JIMA
A Jini-based Infrastructure for Active Documents and Mobile Agents

Fredrik Kilander (fk@dsv.su.se)
Patrik Werle (werle@dsv.su.se)
Karin Hansson (karin-ha@dsv.su.se)
K2LAB
Dpt. of Computing and Systems Sciences
Stockholm University

October 18, 1999

Abstract

An infrastructure for mobile agents based on Jini is presented. Jini technology enables Java-based clients to dynamically discover and employ services in a local area network without prior knowledge of URLs, hosts or port numbers. In a Jini framework agents may act as both services and clients.

In our application scenario mobile agents implement active documents that actively seek out their recipients. A recipient may be locally or remotely connected to the application network. The infrastructure therefore also includes personal user proxies, a localization service, terminal services and subsystem resources. Jini is proposed as a convenient technology for connecting the various components.

1 Introduction

Within the paradigm of ubiquitous computing instant access and connectivity are important concepts. Connectivity is not necessarily restricted to people. In our work, we have focussed on the ability to cast document flows into a framework which allows the document themselves to be intelligent, proactive and social participants rather than inanimate frames of stored data. Instead of people reaching for documents, the documents try to reach the user.

Our approach to active documents presupposes an infrastructure in which documents consists of mobile agents with plans, purpose and knowledge of the payload and the organization in which it plays a role. There are several important issues which must be solved in this infrastructure, for example temporal constraints, transportability, addressing and security. In addition to the documents themselves, the infrastructure should facilitate convenient access to generally available resources like document format converters, user terminals and storage facilities.

On a practical level our amibition is to deliver, if not the document itself, at least a rendition of its content to the recipient. In the presence of an agent, dynamic *presentation* is easily generalized to *interaction*. At that point display becomes an instance in a general framework of distributed applications.

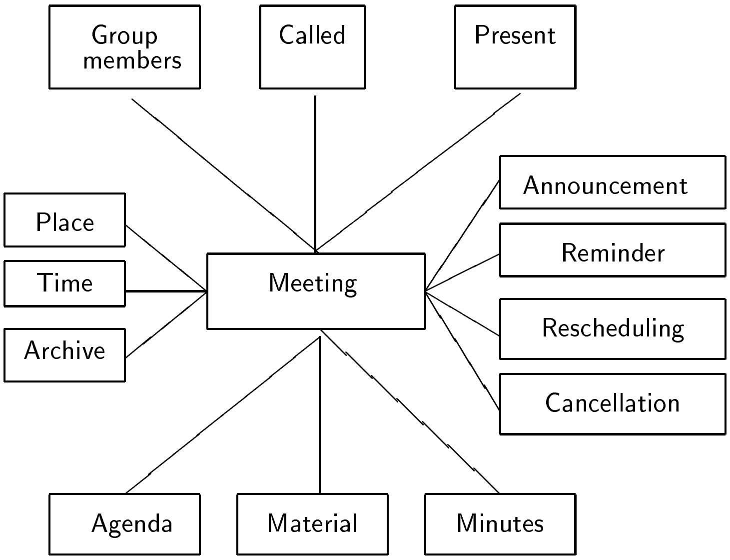

Figure 1: A simple meeting model.

1.1 The Meeting Scenario

We begin our scenario description with a model of a meeting process. The model is then partially instantiated in a number of active document agents with specific tasks and roles.

1.1.1 The Meeting Model

The model shown in figure 1 contains the basic elements which relate to the concept of a meeting between people. We assume that regular project meetings, the assembly of a board of directors etc, falls within this model.

At the top the model contains three groups of people. The *group members* consists of the people from which a meeting is expected to be assembled. The *called* people are those who are requested to participate at a particular instance of a meeting and they are (usually) a subset of the group members. Finally, the group of those *present* are the people who actually attended the meeting.

On the left, there is a *place* and a *time* which represent the planned location time of the meeting. The *archive* is where information about the meeting is stored for future reference.

Before the actual meeting is to take place, certain messages must be sent to those who are expected to participate. These messages are modelled on the right side in figure 1. The *announcement* declares that the meeting is scheduled to occur at the designated time and place. One or more *reminder* messages may have to be sent. It may also be necessary to send out *rescheduling* messages when the place or the time has been changed, and if the meeting is cancelled, a *cancellation* message must be distributed.

Before the meeting documents and resources are prepared by those who will attend. There is an *agenda* and reference or presentation *material*. After the meeting there will also be em minutes which formally records the contributions of the meeting.

1.1.2 Applying Active Documents

Since we want to apply our active document architecture to this scenario we begin by describing the meeting as an active document. The task of this document is to maintain the information which relates to the particular instance of the meeting. The meeting document is to make sure that information gets updated and disseminated in a controlled manner.

Some of the information, like the agenda, can only be updated by a restricted number of people, but may be readable by all group members. Material may be submitted by several people, or perhaps passed before a moderator. The meeting document agent should enforce these restrictions.

In addition, the meeting should make sure that messages are sent to the participants. These messages are of course also active documents, but their task is more concentrated on the job of finding the user and imparting the message content. Announcements, reminders, cancellations and reschedulings are all important, especially when the time remaining to the meeting diminishes. For example, we can immediately see that intelligence in these message documents imply the following behaviour:

- Announcement and reminder messages should be delivered well before the meeting begins. Cancellations should be delivered at once.
- A rescheduling message for a later meeting has the delivery time limit of the original meeting.
- A rescheduling message for an earlier meeting has the delivery time limit of the new meeting.
- A rescheduling message should supercede undelivered announcements and reminders for the original meeting (assuming that no relevant information is lost).
- A cancellation message should supercede undelivered announcement, reminder and rescheduling messages.

Without having dealt with the mechanics of message delivery (more about that below) we assume that message content somehow reach the user in one form or another. We can at this point not guarantee that messages (active document agents) will reach their destination in the same sequence as they were sent. Possible reasons abound; network failures, computer malfunctions, routing problems etc. The common solution to this problem in computer network architectures is to buffer incoming messages and resend delayed messages. However, in this particular scenario we assume that

- messages have a hard delivery time
- some messages supercede previous messages
- messages are programmable.

This makes it inconvenient to apply standard techniques to these issues. A solution which involves negotiating agents appears intuitively more attractive.

Before we end this section, let us return to the meeting and its components. The materials sent to the meeting may be required reading for the participants. In this case, a copy should be sent to those called and each copy falls under the same delivery restrictions as the other messages. Furthermore, it may require the user to interact, e.g. actively confirm that a certain text has been read.

During the meeting, certain material may be required to be physically available to the participants. In this case, resources such as printers, faxes, projectors and software associated with the location must be located and utilised by the documents. For example, the final agenda is printed, slide presentations are downloaded into or accessable from a multimedia system or the participant's mobile computers are given the necessary access rights.

Finally, before the meeting can archive itself, the minutes should be approved and signed by the appointed members. A copy of the minutes should also be sent to those who were present (or called).

2 Overview of the Jima infrastructure

The following discussion assumes that the reader is familiar with general Java programming, Jini basics like clients, servers and lookup, and mobile agent technology.

2.1 Components

The Jima infrastructure consists of a number of Jini services. Several of these services are also Jini clients. A rough categorization of the services and clients results in the following:

- Jini lookup servers
- Mobility servers
- Personal proxies (aka personal routers)
- Active documents
- Localizer services
- Terminal services
- Subsystem services

2.1.1 Jini Lookup Servers

Jini lookup servers play no special part of Jima other than in their ordinary role in Jini, and at the moment we place no particular requirement on either the number of servers or other possible properties. For the moment we will be content by recognizing that the system presupposes the functionality of one or more Jini lookup servers.

Currently we limit the expected scope of a Jima infrastructure to a local area network (LAN) as can be found in a medium-sized company or university department. The purpose of this narrow view is simply to postpone the issue of how to transparently enable Jini lookup and discovery across networks which are separated by non-controlled routers, for example two LANs in different cities or countries.

2.1.2 Mobility Servers

A mobility server, in our framework, is an execution environment (like a Java Virtual Machine) in which the code of a mobile agent may inject itself for execution. Various levels of sophistication exist in the creation of an execution environment, from simple exploitation of Java RMI to more developed environments.

Mobile agents has a common problem in that they must know exactly where they are going next. We propose Jini as a way for mobile agents to discover mobility servers in the network and the service object mechanism as a convenient way for both parties to negotiate entrance and service parameters.

2.1.3 Personal Proxies

A personal proxy is a service which fulfills two important tasks. The first is to persistenly represent the user in the network and give the user an address and a net presence. The address of a proxy is a property of its registration with a Jini lookup server.

The second task is that of routing, i.e. the ability to find and establish a communication with the user.[1] For simplicity we can view message filtering as a special case of routing.

Active documents approach the proxy to reach the user. The document must negotiate with the proxy for a level of access to the user. The proxy and the document together form a decision by weighting together user preferences, document importance, the time of day and the current or expected reachability of the user. In some cases the document is only allowed to hand over a short text abstract, in other cases it may talk directly to the user across a sophisticated graphical user interface of its own design.

2.1.4 Active Documents

Active documents consists of agents and payload. The agent part of the document is there to ensure the proper management of the document. For example, that the correct recipients are notified and encouraged to respond in a timely fashion; that different renditions of the content are available and that workflow protocol and business rules are observed.

Since the agents are mobile they are expected to migrate between mobility servers. Although we do not yet exploit mobility to any critical degree we foresee interesting possibilites. For example, the ability of a document to physically migrate onto a mobile platform (like a laptop computer) and continue to execute there even though the computer has been disconnected from the local network.

The payload in an active document may be many things. In the trivial case it is some data which the agent can manipulate directly. A more interesting and challenging alternative is to define the payload as one or several other agents, each specialized to perform a certain duty once the outer envelope has fulfilled a necessary objective, for example transportation of the document to its destination. This model appears to have a tantalizing potential to generalize recursively into a structure which bears similarities to other layered communication models. We will not, however, expand that topic further in this paper.

Active documents act as Jini clients of proxy servers and mobility servers. In some cases they may also consult a terminal service (see below).

2.1.5 Localizer Services

Localizer services are sensors, interpreters and databases which together forms a system for tracking the physical position of people and devices. There are many debatable issues with this, so we will immediately recognize that localization information may be restricted to certain areas and times.

It is important to include devices as they must be localized too. A process in a portable computer, for example, may be able to guess with a certain probability that user Ahmed is close because Ahmed has logged in. If the laptop could retrieve Ahmed's position from a localizer service, it could also form an hypothesis about

[1] Apart from physical users, there may also be abstract users like *HelpDesk* or *Marketing*, in which case the proxy must mediate between the abstraction it represents and other truly personal proxies. This may be somewhat more involved than simple routing for reasons which are not covered here.

its own position. If the laptop exports the fact that Ahmed is close to the network in general, other systems are able to form similar hypotheses.

We do not specify the existence of a particular localization service, nor do we require one, but it is easier if there is one.

2.1.6 Terminal Services

Terminal artefacts are physical user devices, such as video displays, mice, keyboards, live-boards, printers etc. These devices provide their usual service, i.e. input from and output to users.

A *terminal service* is a Jini service which acts as a front-end for a terminal artefact. In most cases, the service object thruthfully exports the generic capacity of the device. It may also extend the services of one or more *virtual devices*. These may be provided by additional capacity implemented in software, like a simple printer being driven by a postscript driver, or color filtering on a monochromatic display, or a soft keyboard on a pen computer. There are some important points to be made here:

- The same service instance may act as a terminal service for several terminal artefacts.
- Terminal services may be clients of other services, like conversion, security and charging services.
- Terminal services may be connected and disconnected in an ad-hoc fashion.

It should be clear that the Jini terminal services form a layer of abstraction over the total collection of hardware in the network. The purpose of this layer is to enable the agents (primarily documents and proxies) to interact with the hardware and the human users through a (fairly) common and abstract set of operations. There may be Jini-enabled devices in the network, in which case the terminal service only has to act as an interface gateway. There may also be non-Jini devices (as is the most common case today) in which case the appropriate services must be constructed.

2.1.7 Subsystem Services

This is a set of Jini services which act as a general library to the clients. The subsystem services provide data formatting services, data format conversion, storage, compression and whatever else may be generally needed in a distributed environment.

2.2 Interactions

An active document makes contact with a proxy by obtaining the proxy's service object from a Jini lookup server. Through the service object, the document negotiates with the proxy for a way to interact with the user. For their communication they first negotiate for a common language. Java method calls are appropriate for many tasks, but agent communication languages are likely to be represented as well.

The proxy, in turn, must then find a way to communicate with the user. To do this it contacts one or more terminal services which represent devices close to the user. The terminal services of choice (and the devices they represent) fall into two rough categories:

- devices with an expected proximity to the user, and
- devices with random proximity to the user.

Devices with an expected proximity are those which the user is known to patronize, i.e. a certain PDA or laptop computer, a specific office computer, a mobile or stationary telephone etc. The estimated probability of reaching the user on any of these devices can be calculated by the proxy using statistics of past behaviour, the time of day, calendar information submitted by the user or other available sources. Depending on the level of sophistaction implemented in the proxy, the computed estimate may be produced by a user-configurable lookup table, machine learning techniques, or a combination thereof. A proxy which learns and adapts hopefully needs less attention from the user.

Devices with random proximity are those which by chance happen to be close to the user at some time. For example, when the user walks into a conference room the equipment there represents potential contact points (let us for the moment ignore any conflicts with issues of security and privacy). To exploit this random proximity it must be possible to ascertain the position of users and devices.

Since devices are increasingly mobile their identity and location is just as important as that of their users. Coupled to this is the possibility of public terminal services. For example, a user of a laptop computer may be willing to export a public terminal service for a virtual device which implementes a small text window while reserving full GUI interaction for the user's personal proxy. To extend the conference room example above, the availability of possible means for contacting the user is extended to include not only the stationary equipment in the room, but also the public terminal services in mobile devices brought by other users.

For active documents that need to interact closely with the user, the concepts discussed for terminal services can be extended to mobility servers, i.e. execution environments with certain properties.

3 Agents and Jini

3.1 Impact for Agents

Jini offers a framework in which the participants described above may find each other without prior knowledge of hosts, ports, properties etc. This is a great strength in a dynamic environment where both users and mobile equipment can be expected to connect to the network in a random fashion and change locale and context several times per day.

The way a Jini client queries a lookup server may or may not be adequate for agents to make the necessary qualifications in their selections of a service. If it turns out that potential communication partners must be contacted and interrogated the process may quickly become cumbersome and not scale well.

3.2 Services

We look again at four central components that appear as Jini services in our infrastructure:

- User proxies
- Terminal services
- Localization service
- Mobility servers

User proxies should be persistent and available, so the most natural habitat for a proxy is a department server or stationary personal computer. A user proxy registers itself as a Jini service and is thereby available to its clients. The expected clients are active documents and their requests generally result in some kind of interaction with the user. The interaction is either mediated through a proxy or conducted by the application code implemented in the document itself.

Now, if an agent can interact directly with the user, how can the proxy enforce its decisions? What is there to prevent a rouge agent from bypassing the proxy and harassing the user? Obviously some kind of security mechanism is needed. One alternative is that all communications must pass through the proxy, but this appears clumsy. Another way would be to restrict access to localization information to trusted parties, like the proxy. Likewise, access to terminal devices could be restricted by terminal services which require authentication.

A terminal service registers itself as a Jini service. The terminal service represents a piece of identifiable hardware and it would therefore appear natural that it executes on or close to the hardware for which it is responsible. The terminal service is not necessarily in full control of the hardware as this depends on the underlying security configuration.

A terminal service for a virtual device may be sufficiently general to be able to implement its service using the capabilities of other terminal services. For example, a printer spooling system, CD-ROM virtual jukebox or simple messaging broadcast system. As such, the terminal service may itself be persistent and decoupled from the actual hardware. The service should be reachable although the set of supported and available physical devices vary.

The localization service is a Jini service which clients contact to make queries or updates. Proxies are expected to be frequent query clients. Processors of stationary sensors are expected to be updating clients. Further updates may arrive from clients in mobile equipment. Updates may also arrive through clients which act as protocol converters for platforms which cannot run clients.

Mobility servers are represented as Jini services and contacted the same way. Negotiation and entry is facilitated through the service object.

4 Jima and the Meeting Scenario

Since user proxies act as destinations, the message sequencing problem mentioned for the meeting scenario above may be addressed there. In its most simple form, each document that arrives at the proxy presents itself with (at least) a session key and a private data structure. The session key can be anything which is hard to fake and has a low probability of random reproduction. The private data structure is something which is meaningful only to the agents participating in the session. The proxy may or may not choose to remember the session keys and data structures, but if it does it is able to give the document an information trace left by earlier documents in the same session.

A new document then inspects the history and determines to what extent it should impact on its behaviour. Following the meeting scenario, a cancellation message may discover that there is a hole in the sequence, but since it is a cancellation message it can safely ignore this. When a late reminder message arrives it can in turn safely decide to abstain from delivery and immediately expire.

Materials in the meeting will obviously benefit from a localization service if they are to be available in the right place at the right time. As an alternative to printing, documents may download themselves into the public execution environments of the participant's mobile computers. This may present an interesting conflict when a previous version of the same document already is present. Will the document agent in the public space be able to detect this? If it does, should it be able to replace the old version with itself? It may be necessary to introduce a concept of *document integrity* which effectively moves some control from the user to the document. It is quite clear that a document should be able to protect itself from simple manipulation, but it may also be viable to allow the document the right to enforce these properties anywhere. Just because the document is running in someone's personal computer it should not automatically transfer all control to the person controlling the computer. Although it does, in practice, it may help to strive for a common ground where users and documents can meet on equal terms. This implies that documents do not upload themselves undiscriminantly, but have demands and requirements on both their presentation and execution environments.

5 Related Work

A nice treatise on mobile agent programming systems by Karnik and Tripathi [7] illuminates many important issues which this paper omits or glances by. For example, the desired properties of mobility servers (execution environments) and the mobile code itself, the namespaces to be used and how names are resolved (into Jini service templates, for example), security issues, agent monitoring and control and fault tolerance.

The Mobile People Architecture (MPA) [1, 9] embodies many ideas similar to our own, in particular the personal proxy concept. Our approaches differ on the way terminal devices are accessed. The MPA places *application drivers* in the proxy while our infrastructure assume that terminal access is distributed. Another difference is that MPA emphasizes communication between persons, while we assume communication mediated by active documents.

We hope that the dynamic availability of terminal services in our infrastructure may be a useful element in the development of collaborative environments like *Information Workspaces* [5] at Stanford University, or virtual workspaces [8, 10, 3]. In particular, we expect that transparent blending of mobile and stationary equipment will become increasingly important.

We choose to indicate only a few sources for information about Jini [6, 2, 4] as this is a developing area.

6 Summary

We have described an infrastructure based on Java and Jini for active documents in the form of mobile agents. As a case for our studies we have choosen a meeting scenario where information and resources are manipulated by a group of co-workers, and where each component must be on hand in a timely fashion. Users are persistently represented in the network by personal proxies. Proxies communicate with users by invoking terminal services, the choice of which is based on an estimate of the user's location relative to a particular terminal device.

Agents use Jini to discover and negotiate with other agents. In the Jini framework, agents thus appear as both clients and services. Some services (like proxies) are persistent within the local network, while others (like terminal services) may appear and leave at random.

To dynamically associate users with terminals a localization service is helpful, but not critically essential.

References

[1] Guido Appenzeller, Kevin Lai, Petros Maniatis, Mema Roussopoulos, Edward Swierk, Xinhua Zhao, and Mary Baker. The mobile people architecture. Tech-

nical Report CSL-TR-99-777, Computer Systems Laboratory, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, California 94305-9040, January 1999.

[2] Ken Arnold, Bryan O´sullivan, W. Scheifler, Robert, Jim Waldo, and Ann Wollrath. *The Jini (TM) Specification.* Addison-Wesley Publishing Company, 1999. ISBN: 0201616343.

[3] C. Davis, Richard, A. Landay, James, Victor Chen, Jonathan Huang, B. Lee, Rebecca, C. Li, Francis, James Lin, B. Morrey III, Charles, Ben Schleimer, N. Price, Morgan, and N. Schilit, Bill. Notepals: Lightweight note sharing by the group, for the group. In *CHI 99 Conference Proceedings, Human Factors in Computing Systems*, pages 338–345, 1999.

[4] W. Edwards, Keith. *Core Jini.* Prentice Hall, 1999. ISBN: 013014469X.

[5] Interactive workspaces. http://graphics.stanford.edu/projects/iwork, 1999.

[6] Jini (tm) connection technology. http://www.sun.com/jini, 1999.

[7] M. Karnik, Neeran and R. Tripathi, Anand. Design issues in mobile agent programming systems. Technical report, Departmen of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA, June 1998.

[8] Jun Rekimoto and Masanori Saitoh. Augmented surfaces: A spatially continuous work space for hybrid computing environments. In *CHI 99 Conference Proceedings, Human Factors in Computing Systems*, pages 378–385, 1999.

[9] Mema Roussopoulos, Petros Maniatis, Edward Swierk, Kevin Lai, Guido Appenzeller, and Mary Baker. Person-level routing in the mobile people architecture. Technical report, Department of Computer Science, Stanford University, Stanford, California 94305, 1999.

[10] Jason Stewart, B. Bederson, Benjamin, and Allison Druin. Single display groupware: A model for co-present collaboration. In *CHI 99 Conference Proceedings, Human Factors in Computing Systems*, pages 286–293, 1999.