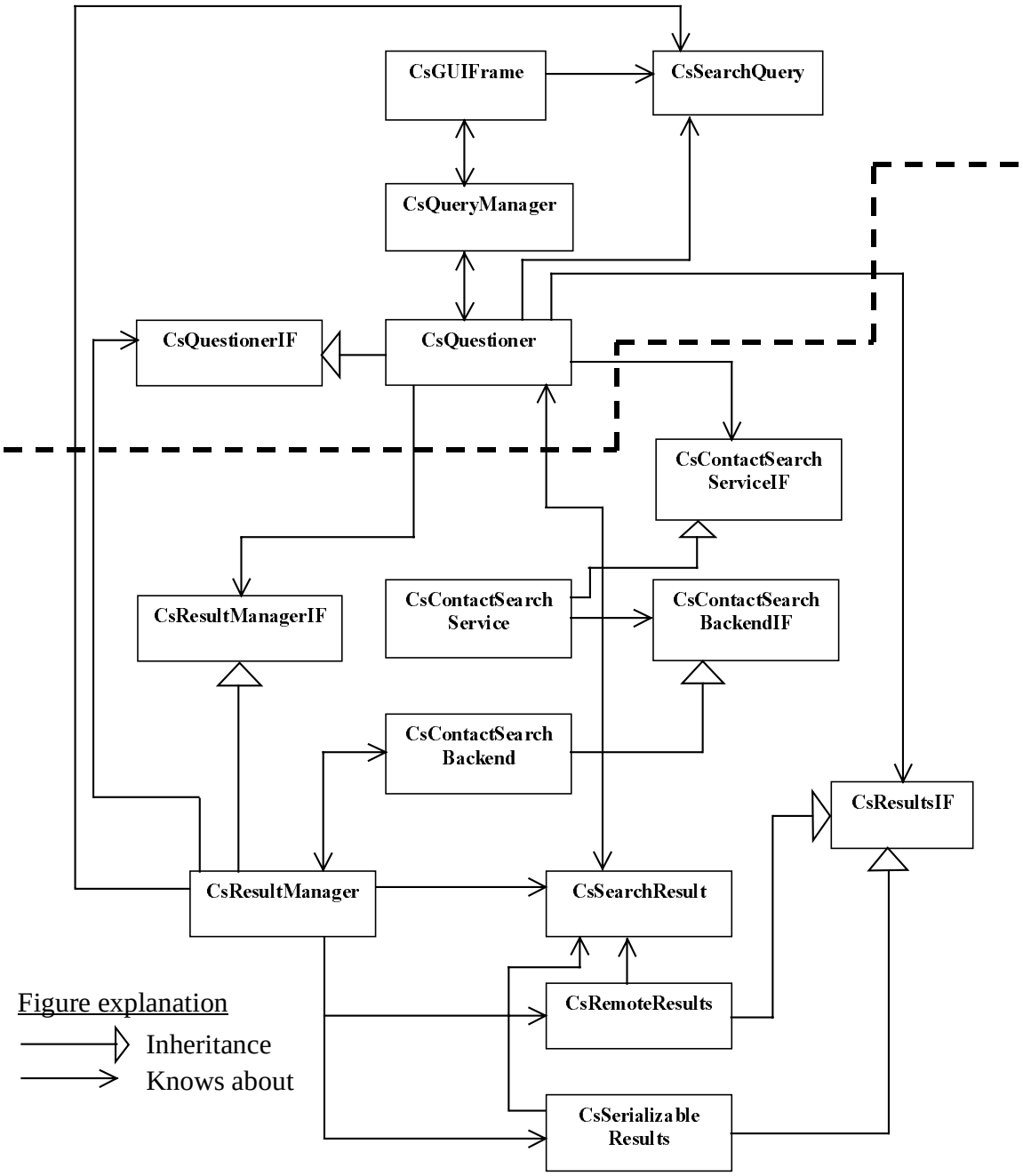

Figure 4: The class diagram for the search service

The parts of design that were implemented in the prototype were as follows. A complete Jini network was set up including its Look-up service. Two simple types of GUI service were also implemented: one for name of a person and one for the company. Instead of connection to databases two simple search services were built to begin with. The search was performed on text files with relevant data (name, attributes, telephone number), on which searching on name or parts of a name were implemented. The idea with the prototype was to investigate the practicality and ease of programming a functioning application based on Jini. Thus, fancy user interfaces and intelligent search strategies were

of lesser importance at this stage. Also, we decided that all the hits be shown in the GUI to begin with. The system can later be augmented by smart filtering mechanisms or, result ordering according to some user criteria.

It was not necessary to connect the nodes in a particular order. If a unit is connected to the network and there is no Look-up service available yet, the node simply waits and "listens" until such a service appears. A web server has to be present in order to start the Look-up service in order to get a Jini network going. This will be used to download the files into the connected units or the proxy objects. We chose to use the existing HTTP server within the Jini package on a desktop.

To test the idea with multiple types of units on the same net, we implemented two example GUIs. One larger screen to simulate a desktop computer, and a smaller screen to simulate a handheld device. When the GUI service is started it will ask the user to choose the relevant sized screen. The results of the search are presented in the same window too (see Figure 5 for an example).

Figure 5:GUI for the contact service prototype

3.4 Experience from the implementation

The main effort in the implementation was spent on the search service; not because there were complex search algorithms included (yet) but due to the need of being able to deal with dynamics of the application. There were 18 calls included in a search action as documented in our interaction diagram for the design[2]. For example, after the call to the search service, a result management object is created with its own thread in order to enable the search server to deal with new incoming calls. We can summarize the properties that affected our implementation decisions as follows:

- ✓ **Availability:** This affected the spawning of new threads to deal with each stage of the query processing so that new queries could be accepted in a dynamic fashion.
- ✓ **Transparency:** This affected a number of choices, e.g. in the result management. When a result is found by e.g. searching a database, it may be stored as a remote object (accessible from other nodes via the network) or as a serializable object (which can be sent to another node via the network). In which form the result manager chooses to store the results, is transparent to the client who only sees the results interface.
- ✓ **Flexibility:** The above choice affects the flexibility in delivery of the results. For example, if the number of hits is small then a serializable object is a better choice since all results can be sent to the querying node at once. Whereas if the number of hits is larger than a given limit, then they can be stored remotely and serialized in chunks. The choice of the limit is application dependent and decided for each implementation.

- ✓ **Efficiency:** The choice of number of resulting hits to send over at any time also affects the network performance. If the results are serialized with one hit at a time, there will be many messages back and forth, in cases with large hits. So this aspect is also to be considered when fine-tuning an implementation.

To be able to adjust all these aspects of the implementation with ease and based on the criteria from future new applications was a major part of the implemented contact service – which we tried to consider as a specific instance of a large class of applications.

3.5 Difficulties and reflections

To get a Jini network going requires a good knowledge of distributed computing mechanisms. In our case the used mechanism was RMI, but the same would apply to CORBA-based settings. Once this background knowledge was in place, the actual Jini functions were not difficult to run.

4. CONCLUSIONS AND FUTURE WORKS

This study has shown us the strengths and weaknesses of building a spontaneous network based on the Jini technology. The prototype has demonstrated how our major design criteria, flexibility, dependability, efficiency and transparency affect the design of a dynamic network of devices using the network services (also dynamically added and updated). There is more work to be done in order to get the dependability and QoS requirements guaranteed in a Jini-based network. For example, how and when the user of a service should be notified when the nodes involved in a search chain crash or leave the network. This will be an interesting direction for future works.

Also the way the search is guided, via QoS parameters, will affect the number of hits, and the number of result manager objects spawned. It will in its turn affect the achieved performance of the network in response to the query. This is thus an instance of a trade off between an architectural decision (to serve robustness and availability) and a user supplied parameter (to achieve flexibility and QoS). The study of such trade off based on more realistic applications and given alternative (search) algorithms is an interesting topic for the future.

The work has been somewhat restricted in the comparative analysis with other commercial technologies. However, there are some aspects that one can already point out: the direct connection to a service provider via a Look-up service has a lot of advantages as pointed out in the above sections. However, the direct connection has also its disadvantages. Where the service provider needs a record of the usage of a service, e.g. in the case of billing or editorial moderation of publishing services, additional work has to be done in a Jini network, whereas the E-speak solution is tailored to these scenarios by design.

Preliminarily we can say that the ability of Jini to build on the strengths of Java (platform-independence, etc) is seen as a major advantage. Seen from a user perspective, Jini makes system configuration an easy task. No special knowledge of device drivers and installation scripts will be necessary for a Jini connection. However, for pervasive computing applications small footprint JVMs are essential, and unless progress is made in that direction, a Jini-based paradigm for spontaneous networks will not lift off.

5. REFERENCES

1. The Jini Community, What is Jini Network Technology? Currently available on http://www.jini.org/whatisjini.html, February 2001.

2. Jan Bäckström, Design av en kontakttjänst i ett spontant nätverk, Masters thesis, number LiTH-IDA-Ex-00/101, December 2000, Dept. of Computer and Information Science, Linköping University.

3. B. P. Douglass, The Evolution in Computing, currently available on http://www.sdmagazine.com/articles/2001/0101/0101b/0101b.htm, January 2001.

4. AAAI Spring Symposium on adjustable autonomy, call for papers, http://www.aaai.org, 1999.

5. Hewlett Packard Company, E-speak information page, currently available on http://www.e-speak.hp.com, February 2001.

6. Microsoft Corporation, Universal Plug and play Forum, currently available on http://www.upnp.org/forum/default.htm, February 2001.

7. P. Ciancarini, A. Giovannini, D. Rossi, Mobility and Co-ordination for Distributed Java Applications, in Distributed Systems, Lecture Notes in Computer Science Volume 1752, January 2000, Springer Verlag.

8. Voyager Object Request Broker, currently available on http://www.objectspace.com/products/voyager/, February 2001.