Providing location transparent services with Java technologies

Testing support of Open Distributed Processing transport transparencies in JINI, Web Services and JXTA

Rune Fauske
Per Wollebæk

Agder University College
Grimstad 2002

Abstract

This thesis tests the presence of Open Distributed Processing (ODP) distribution transparencies in Jini, Jxta and Web Services. The thesis presents an introduction to each technology, a description of the test criteria developed, selection of prototype applications, test execution and presentation of the results. Main focus is on testing support of transparencies at the application developer and system designer level.

The work has shown that all technologies support transparence types that have to do with the location of a service; location transparency and migration transparency. But when it comes to replication transparency, this is not supported by Web Services. Jini provides functions to achieve failure transparency in the JavaSpace extension; neither Web Services nor Jxta provides this.

Preface

This thesis is part of the Graduate degree (Siv.ing) in Information and Communication Technology (ICT) at Agder University College, Faculty of Engineering and Science in Grimstad, Norway. This work has been done in cooperation with ObjectNet AS and the R&D project “Den Mobile Student”.

We would like to thank our employer Objectnet AS for providing two experienced supervisors; Nils Chrisian Haugen and Stein Grimstad.
They both have given us advice and guidance through the project, even at the cost of their own spare time.

We would also direct a special thanks to Stein Bergsmark at Ericsson AS for reviewing of our work and Jan P Nytun at Agder University College for helping arranging this thesis.

Rune Fauske and Per Wollebæk, Grimstad, May 2002

Contents

1 Introduction

1.1 Thesis introduction

We want to communicate and share resources. This appears to be the main motivation to build and use distributed systems. The most common example of a distributed system is the Internet. Though it's simple it's not poor, since anyone nowadays has an opinion of what it is: Communicating and sharing resources.

There have been a lot of descriptions which try to describe what a distributed system is. To give a simple description, distributed systems are a collection of separate computers which are comprehended as a single system.

Distributed systems are often organized as layers of software. Middleware separates the application from the operating system, and creates a platform for distributing services.

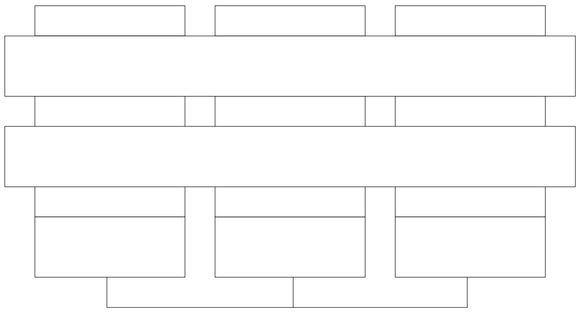


Figure 1-1 Distributed system as middleware [18]

One of the ambitions of a distributed system is to hide the fact that it is physically distributed. If we look at the definition of a distributed system mentioned above, we could say that the system is transparent, since the system appears as a single system to the client. Transparency is one aspect of the distributed system which is hidden from the user[1]. Transparency must be provided by a function in a layer below the layer of where it is needed. If our application needs a function that provides transparency, then it must be implemented in the layer below; the middleware service (Figure 1 1).

Leslie Lamport has this slightly humorous reflection to distributed systems and transparency.

> *"You know you have one when the crash of a computer you've never heard of stops you from getting any work done"*

Transparency is not preferred in all situations. Distributed processing introduces many new problems to a system. A high level of transparency will most likely result in performance loss, and sometimes we don't want to hide the distribution aspect.

The main motivation for transparency in distributed systems is to reduce implementation complexity, and raise usability. Another important aspect is separations of concerns.

[1] The user could be the programmer, system developer, user or the application program.

Open Distributed Processing – Reference Model (RM-ODP) is a standard for Open Distributed Processing. It creates an architecture within which support of distribution, interworking and portability can be integrated [8]. The distribution transparencies which we look at in this thesis are standardized in this ISO document.

Max Goff at Sun Microsystems has performed a critical comparison of the technologies Jini, JXTA and Web Services in the scope of the *eight faculties*[2] [22]. We'll take this testing a step further, focusing at several RM-ODP distribution transparencies and verify if the selected technologies conform to them. The comparison is performed by both an application test based on simple prototypes, and discussions of the technologies supporting the transparencies.

[2] [2] The eight faculties [27] are a set of assumptions often made by distributed software developers, which in the long run are proved to be wrong.

1.2 Task description

In distributed systems a separation from a static resolution of resources is wanted. To achieve this, a dynamic registry of resources is needed. It must be able to separate the location independent name from the actual reference to the resource.

An example of this is accessing a web server. Knowing the static location of the server is in many cases not necessary, an URL is enough. After transmitting the address it will be translated by the DNS to an exact address and the request will be forwarded to right location.

Some distributed technologies like JINI, JXTA and Web Services offer similar solutions to provide resources such as hardware or software by dynamic addressing.

The assignment will be both a theoretical and a practical comparative study of different java technologies for providing location-transparent services including locating and providing resources.

The theoretical part will include:

- An introduction to location transparent services
- Surveying of technologies (description of three distributed system technologies)
- Identify criteria for testing the technologies
- Identify an adequate application for an evaluation
- Evaluate different solutions for distributing services

The practical part will include:

- Implementations of prototypes in different technologies
- Test runs of prototypes.

Technologies

- Technologies which are interesting to compare and study are JINI, JXTA and Web Services. They are all offering service providing but have different approach to the problem.

1.3 Thesis outline

The next three chapters of this thesis provide an introduction to the technologies which were tested: Jini, JXTA and Web Services.

To have a proper test, the test criteria and test definitions based on the ODP distribution transparencies are developed and described in chapter 5.

A set of application prototypes have been developed for each technology. These applications are described in chapter 6 along with the tools used in the development process.

In chapter 7 each of the test cases are performed at each technology. This chapter includes description of the execution, presentation of results, discussions of each test and finally a conclusion for each test.

The next part discusses important observations from the test in previous chapter.

Finally, a conclusion from the application tests and discussions are reached and presented in chapter 8.

2 Jini

2.1 Overview

This chapter provides an introduction to the Jini technology. We start with a brief history review and introduction to network structure and communication protocols. Then there is an introduction to a service built on Jini architecture; the JavaSpace.

2.2 What is Jini?

Jini technology was established in Sun Microsystems Laboratory at Sun east cost facility just outside Boston January 25, 1999.
It is a distributed computing environment that can offer “network plug and play” and a user-friendly access to information.
Jini is based on the Remote Method Invocation (RMI[3]) infrastructure but gives the opportunity to make much more advanced distributed applications.

In an article in Dr. Dobb’s journal [12], Bill Pierce writes that “Jini puts RMI on Steroids” and gives a detailed description of how RMI is corrected under the implementation of Jini. Examples of such are; lookup process is advanced from a string based Naming service to a lookup specification that uses java concept of interfaces, which again allows much more complex lookup queries to be built.

While the RMI implementation is bound to network address which the client must know beforehand, the Jini infrastructure is using IP multicast which eliminates this constraint and allows lookup services and services to be found without any knowledge of where those services reside.

RMI’s naming service allows only stub references to Remote object to be stored. Jini Lookup services can store both references to remote services and serializable service object that can be downloaded and executed locally on the client.

Finally the RMI structure has no established mechanisms for a service object to tell when its client has failed, leaving them holding resources for clients that no longer exist. Jini has solved this with a leasing mechanism.

Jini is built on top of a solid foundation of Java networking technologies. Java networking itself has been available as part of Java since version 1.0 of the JDK release. In fact, Java is the first widely available language/platform that was designed with networking in mind right from day one.

Jini was conceived as the foundation upon which robust, truly distributed systems can be built. Jini is nothing without a network – in fact, by definition; Jini doesn’t exist outside of a network.

2.3 Jini networking

The Jini network consists of services. Applications are created to combine the services to groups called federations. In a Jini network there may be many federations, each one offering specific services.

[3] To read more about RMI see (10.05.2002)
http://searchsolaris.techtarget.com/sDefinition/0,,sid12_gci214267,00.html

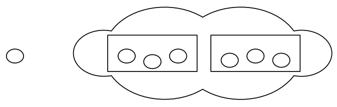


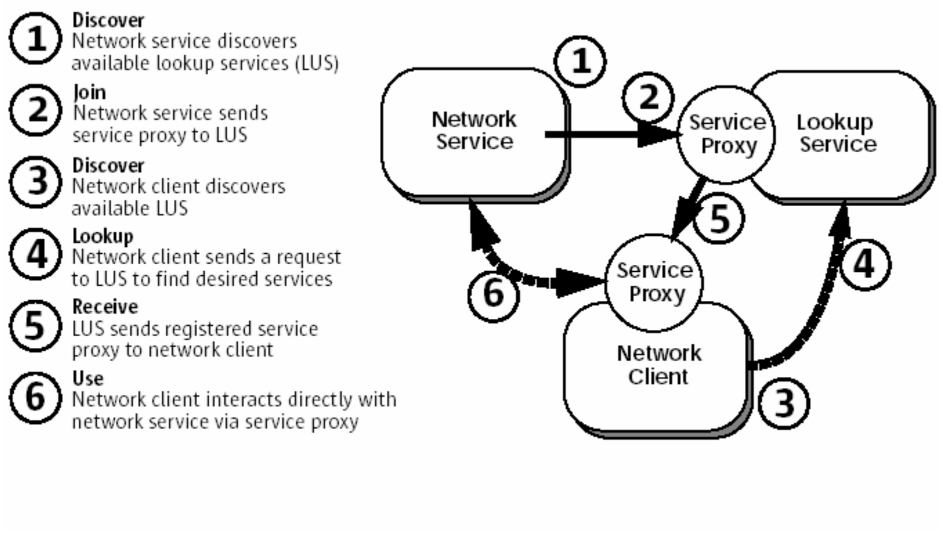
Figure 2-1 Jini services in Federations

Inside a federation a client can search for services through an intermediary service called a lookup service. There is always at least one lookup service available in a Jini network, most cases there will actually be more than one to implement a basic level of redundancy and fault tolerance.

The search process is usually initiated by a client telling the lookup service what type of function (or work) it wants the service to perform. This is done with a Java interface or attached properties. The lookup service will return a service within the federation that fulfils the criteria.

Any service in the Jini network may make use of other services in performing its own work. This way a service may also be a client of another Jini service.

In a running Jini system, there are three main players; Lookup service, the service and the client. To give a clearer indication what the different players are doing, a short description will be given below together with a description of how they are interacting with each other.



1 Discover
Network service discovers available lookup services (LUS)
2 Join
Network service sends service proxy to LUS
3 Discover
Network client discovers available LUS
4 Lookup
Network client sends a request to LUS to find desired services
5 Receive
LUS sends registered service proxy to network client
6 Use
Network client interacts directly with network service via service proxy
Network Service
Service Proxy
Lookup Service
Service Proxy
Network Client

Figure 2-2 How Jini works, a flow diagram [44]

2.3.1 Service

Jini is designed to support a wide variety of hardware platforms, and to bring together hardware ranging from the very cheap (under a dollar) to the very expensive. (Million dollar servers) This makes a definition of a service very wide, and can at best be described as any hardware or software component that can be used by other components.

The book Core Jini [9] (page 93) gives an indication of what kind of requirement (hardware) which the component must support before it can be used in a Jini network.

No matter what type of machine or software component you will mount to a Jini network it has to register at a lookup service to be available to the Jini community. The registration is initiated by a discovery process.

If there is not implemented some restrictions, a service will join all the lookup services that is supported by the community. So if one lookup service fails, others can stand in for it.

When a service wants to register with a lookup service it will typically upload a proxy object. The client will later download this object from the Lookup service.

A proxy object will work as a middleman between the service and the client. The communication platform is option by the developer (RMI, CORBA (Common Object Request Broker Architecture), SOAP (Simple Object Access Protocol), etc)

2.3.2 Lookup Service

As mentioned before a service must register to a lookup service if it wishes to join the Jini community.

A lookup service is a store which contains “service items”, each of them are again containing the proxy for a service, a unique ID for the service, and a set of descriptive “attributes” associated with the service. You can think of Jini Lookup service as being like name servers, since they keep track of all of the services that have joined a Jini community. But unlike a traditional name server, which provide a mapping from strings to stored objects, the Jini lookup service provides a much richer set of semantics. For one thing, the lookup service understands the Java system, so you can search for proxies that implements particular Java interfaces and the lookup service will return you the proxies that implements these interfaces. You can even search for superclasses and superinterfaces of proxies.

2.3.3 Client

A client is a process that wants to use one or more services. (Note that this client can also be a service in the Jini community) It will announce its presence to a lookup service and start searching for services in the community.

If the lookup service doesn’t have access to a service that the client seeks, the lookup service can store information about the interface. If the lookup service later gets a registration from such a service, it will then announce this to the client by sending it an event.

On the other hand if the lookup service has access to the wanted service the lookup service will upload the proxy object to the client, and the communication between the service and client can start.

A client and service are extremely loosely coupled, and this allows Jini to survive fault to service and LUS. A client that needs work to be carried out doesn’t need to use the same service again, unless it is the only one on the network.

2.4 Jini network communication protocols

2.4.1 Discovery

The discovery and join protocols govern the way individual computers join, leave and interact with a Jini federation. A federation is the collection of Jini clients and services that come together to get work done. (Also referred to as djinn, or Jini community)

Broadly there are two types of discovery in Jini networking. One form is used to support “serendipitous” interaction between services and lookup services. Serendipitous interaction means that the lookup service and Jini service find each other without any previous configuration and without explicitly being told to search for one another.

This form of discovery is used when a service starts and needs to find all the lookup services that may be running in its neighborhood. It is also used when a lookup service starts and needs to announce its presence to any Jini services that may be running, in case they wish to register with it.

The second type of discovery is used to connect a Jini service to a certain lookup service. Unlike the serendipitous form of discovery, where services find any and all lookup services in their neighborhood, this direct form of discovery allows services to contact particular lookup services that they may know about ahead of time. This form of discovery uses an URL-based naming scheme for specifying lookup services.

Since we want to look at transparent solution in distributed networking the serendipitous way of discovering is the most interesting.

2.4.2 IP Multicast

The network technology used in the serendipitous discovery process is IP multicast. Multicast is a way to send messages that may be received by any number of parties.

It is not broadcast, because it does not go to every party on the network; it only goes to those who are explicitly listening for it. And it is not unicast, which would entail sending a separate message to each intended recipient, and would require that the sender know the IP address of each recipient ahead of time.

Multicast makes efficient use of network resources, because a message sent to multicast receivers it transmitted as a single message wherever possible.

The form of multicast used by Jini is based on the UDP/IP protocol. UDP provides a connectionless, unreliable transport. This means that UDP provides no guarantees that a packet will arrive or in which order they are arriving if there is more than one sent.

In UDP multicast, a range of special IP addresses are used as multicast groups. Interested parties can effectively “join” a group by listening for messages sent to that IP address. All parties listening to that address would receive any message sent to it.

Each message sent with multicast has a scope associated with it, which is used to limit the distance the message will travel. The benefit of scoping is that an IP address used as a multicast group for one set of hosts may be invisible to another set of hosts using the same IP address. In this way, scoping effectively limits the visibility of multicast groups, and allows the addresses used for multicasting to be reused many times across the Internet.

2.5 JavaSpaces

Many applications in a distributed setting, just as in local settings, have a need to share data. Jini has a solution to this demand, and it is in a service built on top of the basic Jini substrate. This service is called JavaSpaces and provides storage facilities for java object; in fact it can hold only Java objects.

In many ways JavaSpaces can be compared to a network file system, where Jini services and clients can create and manipulate persistent collections of java objects. - In other words JavaSpaces provides a facility for distributed shared memory that can be used by any other entity in a Jini community.

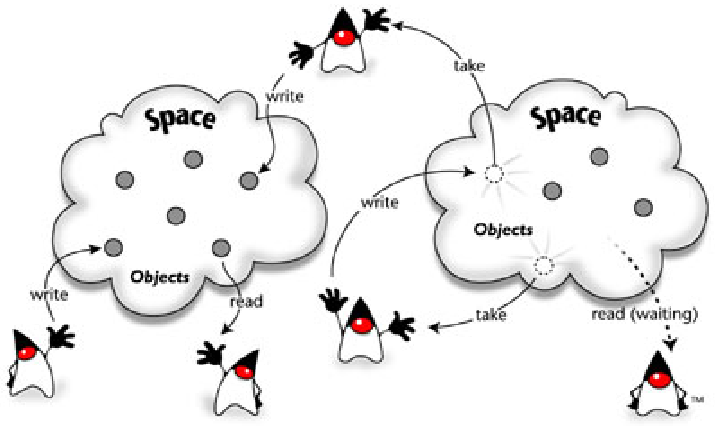


Figure 2-3 Example of interaction in a JavaSpace

JavaSpace is based on Gelernter Linda system [46]. An alternative technology to JavaSpace is the IBM T Space [47]

JavaSpace defines only three different actions to interact with the space, each having two modes of operations: read, write and take. They implement blocking and non-blocking semantics. There is also and asynchronous notification registration method available, notify(), that utilizes Jini remote events.

2.5.1 The write Operation

The write method is used to put a new entry into JavaSpace. The object is serialized and a copy of it is stored, together with a RMI codebase annotation that can be used to retrieve the class information on the entry. This information allows the object to be re-constituted for any reader of the JavaSpace.

If you write the same entry multiple times, there will be multiple copies of the entry residing in the space. A successful write operation returns a lease, which the writer must renew in order to keep the written entry in the space. On the other hand if a remote exception is raised you cannot tell if the write was a success or not.

2.5.2 The read operation

With the read operation will the JavaSpace attempt to mach a user-filled template against the entries held in the store. Using an associative search, a copy of the appropriate entry will be retrieved and returned to the reader of the space. Any null fields in the template entry act as wildcards during the mach operations.

Note that JavaSpace API provides no way to return multiple values from a search.

The read method uses blocking, in other words will the caller be blocked until ether a matching entry becomes available, or a specified time period has elapsed.

The readifExist() method provides a non-blocking variant that reads entries only if an immediate mach can be made.

2.5.3 The Take operation

The take operation works much like the read operation, with the difference that it removes the object from the space.

If more than one client attempts to take the same object from space simultaneously, one of them, and only one, will successfully match and retrieve it. JavaSpace will not guarantee that the first one to give a request will be the one to retrieve the object.

2.5.4 The notify operation

The notify operation work much the same way as the method of the same name on Jini Lookup service. The method takes an Entry as a template that will be matched against future writes to the JavaSpace. If a new Entry is written that matches the template, an event will be sent to the listener specified in the notify call. The client then have to make an read or a take operation,- there is, however, no guarantee that the matching entry will still be in the space by the time the client perform this operation.

2.6 Summary

In this chapter we have learned that Jini is built on RMI and that is offers a transparent environment for developing distributed applications in Java. The discovery process offers two different options of discovery, serendipitous and a direct connection. The serendipitous uses multicast, the other alternative is an URL-based naming scheme for specifying the lookup services.

We have also learned that JavaSpace is a service built on the Jini structure, that offers a tuple-based environment for sharing objects.

3 JXTA

3.1 Overview

This chapter provides an introduction to JXTA technology, starting with a brief history review and introduction to network structure as well as communication protocols.

3.2 What is JXTA?

JXTA, or Juxtapose, began in the summer of 2000 as a Sun Microsystems research project or "intrapreneuring incubation" led by Chief Scientist Bill Joy. Beginning with conversations with innovators in the Peer to Peer (P2P) space, the JXTA team began assembling a picture of what a core common distributed computing framework would look like. In the months since, the sketch became an API, and finally an Open Source release on April 25th, 2001 [41].

P2P computing is becoming increasingly popular. Software based on P2P technologies includes Napster and Gnutella for file sharing, SETI@home for distributed computing, and AOL and other instant messenger services. While each of these applications performs different tasks, they all share many of the same properties, such as discovery of peers, searching, and file or data transfer.

What is missing is a standard set of interfaces on top of which developers can build P2P applications. The direct consequence of this is that currently, application development is inefficient, with developers solving the same problems and duplicating similar infrastructure implementation.

JXTA strategy is one of complete openness and standards conformance, from design to XML-based protocols to open-source implementation. JXTA is lightweight and minimalist, providing only what's vital across P2P applications. It leaves everything below (operating environment, TCP/IP network-awareness, and hardware) and above (application specific) to the device/environment and application at hand. This leaves only the building-block constituents that almost all P2P applications can use, regardless of their intended users and specific implementation [48].

Sun has provided an initial Java language implementation for JXTA; but, amazingly enough, Project JXTA is specific neither to the Java programming language nor to the Java platform. In other words, anyone can implement JXTA-based networks on any hardware platform, with any operating system, using any programming language [49].

3.3 JXTA software architecture

The Project JXTA software architecture is divided into three layers, as shown in Figure 3-1.

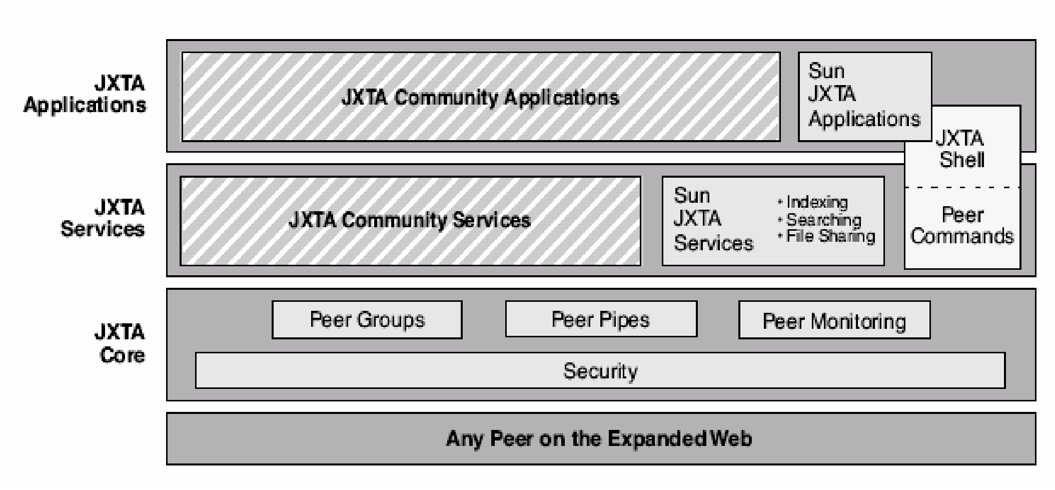


Figure 3-1 Project JXTA software guide [42]

3.3.1 Platform Layer (JXTA Core)

The platform layer, also known as the JXTA core, encapsulates what is common to P2P networking. It includes building blocks to enable key mechanisms for P2P applications, including discovery, transport (including firewall handling), the creation of peers and peer groups, and associated security primitives.

3.3.2 Services Layer

The services layer includes network services that may not be absolutely necessary for a P2P network to operate, but are common or desirable in the P2P environment. Examples of network services include searching and indexing, directory, storage systems, file sharing, distributed file systems, resource aggregation and renting, protocol translation, authentication, and PKI (Public Key Infrastructure) services.

3.3.3 Applications Layer

The applications layer includes implementations of integrated applications, such as P2P instant messaging, document and resource sharing, entertainment content management and delivery, P2P e-mail systems, distributed auction systems, and many others.

The boundary between services and applications is not rigid. An application to one customer can be viewed as a service to another customer. The entire system is designed to be modular, allowing developers to pick and choose a collection of services and applications that suits their needs.

3.4 JXTA networking

The JXTA network consists of a series of interconnected nodes, or peers. Peers can self organize into peer groups, which provide a common set of services. Examples of services that could be provided by a peer group include document sharing or chat applications. JXTA peers advertise their services in XML documents called advertisements. Advertisements enable other peers on the network to learn how to connect to, and interact with, a peer’s services. JXTA peers use pipes to send messages to one another. A pipe is an asynchronous and unidirectional message transfer mechanism used for service communication. Messages are simple XML documents whose envelope contains routing, digest, and credential information. Pipes are bound to specific endpoints, such as a TCP port and associated IP address.

3.5 Summary

What we have learned in this chapter is that JXTA is a Peer to Peer (P2P) platform which is built on XML technology. It is not bound to a specific developing platform, and it is by so possible to be deployed on any device with a digital heartbeat. The task which JXTA is to fulfill is to bring all the P2P applications together under a common platform for communication and message exchange.

4 Web Services

4.1 Overview

This chapter gives an introduction to Web Services in the scope of business-to-business interaction in XML messages. We are looking at the technology stack and describing each element of it. This introduction should give the reader a common understanding of Web Services before the concrete tests are performed.

4.2 What is Web Services?

“A web service may be defined as: An application component accessible via standard web protocols.” [1]

The only requirement to a web service is that it must be able to send and receive messages with a combination of different web protocols. This indicates that web services are nothing new, but only an evolution of principles [2].

4.3 Web Services model

The conceptual web services model could be separated into three different roles.

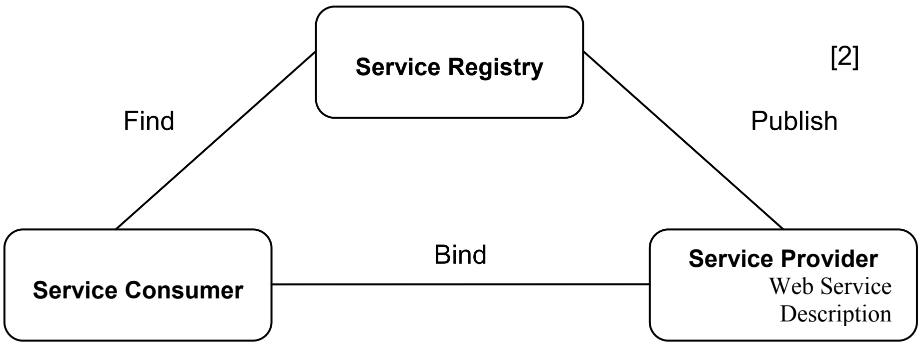


Figure 4-1 Conceptual Web Services model

Service Provider
The provider is the entity that creates the web services; typically a business function is exposed through a web service.

Service Consumer
The consumer is the entity that uses the service. The consumer might have prior knowledge to the service API, or download a description from a URL.

Service Registry
Service Registry is a central location where service providers can register and list its Web Services. The registry typically contains business information, what Web Services the business provides and details about each Web Service.

A standard protocol to publish or find Web Services is The Universal Description, Discovery, and Integration (UDDI) [17].

A standard protocol for applications to bind to Web Services is The Simple Object Access Protocol (SOAP) [19]. SOAP is a language and platform independent mechanism for exchanging information between applications based on Extensible Markup Language (XML) [21] messages.

A standard way of describing Web Services is The Web Services Description Language (WSDL) [20]. A WSDL would describe methods, parameters, data types, transport protocol and the web service's access point(s).

4.4 Web Services Stack

In this chapter well look closer to each of the standards that are part of the web services technology stack.

Service Publication/Discovery	UDDI
Service Description	WSDL
XML Messaging	SOAP
Transport Network	HTTP,SMTP,FTP,HTTPs over TCP/IP

Figure 4-2 Web Services technology stack [5]

4.4.1 Transport Network

Web Services are built on existing communications standards witch makes them transport-independent, and web services might be built on top of nearly any transport protocol. The primary role of the transport layer is to move data between two or more locations on the network.

The choice of transport protocol is based on the transportation need of the Web Service being developed.

4.4.2 XML Messaging – SOAP

This layer defines the message format used for application communication between Web Services. The most commonly used protocol is SOAP, which makes inter-application communication possible regardless of operating system, programming environment and object model.

The communication is done using the text-based XML protocol instead of a binary protocol used by CORBA, RMI and DCOM. This makes SOAP highly interoperable across hardware platforms, operating systems, programming languages and network hardware platforms. SOAP doesn't define the semantics of the messages; it only provides a framework for it. Semantics will in most cases be application dependent [5].

4.4.2.1 SOAP Message

A SOAP message consists of an envelope containing an optional header and a required body. The header contains information of how the message is to be processed. The body contains the actual message to be delivered and processed.

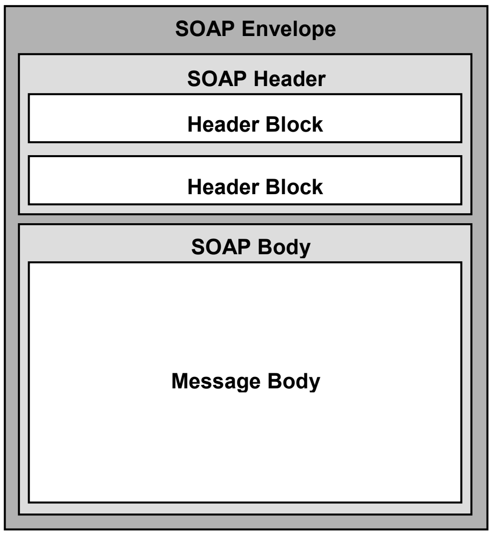


Figure 4-3 SOAP message

The XML syntax of a SOAP message is based on the soap-envelope namespace [4] defined by W3C [6].

4.4.2.2 RPC Messages

Typically RPC messages come in pairs as the request and the response. SOAP doesn’t require a response to every message, but it’s common to have the request-response pairing.

An example of a request message querying the method

```
  Public Float getQuote(String symbol);
```

```
<s:Envelope
    xmlns:s=”http://www.w3.org/2001/06/soap-envelope”>
  <s:Header>
<m:transaction xmlns:m=”soap-transaction”
s:mustUnderstand=”true”>
  <transactionID>1234</transactionID>
</m:transaction>
  </s:Header>
  <s:Body>
<n:getQuote xmlns:n=”urn:QuoteService”>
  <symbol xsi:type=”xsd:string”>
    IBM
  </symbol>
</n:getQuote>
  </s:Body>
</s:Envelope>
```

The response message could be something like this.

```
<s:Envelope
    xmlns:s=”http://www.w3.org/2001/06/soap-envelope”>
  <s:Body>
<n:getQuoteResponse xmlns:n=”urn:QuoteService”>
  <value xsi:type=”xsd:float”>
    98.06
  </value>
</n:getQuoteResponse>
```

```
  </s:Body>
</s:Envelope>
```

4.4.3 Service Description – WSDL

This layer provides a mechanism for describing the capabilities and functionality of a Web Service. WSDL is XML grammar for describing the Web Service. A WSDL document describes endpoints and ports on both document-oriented and procedure-oriented messages. WSDL provides information about the Web Service in two levels, high-level transport protocol independent abstract definition of a Web Service, and second is a transport dependent binding description [5].

WSDL Specification	
Binding	Abstract Definition (Reusable part)
PortType	
Messages	
Types	
Service	Implementation Specific
Port	

Figure 4-4 WSDL document overview

Port Type	The portType element describes the different operations exposed by the Web Service.
Messages	The message element contains description of the data that is transmitted. This is analogous to method parameters.
Types	The types element contains the data types that is present in the message.
Bindings	The binding element maps operation element in a portType element to a specific protocol.

4.4.4 Publication and Discovery – UDDI

In stead of providing information about each Web Service to each client, the information could be published into a centralized registry publicly available to interested consumers.

UDDI provides a specification for registering business information, services and technical specification to each service into the same registry.

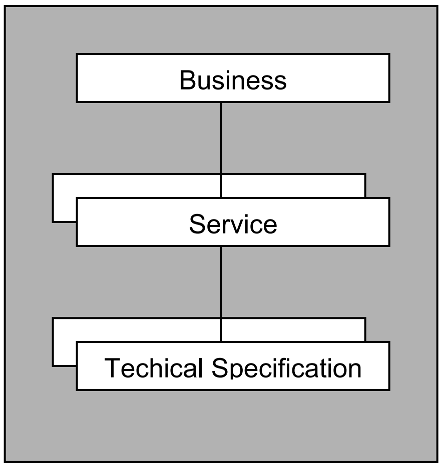


Figure 4-5 Simple UDDI registry overview

Business	This contains the business information like name, contacts, address and so on. A classifications system in UDDI allows each business to describe what kind of business they are involved in.
Service	A structure which describes each Web Service this business provides. Services might be categorized into categories. Each service will have a corresponding technical specification.
Technical Specification	This section contains technical data about a service such as a WSDL document.

4.5 Summary

In this chapter we have learned that Web Services is nothing new, only evolved principles. UDDI can be used to register and lookup Web Services by name or based on Web Services descriptions like WSDL. SOAP is used to send XML messages to and from Web Services over the HTTP protocol.

5 Test Criteria

5.1 Overview

This chapter will give an introduction to the test criteria. They are divided in two separate categories: Technical criteria and Market review.
Each criterion will be given a short description for easier understanding their relevance.

5.2 Introduction

Location transparency is one of the distribution transparencies defined in ODP-RM ISO standard [8].

The topic for this thesis is to provide location transparent services with Java. This could be simple. By referring a service by its Domain Name System (DNS) name the location of the service is transparent since we are hiding the service's IP-address from the client. But in this thesis we will look at a higher level of location transparency, since objects tend to change regularly.

In order to achieve this, the technology must support some kind of naming system to keep and maintain service locations, and separate them from static client bindings. This leads to several new test criteria. The technologies selected for this comparison are already known to have location transparent object models.

There are several other interesting aspects when providing location transparent services. When is a service location transparent? How will the client locate the service? What if the service moves? What if the service fails? Could we use several identical services to raise availability? Four criteria have been selected based on these questions.

5.3 Technical test criteria

Each of the criteria developed for this thesis will be described thorough in the next chapters.

5.3.1 Service location transparency

DNS provides naming for systems with more or less fixed locations, but this type of naming system is poor in name-to-address bindings for objects that change locations regularly [18].

To support service discovering and location transparency, the system should offer a naming facility which hides all aspects concerning the location of a service. Naming systems provide different types of lookup mechanism. Some only support lookup-by-name, where others resolve objects based on state or semantics.

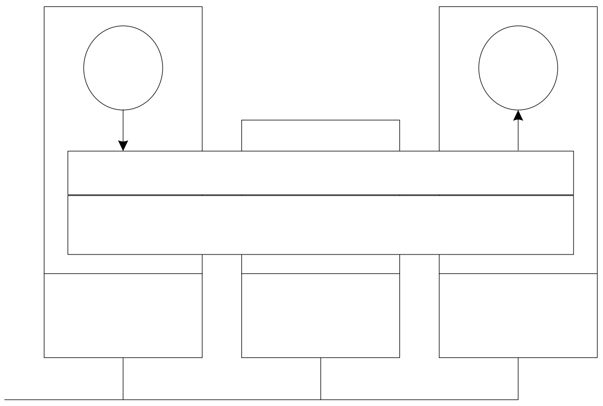


Figure 5-1 Location transparency

In Figure 5-1 the client c_1 uses the naming system (part of the middleware) to locate the service before the actual invocation. The client has no knowledge of the fact that service s_1 is located at computer 3.

In addition to just lookup, it's important for a naming system to remove non-existing objects from the registry to keep it consistent [18].

The test section will answer the following questions in order to verify that the system support lookup of services before invocation.[4]

1. Does the technology provide naming mechanisms to achieve service discovery functionality?
2. What type of lookup is supported?
3. Does the naming service deal with the problem of unreferenced objects?
4. How does the naming system scale in large distributed systems?

5.3.2 Service migration transparency

"Migration Transparency: A distribution transparency which masks, from an object, the ability of a system to change the location of that object …" [8]

Migration Transparency hides that a service can be moved to another location. Migration is often used to achieve load balancing.

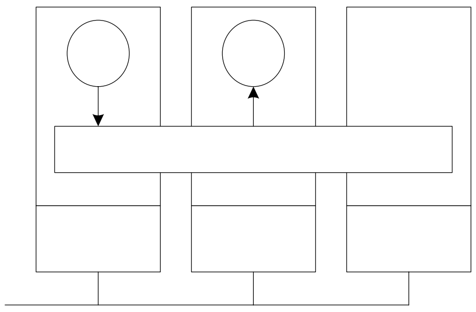


Figure 5-2 Before migration

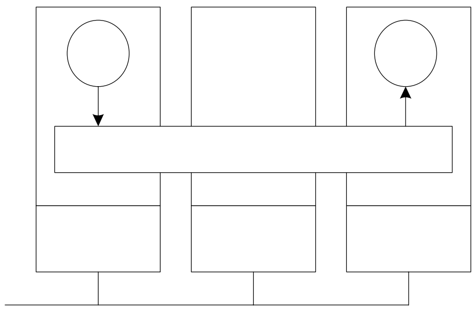


Figure 5-3 After migration

In Figure 5 2 the client, c1 contacts service s1 which currently is located at computer 2. Next in Figure 5 3 the service s1 has moved to computer 3, and the client follows same procedure to contact the service. This would only be transparent to the client if the middleware supports migration.

[4] Questions 2 to 4 only make sense if location transparency is achieved.

This test will verify if and how the technology support migration of the service without affecting the clients way of addressing the service.

1. Does the technology support migration function to easily migrate code between hosts?
2. If supported, how is this solved?

5.3.3 Service replication transparency

Replication transparency is a distribution transparency which masks the use of a group of mutually behaviorally compatible objects to support an interface. Replication is often used to enhance performance and availability. [8]

The word replication transparency is often used in database systems when referring to functionality built into a server which maintains and uses several instances of the data automatically. Many heavily loaded websites or ftp-sites use a poor way of location transparency by providing the same document or file by different name. This doesn't scale at all.

Replication could be achieved both on client side and server side. The difference between them is that on the server side replication the client will act as if it was only one service. Otherwise on the client side replication, a proxy at the client side will handle the connection to each service replica.

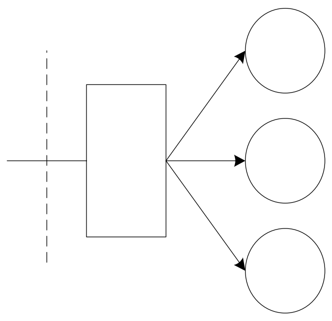


Figure 5-4 Server side replication

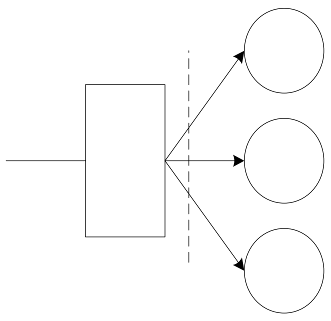


Figure 5-5 Client side replication

By this criterion we will look into the possibility of providing services by letting the infrastructure or service handle the replication transparency by deploying a service at several servers. In this case the client would to know the fact that the service might exist at several locations.

Each of the following questions will be answered in the test section:

1. Does the technology provide functions to create service clusters, and then let these act atomically?
2. If provided, how is this supported?
3. If not supported, are there extensions to technology in order to provide the replication function?

5.3.4 Service failure transparency

"Failure Transparency: A distribution transparency which masks, from an object, the failure and possible recovery of other objects (or itself), to enable fault tolerance" [8]

In large and complex distributed systems, some kind of failure transparency built into the RPC layer would be extremely useful. This is, however, not directly supported by most distributed object models.

Failure transparency mechanisms should be able to recover from network failures or application failures without help from the programmer.

The ODP-RM ISO standard [8] proposes two approaches to gain failure transparency.

1. Replication might be used to raise the level of availability.
2. A checkpoint and recovery function will make it possible to continue execution after a failure.

In this test we'll figure out each technology's ability to survive service failures, which party has to take action, and if possible, how the system has to be configured in order to make it work.

1. Does the technology provide any of the functions mentioned above?
2. If provided, how does it work?
3. If not provided, is there known extensions to the technology which can achieve failure transparency?

5.4 Market review criteria

5.4.1 Technology's market position

When selecting a middleware for implementation of a distributed system, there are other non-technical aspects that have to come into consideration.

In some settings, a system built on "new hot" technology tends to sell better than a more "duller" one. In others, well known and sturdy technologies sell better.

This criterion will be discussed with respect to each technology. The arena of this criterion is not to give an absolute answer to the technology's position in the market, but only an indication of the general acceptance to this technology.

6 Building test applications

6.1 Overview

This chapter will give a description of the developing process of the prototypes. The first part of this chapter introduces the test application, and why this application is suitable for the tests. 6.3 – 6.5 will then point out differences in the test applications for each technology.

6.2 Introduction

The test criteria which are the foundation of this thesis are based on testing the platform of distributed systems, and the focus is set on the transparent abilities. This indicates the ability to discover and deploy services, filter mechanisms and persistent data support. The model should promote these abilities and support the platform to achieve this goal.

We had to take into account the fact that the technologies which are target in this diploma are very different in abilities and areas which they are used. - If we chose a prototype that is too advanced, it could end up that it is impossible or very difficult to implement on some platforms. To prevent this we defined a minimum set of rules which the model should support; we chose that the communication should be done through a simple string based transmission. The ability of string transmission is in most cases supported, and the mandatory “Hello World” example often uses exactly this form of communication.

The next requirement was the ability to give live data and to identity of which server supported the data stream. Since we are working with transparent solution the source of the data stream could change from time to time. We wanted our model to give us indication if this happened and at the same time give us the possibility to spot when. - This could be done with timestamp or some sort of a counter. The identity of the server could be set by the domain name system (DNS) name.

We struggled a while to find a model which could accomplish the tasks and still be simple to construct and implement. At last we settled on a distributed clock system, which we found capable to support the entire requirements we had set.
This type of prototype would have a constantly call for attention from the service, - we found it interesting to see how this affected the platform. The time stamp would give us an indication of latency in the system, and the possibility of lost request. And the fact that it is a really simple model will make the difference in implementation less important.

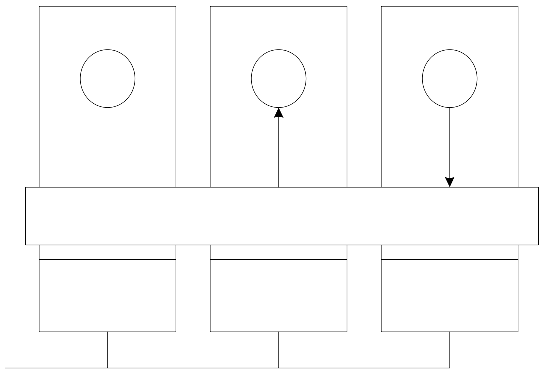


Figure 6-1 Conseptual Service model

6.3 Jini test application

Jini is an easy platform to develop distributed applications, and the construction of the prototype went in most cases without any trouble.
The most difficult part was actually the startup environment that consists of web servers, lookup servers and rmid, everyone needed script file for working properly in windows and Linux.

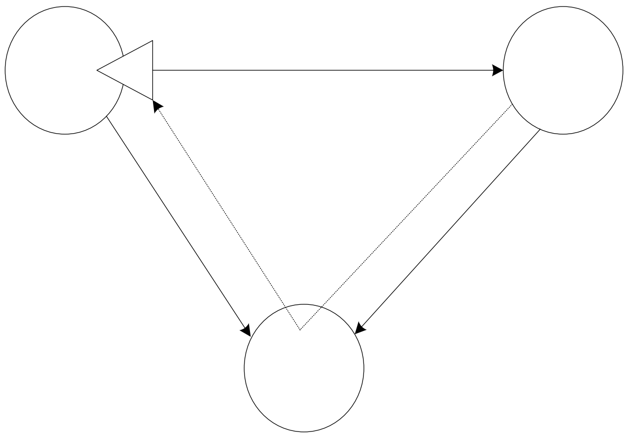


Figure 6-2 Conceptual model of Jini test application

The developing process was divided in two different areas; the developing of clock service and client, supported by regular Jini architecture and the developing of clock service and client, supported by JavaSpace architecture.

We chose to make two different models because of the topic, load balancing. We were struggling by the idée how to solve this in a Jini network. After looking around and reading the Jini newsgroup, - which by the way is excellent, we came to the conclusion that we had to use JavaSpace.

The rest of the developing process can be found in the Appendix B.

6.4 JXTA test application

Realizing the prototype we struggled a while to make the first progress. There is not much available material about JXTA, mostly all programming examples is gathered at JXTA home page. Luckily we found what we needed and we could start on our first prototype of the clock service.
We tried to make it simple and concentrated on the ability to deploy and search for services.

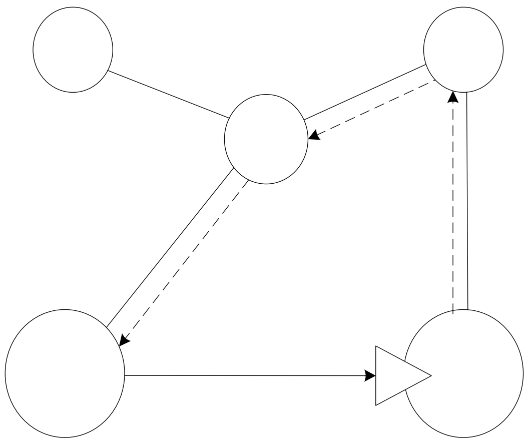


Figure 6-3 Conseptual model of JXTA test application

The rest of the developing process can be found in the Appendix D.

6.5 Web Services test application

The "definition" of web services above tell us that web services are nothing new. It uses well known protocols and technologies both for transport and application. Because of this; web services could be implemented in nearly any language. In this thesis we'll look at web services and clients implemented in java.

Several development environments exist, i.e. IBM's Web Services Toolkit v1.0 (http://www.alphaworks.ibm.com/tech/webservicestoolkit), The Mind Electric Glue v2.0 (http://www.themindelectric.com/glue), Sun Microsystems Java Web Services Developer Pack v.Ea2 (http://java.sun.com/webservices/downloads/webservicespack.html) and more.

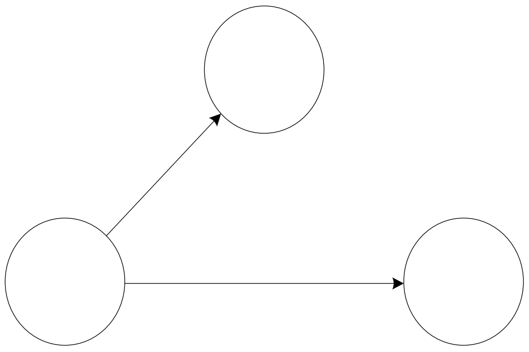


Figure 6-4 Conseptual model of Web Services test application

The Web Services test application consists of two parties, the client c and the service s.

The rest of the developing process can be found in the Appendix C.

6.6 Summary

In this chapter we have learned how we implemented a prototype that is used in the testing. It is also given a short description of the developing process for each prototype.

7 Prototype testing and result

7.1 Overview

The two preceding chapters establish an understanding of the test criteria and how the prototypes where implemented. This chapter contains all test-runs against each technology and presentation of the results with following discussion.

The structure of the chapter is important. Each technology is evaluated separately against each criterion. Each test chapter is arranged into sections of test-run, results and discussion. Finally a summary presents all the results in a table.

7.2 Jini

7.2.1 Service location transparency

7.2.1.1 Test execution

The lookup service was started in three phases; first the web server, which makes sure every clients and service can download the reggie proxy. The next step is to start the RMI activation daemon rmid, And finally the lookup service reggie. These processes are all running at computer 1.

The next step in the test is to start a service. This service has a web server as the lookup service, which support proxy download. Both of them were located at computer 3.

Finally the client was stated at computer 2.

7.2.1.2 Test result

When the service was started it listed out all the LUS it registered to. The search process discovered the LUS without any trouble. What came as a surprise was the response from LUS around the world; - Trondheim, Oslo, Germany and even Taiwan were not rare.

When the service registration process was finished the client was started. Not surprisingly it found our own lookup service first, and downloaded the service proxy. However the client received response from around the world which the service had registered to.

When the proxy was downloaded the communication could start.

We experienced no problem in this test.

7.2.1.3 Test discussion

> *"Does the technology provide naming mechanisms to achieve service discovery functionality?"*

By doing this test it can be concluded that Jini has the ability to support service discovering in a transparent environment. This ability is mainly supported through a regular service, the "Lookup service".
Jini does not support a traditional naming service like those found in object based distributed systems or distributed file systems [18]. Such name services can however be easily implemented as part of the service layer within the Jini architecture. This is exactly what is done with implementing the Lookup service on top of the Jini service layer.

A standard approach in many distributed systems is to configure the service and client with a well known address to the lookup registry. The Jini community has taken a different approach.

The search for LUS is done by multicast, - when a client or service wants to contact a LUS, it is through this protocol the UDP packets are sent. When a LUS has received a multicast message, it replies by connecting directly to the requesting service/client, and sends a unicast (point-to-point) message. This message contains the proxy object that the service/client can use to contact the lookup service. Since the LUS is a service just like any other Jini services all access is done through proxy objects. In addition to discovery, LUS will regularly announce their presence by using multicast.

As mentioned in the test result the discovery process worked fine on our own local area network. What came as a surprise was the discovery process also seemed to work in a larger area. Since we have no data of how many LUS is located in our network neighborhood, we can not conclude if the multicast discovery process will work on a larger area, but lookup process indicated that the multicast worked outside our local area network.

“What type of lookup is supported?”

When a service had registered to a lookup service, clients can start to search for it. When clients search for services, they create templates which describe what they are looking for. This template will be matched against the service items stored in a lookup service.
Templates provide a way to search for a service given its unique ID, the interface the service-proxy implements or the attributes attached to it.
In most cases the interface is the only search criteria, but it can be supplemented with attributes which containing a description of the service; this is done by implementing Entry objects.
A well used example of this is the search for a printer. When printing a document you are probably not interested to discover that you have to walk a couple of mile to retrieve it. You are very likely eager to know where the printers are and which one is closest to you. The Entry object is used to make this process easier; you can supply string information about your service that could describe which floor and building the printer is located.

We didn’t supply our service with entry objects to describe their abilities but the search process still managed to find the service it was looking for. Our search criterion was the interface. We can conclude that this was enough in our case and that the search process worked perfect.

“Does the naming service deal with the problem of unreferenced objects?”

The Jini network handle unreferenced objects through the leasing time. If a service doesn’t renew it lease it will be deleted from the LUS when it is expired.

“How does the naming system scale in large distributed systems?”

Jini address scalability through federation. A federation is Jini communities that are linked together. Ideal the size of a Jini community is about a size of a workgroup (10 – 100) objects.

LUS can be divided into different communities; this is done by simply starting it with a group name. In this example we have used the group name public which means that every one is free to connect to it, - this is the default configuration. We could however easily have used a different name such as HIA.

```
GROUP=public
java -Dcom.sun.jini.reggie.proxy.debug=true -jar $JARFILE $CODEBASE
$POLICY $LOG_DIR $GROUP
```

Listing 7-1 Staring reggie

Look up services could be joined into federations by just adding group names to the startup string in each LUS. Each LUS can support several groups, just by adding multiple names to the startup script and separating them with a comma.

```
GROUP=public,hia
java -Dcom.sun.jini.reggie.proxy.debug=true -jar $JARFILE $CODEBASE
$POLICY $LOG_DIR $GROUP
```

Listing 7-2 Starting reggie with group

The federation will probably work fine as long as the network topology doesn't depends on letting the multicast messages travel to far. The multicast messages are probably Jinis greatest advantage and disadvantage.

The fact that neither the client nor the service needs any information about the location of the LUS makes the discovery process extremely flexible and self maintaining.

The problems start when the Jini network grows large. Some routers will dump multicast messages that try to leave the network, other will just make sure that the TTL counter is not set to high and by that make sure to decrease the radius which the packets can travel.

7.2.1.4 Test conclusion

The Jini network support service location transparency. The client creates templates which describe what they are looking for. This template will be matched against the service items stored in a lookup service.
Templates provide a way to search for a service given its unique ID, the interface the service-proxy implements or the attributes attached to it.

7.2.2 Service replication transparency

7.2.2.1 Test execution

Test more than one service
First we started the lookup service as described earlier.
Then we started two services, one at computer 3 and one at computer 4.
Both services were supported with a web server.
Finally we started the client, and let it search for services.

Test more than one service on each server
We did the same test as shown above, but this time we started more than one service on computer 3 and 4.

Test a load balancing model in JavaSpace
Two new services had to be deployed on computer 1 in order to test JavaSpace. First the mahalo service which is the transaction manager. Second is the outrigger service which implements JavaSpace.

The services were started at Computer 3 and 4, and the client at computer 2.

7.2.2.2 Test result

The services on machine 3 and 4 seemed to work fine since both were registered at all the lookup services.

The test with more than one service on each server gave no side effect and they all got registered.
When we started our client, it searched among the lookup services for the service. The search process was successful and the client received a proxy which corresponded to the request. The fact that there was more than one service that fitted into the search criterion did not seem to confuse the lookup mechanism.

The JavaSpace model handled load balancing very well. When the client had written the tuple to space, the worker/servers took turns at processing them. The clock service was as before updated each second. And this seemed to work fine
(We used a non-persistent version of JavaSpace)

7.2.2.3 Test discussion

> *"Does the technology provide functions to create service clusters, and then let the cluster act atomically?"*

The reason for replication can be divided in two primary reasons; reliability and performance. First, data are replicated to increase the reliability of a system. If a file system has been replicated it may be possible to continue working after one replica crashes by simply switching to one of the other replicas.

The other reason for replicating data is performance. Replication for performance is important when the distributed system needs to scale in number and geographical area. Scaling in numbers occurs, for example when an increasing number of processes needs to access data that are managed by a single server. In that case performance can be improved by replicating the server and subsequently dividing the work [18].

To test the support of replication we made two different prototypes, one that uses regular Jini network and another one that used the tuple space JavaSpace.

After doing our test we can conclude that the Jini model can offer replication transparency. Services and lookup services can be replicated, without affecting the client. The replication is possible since there is a loose connection between the client and the service and lookup service. The multicast network makes this possible by dynamically re-connect to new entities if one fails.

Since the selection of services in a regular Jini network is often random, the result of this can be that the same service is chosen over and over again. The JavaSpace model was tested to see if it could offer a better solution. This model has a shared memory space where clients can write objects into, and the worker/servers can extract the object to work on, the result is returned to space.

This solution seemed to work fine; it offered a better solution of load balancing since the worker/server will not undertake another project if it is busy working.

To increase the work capacity it is simple to introduce new server/worker to the space.

7.2.2.4 Test conclusion

Jini supports replication both to the service and lookup service. JavaSpace gives the opportunity to make a better load balanced system.

7.2.3 Service migration transparency

7.2.3.1 Test execution

In order to test if Jini supports migration transparency the service was deployed on computer 1 and then moved to computer 2 while tracking all modifications which had to be done.

The next test verifies how the system handles service migration. We started a service on computer 1 and let it register at the lookup service. Then we killed the process and moved the service to computer 3 and started it.

The final test run will verify if the technology supports relocation after migration.

7.2.3.2 Test result

In The first test we simply moved a service from a Linux box to a windows platform. Some changes had to be done in the shell scripts; the path separator is defined different in windows and Linux (";" to ":").
Since we prefer to have the web server running at the same machine as the service, we had to change the IP address to the new web server as well. The registration was successful when these changes were done.

The next test was the moving service, and the effect on the Jini network. The registration at the LUS was successful after moving the service. The old reference to the service was automatically deleted from the lookup service when the lease time was expired.

The final test was the effect on the client if the service is relocated.
Both the services where registered at the lookup service when the client started its lookup discovery process. It registered successfully to one of the services and started communication.

To test relocating we killed the chosen service. This led the client to choose the next available service.

The process succeeded: it registered to next available service.

7.2.3.3 Test discussion

> *"Does the technology support migration functions to easily migrate code between hosts?"*

Jini is pure Java technology. Since Java software runs in a virtual machine, it's more or less platform independent. It's only required that the platform has an implementation of the virtual machine. Because of this the only modification needed is small changes in the startup scripts.

"If supported, how is this solved?"

Jini supports migration and relocation transparency to some extent.

The lookup service has a leasing mechanism. Every service has to implement a leasing time when it is register to the lookup service. The leasing time is chosen by the developer, we had set it to 1 minute. This makes it sure that the lookup service doesn't get flooded with non referenced services. If a service doesn't renew it lease before it expires the object is deleted.

A moved object will consequently be deleted after a while and this will make sure the clients that connect in the future will not be affected by the movement.

To make this work in runtime there must be implemented support for this in the client. In our prototype we stored all incoming proxy discoveries in an array. When a service didn't response to a request it was deleted from this array and the next one in the list was selected. This kept on until the array was empty, then the discovery process started over again to fill the array. This model is probably not optimized; one improvement would be to make sure to always have a full array list. This could be done by letting a thread do a discovery process regularly. However our model demonstrated the principle that it is possible to achieve relocating in a Jini network.

7.2.3.4 Test conclusion

Jini support migration of services. Old references are deleted over time with help of a leasing mechanism.
To some extend there is also possible to move services in runtime, this future must however be implemented by the developer.

7.2.4 Service failure transparency

7.2.4.1 Test execution

Testing the lookup service.
We started the lookup service at computer 1, and let a couple of services register at it - computer 2 and 3. Then we killed the lookup service, and observed how it was affected after restart.

Testing the JavaSpace
We started as always the lookup service at computer 1. At the same computer we started also the mahalo and outrigger (not the transient version). We let some tuples be written into space and killed the JavaSpace. After reboot we observed if the reboot had affected the network.

7.2.4.2 Test result

The lookup server did not loose data during the reboot, neither did the JavaSpace implementation.

This service would however not be accessible by the client through the reboot.

7.2.4.3 Test discussion

We have decided to test the support of persistent data and the support of transaction. By support of persistent data we mean that objects are written to disk, this again will give the possibility to continue on the failed work after a crash or failure. A transaction means a sequence of information exchange and related work (such as database updating) that is treated as a unit for the purposes of satisfying a request and for ensuring database integrity. For a transaction to be completed and database changes to make permanent, a transaction has to be completed in its entirety[5].

Jini has no support of persistent data except at the LUS (reggie) which store all the registrations to disk. If the LUS crashes it would not loose any registered services.
The service and client will not have any support unless a specific solution is implemented by the developer. We have not implemented this on our clock service since it doesn't make any sense to store time for later.

The JavaSpace structure has on the other hand support of persistent date. Objects written into space can be stored at the hard-drive in order to survive a system failure. This is done by starting outrigger.jar instead of transient_outrigger.jar (see script file for JavaSpace for more information). Note that transparent system has poorer performance, and is not recommended during a development process.

Jini has also transaction support to assist in carrying out series of operations on multiple objects, the transaction manager in Jini uses two-phase commit. This support is essentially given only in the form of a set of interfaces; their actually implementation is left to others. However, Jini can be configured with a default transaction manager [18].
JavaSpace has also support of transaction, writing and reading from space is operations which are controlled by transaction.

7.2.4.4 Test conclusion

Jini core have implemented transaction mechanisms. The lookup service has support of persistent. JavaSpace can be implemented with a persistent space. Client and service has however no support of failure transparency.

7.2.5 Technology's market position

7.2.5.1 Discussion

> *"That was our original thought. Change is not a rare event - it's constant. We had to figure out a way to allow change to happen without involving people. If change required people, and considering networks now growing into millions of machines and the amount of change those networks experience, we would all have to become system administrators. The only way to avoid that is to automate the ability to deal with change."* [6]

Jini has been around since 1999 and is by definition (in the computer world) already a matured technology. That Jini has survived for so long has not been a matter of course after a tough start, this drift seems to turn.

At the moment the Jini technology is licensed by nearly 40 companies that span a huge gamut from device vendors to enterprise software companies. These include disk drive manufactures such as Quantum and Seagate, cellular phone manufactures Nokia and Ericsson, printer vendors including Xeroc, Canon, Epson and Hewlett-Packard, camera manufacturer Kodak, network vendors Cisco and 3Com, software producers BEA Systems, Novell and Inprise, and a huge number of consumer electronics companies

[5] Definition collected at http://whatis.com
[6] Jim Waldo, Sun Microsystems, responding to an interview question in Java World, about the cost of change, Nov. 21, 2001

including Sony, Sharp, Philips and Toshiba. Jini has also been licensed for use by companies as diverse as AOL and Kinko's [9].

Examples of products that have been brought out of this license agreement are Cisco systems "The Spanish Inquisition" (SI). It is a scalable communication framework built upon Jini and JavaSpace functioning through the use of agents (see [15] for more information).

But Jini technologies are still struggling to conquer the world. – The greatest problem is the large memory footprint, which had restricted it to environment of sufficient computing capacity.
Mark Driver at the Gartner Group reiterated this dilemma.

> *"Although Jini ™ still promises to lay a sumptuous table in the next few years in enabling next-generation networks, its current networking capabilities have proven to be a little too rich for application developers who, of necessity, are on strict memory and processing power diets."[13]*

The Gartner group goes on to predict that if the footprint issues of Jini can be overcome successfully then

> *"Jini should begin to emerge as a key enabling technology used in 70 percent of commercial network computing applications."[13]*

In answer to this JMatos was developed. PsiNaptics's JMatos software is a fully compliant micro version of Jini network technology ideally suited for mobile constrained devices. It has a footprint of less than 100 kilobytes for full Jini technology compatibility. This means that a small device does not need to rely on the presence of Jini servers or surrogates in the network, allowing clients to independently discover and utilize the services offered by the device.[7]

On JavaOne 2002, PriNaptic demonstrated an integration of Jini technology with the new mobile OS, Symbian. Considering Symbians licensees account for approximately 70 per cent of worldwide mobile sales, the importance of this technology breakthrough is enormous for the future of Jini. Together with the knowledge that java will be supported by new mobile nodes produced by Ericsson, Motorola and Nokia the foundation should be obvious.
Jonathan Allin, Symbian's Strategic Product Manager for Java technology says;

> *"There is an emerging need in the wireless environment for mobile phones to be able to interact with each other, and with other devices such as back-end servers, printers, and domestic appliances. In many cases the applications and services provided will operate within a local Bluetooth network, though 2.5G/3G wide area networks also present exciting possibilities,"[14]*

Jini will solve a lot of the problems challenged by mobile units. As the Jim Waldo says in the beginning, the need for technology that make communication easy and "*automate the ability to deal with change*" is the key to the future. The need to manually upgrade the device with new drivers whenever it experiences a new environment is not an issue in a Jini network. Jini devices will automatically deal with discovery and downloading of protocols, offered by the network.

[7] To read more about JMatos use reference [13]

DaimlerChrysler has done a research on how they can use the new Jini technology in their cars [16]. The idea is that users of the car should be able to connect to their home. Application should control through the Jini network lights, climate, sprinkle systems, cameras, alarm systems and the home entertainment system.

A mock home that is set up to demonstrate the DaimlerChrystler Infotronics system enables the drivers to turn on light in the garages and in the house as well as get real time pictures from a security system and of the family pet.

Today, networked home applications are converging upon a Java standard being created by the Open System Gateway Initiative (OSGI) standards group. By incorporating the same technology in the car, the vehicle can easily join the network home environment.

The possibilities of the future are endless, but prediction is difficult if not impossible. Jini has answers to many of the problems which mobile computing will be struggling with; it is already accepted as a serious alternative as network bridge between the different entities by firms that will lead us into the next generation of technology. Still this makes no guarantee that it will be the chosen product, examples as WAP has shown us that the market has the final word.

An important aspect that can be the key to solve this problem is the source licensing Sun provides to this technology. It is in many cases similar to the “Open Source” movement that has given us the Linux distribution and apache web server we know today. The great advantage is the rapid development of new software and this again can give the technology a kick start on a new direction when this is needed.

The drawback of this is that many companies who might otherwise be eager to develop for the technology may be reluctant to be required to turn over their intellectual properties to a third part.
The community license model that Sun is using for this product allows very open access to the source code, but it doesn’t require companies or developers to hand in their developments back to Sun.

The only requirement that Sun expect before you publish your product is that your implementation pass the Jini Platform Compatibility Kit. This requirement is to make sure that there is no Jini service which doesn’t comply with the original Jini specification.

An example of such product is the GigaSpaces implementation which is an Enterprise-scale JavaSpace implementation [17]. Just a better version of the original JavaSpace implementation from Sun, it would probably not exist if the license requirements had been different.

7.2.5.2 Conclusion

Jini has been embraced by the mobile companies, Cisco has used it in their switches, DaimlerChrysler is predicting future where their cars are using the Jini network. Problems like a too big memory footprint have been solved with the JMatos and, together with J2ME, the millions of small devices have now the option to be a part of the Jini community.
The open software license Sun is using on this product should also give the foundation of a rich product line maybe as much as 70% as the Gartner group is predicting.

7.3 *Web Services*

7.3.1 Service location transparency

7.3.1.1 Test execution

The service is registered in the IBM UDDI beta 2 register (see appendix C). This was done manually, but could easily have been done by using the UDDI4J API.

The service discovering test here uses UDDI to lookup services. Since we already know who's providing the service, we search for the actual service at our record in the UDDI registry.

In UDDI the service is registered with an access-point and a technical model including a WSDL document (see appendix C for details about information registered in the UDDI registry).

For the lookup process to be successful when you have no prior knowledge to the service, it's important that a service has a fine-grained description. In our test application we know the service provider and what service we want, the only thing we need to know is where the service is located. This information is both located in the UDDI registry and in the WSDL document referred to in the technical model.

The client uses UDDI4J to locate the business by name, and then locate the Clock service under the service. Then the client accesses the technical model related to this service and uses WSIF to invoke a method at the web service based on the WSDL document.

The service is located and the access point (URI) is extracted from the registry and used for accessing the service.

7.3.1.2 Test results

The service was located and invoked by the client.

Web Services *provide location transparency* when the client uses UDDI to look up the service's access point.

7.3.1.3 Test discussion

"Does the technology provide naming mechanisms to achieve service discovery functionality?"

There has been "hype" around web services and dynamic service invocations based on UDDI and WSDL. When interacting with web services there are at least two parties. They have to seek common understanding on both how the service should be invoked and what each part of the service actually relates to. Steve Vinoski of Iona states in a paper at WebServices.org that invoking web services needs a common knowledge at the business-to-business (b2b) layer [7]. UDDI and WSDL do not have any support for such a feature[8], and it is therefore nearly impossible to have true dynamic lookup and invocation in web services at least in the near future.

"What type of lookup is supported?"

[8] except from the service locator taxonomies

The UDDI registry is arranged around business registration. UDDI support several ways of discovering a particular service. If you don't know who delivers a particular service of your need, you would first search for businesses which deliver services in the desired business sector. If you already know the service you would go directly to that business's record in the UDDI registry and search for the service. If you know how the service signature as described in a technical model, you would search for services which conforms to that technical model.

UDDI lookup is neither the only nor always the best tool for service discovering. Microsoft and IBM has worked together to describe a proposal for a Web Service Inspection language which can be used as an index to web services at a known network location [2]. Pretending you are only interested in interaction with services provided by IBM, you would download the inspection document from a known location at IBM (e.g. http://www.ibm.com/ispection.wsil) and use it to find services.

There are several companies providing UDDI registries (e.g. IBM http://www.ibm.com/services/uddi, Microsoft http.://uddi.microsoft.com).

A drawback to UDDI is that you would have to know the exact URI in order to access the registry, since it does not provide any mechanisms for dynamic discovery of the location registry.

> *"Does the naming service deal with the problem of unreferenced objects?"*

The UDDI registry does not have any mechanisms for automatic removal of unreferenced services. It's entirely up to the service provider to remove services which don't exist from the registry.

> *"How does the naming system scale in large distributed systems?"*

UDDI is designed to support large scale systems. Several providers (see list at http://www.uddi.org/) replicates registry data between each other, and then the consumer could query any UDDI registry for the same information. Pretend a centralized registry like UDDI would store all information in a single system; it would be a bottleneck and a critical point of failure [25].

7.3.1.4 Test conclusion

Web Services supports location transparency through the use of UDDI to lookup services before invocation.

7.3.2 Service replication transparency

7.3.2.1 Test execution

It was not possible to implement any test application for this criterion since replication is outside the scope of web services (see 7.3.2.3).

7.3.2.2 Test result

Not relevant

7.3.2.3 Test discussion

> *"Does the technology provide functions to create service clusters, and then let the cluster act atomically?"*

Replication transparency is traditionally a server feature. Web services are similar to document-based systems like HTTP and FTP, and use the same infrastructure [18]. A SOAP invocation is received by an RPC router which decodes the SOAP message, loads the class and invokes the method before the result is returned as a SOAP message. In most cases the RPC router is a servlet hosted at a Java enabled web server.

Then most likely web services could gain from replication technologies developed for traditional web sites.

> *"If provided, how is this supported?"*

Two types of replication solutions for document based distributed systems are discussed in [18]. First service clusters is a client transparent solution which uses a front-end responsible of distributing requests to a server in a group of servers providing the same service. But this solution leads to a potential performance bottleneck since all communication goes through a single point.

In the ScalaServer project, S.Pai have developed a TCP handoff mechanism which *"allows a cluster front-end to hand off an established TCP connection to another node in an efficient and client transparent manner"* [24].This mechanism will reduce the load at the front-end of a traditional cluster solution.

Apache SOAP has a pluggable architecture which allows web services to be implemented in several different technologies such as Enterprise Java Beans (EJB) [j2ee website] and standard java classes. Additional the SOAP RPC router could be deployed on nearly any Java enabled web server.

At the UBEANS [27] web site there is a description of how an Apache 1.3.23 server could be used as a proxy front-end for two Tomcat 4.0.2b2 servers by using the mod_jk module for Apache. The work can be balanced between servers by a factor.

The Bea WebLogic Application server provides a cluster solution where several EJB services could be deployed on a server group [26]. The WebLogic server uses three mechanisms for service replication. First of all it uses DNS load balancing. When a DNS name is mapped to several IP addresses in the server cluster it will cycle through the IP addresses each time a client performs a DNS lookup. Secondly it uses a content aware proxy system where certain requests are forwarded to the correct server in the cluster. E.g. requests for static html pages could be forwarded to one server and request for servlets could be forwarded to another server. In addition to the content aware proxy, the application server also supports a traditional cluster solution with a front end which balances the load between each server in the cluster.

By deploying web services at a cluster enabled application server such as Bea WebLogic and Apache Tomcat, it would support replications transparency.

> *"If not supported, are there extensions to technology in order to provide the replication function?"*

See previous discussion. Replication would be implemented in underlying layers like the web server.

7.3.2.4 Test conclusion

Web Services does not support replication in its service model. But we have discussed how Web Services could benefit from standard clustering mechanisms.

7.3.3 Service migration transparency

7.3.3.1 Test execution

This test was based on the same prototype as the location transparency test.

First the code was deployed at computer 4, and executed as in 7.3.1.1. Then the web service implementation was moved to computer 3, and executed there.

The changes in code and configuration were tracked.

7.3.3.2 Test result

Migration of Apache SOAP web services requires only reconfiguration. No code has to be altered.

First of all, the actual web service implementation has to be moved to the new location (ant build-script). Then the Apache SOAP deployment tool has to be executed. Now the web service is operational in its new location, but in order to let the client find the service, the UDDI registry entry has to be altered with the new location. The last step is required if the client uses WSDL for invocation, then the `<service>` section of the WSDL document has to be altered to reflect the new service location.

7.3.3.3 Test discussion

> *"Does the technology support migration functions to easily migrate code between hosts?"*

Migration and relocation transparency is mainly part of the mobile object discussion. Web Services is coupled to web technology and is by definition a document based distributed system. These systems are hardly ever mobile. Web applications are deployed on one server and they remain there. When a web site is moved, the most common solution is to place a redirector in the old location which forwards the consumer to the new location.

We could foresee an equal solution when moving web services, but the location transparency function tested in section 7.3.1 is an even better solution than the redirector.

Verified by the test, we could state that migration transparency is supported.

> *"If supported, how is this solved?"*

Migration transparency is mainly gained by reconfiguration. Simple ant build scripts, similar to the one generated for these tests, could be used to automate the process of migrating the service.

The Apache Axis has, in contrast to Apache SOAP, better support for automatic deployment and WSDL generation. The process could be even more unattended than the method outlined above.

7.3.3.4 Test conclusion

Web Services supports migration transparency when UDDI is used for lookup of services. The migration of services requires some reconfiguration at the server.

7.3.4 Service failure transparency

7.3.4.1 Test execution

Web Services is a document-based distributed system (see section 7.3.2.3) and gain from the mechanisms developed for web servers. Web services do not provide any mechanism for failure transparency.

7.3.4.2 Test result

Not applicable

7.3.4.3 Test discussion

> *"Does the technology provide a replication or a checkpoint and recovery function?"*

Fault tolerance in document-based distributed systems is achieved by client-side caching and server replication [18]. Since web services provide dynamic content, it is not relevant to use client caching. But by using service replication as described in section 7.3.2.3, availability could be raised. But failure recovery and transaction control is not supported.

> *"If provide, how does it work?"*

Not totally failure transparent, but replication is supported which would increase web service accessibility. See section 7.3.2.3 where this is discussed.

> *"If not provided, is there known extensions to the technology which can achieve failure transparency?"*

We have not been able to locate any technologies which enable full failure transparency in web services.

7.3.4.4 Test conclusion

Failure transparency is not supported in Web Services. But replication could be used to increase availability.

7.3.5 Technology’s market position

7.3.5.1 Discussion

Beyond the hype, Web Services seam to have got a kick-start as nearly every company in the e-commerce applications server business sector nowadays is advertising support for web services.

A poll at WebServices.org *“Do you think Web Services are just hype, or have huge potential?”*

Huge potential	48%
Have promise	30%
Too early to say	14%
Just hype	8%

Table 7-1 Web Services "hype-poll" [31]

Most standards used by Web Services are evolved mainly by companies and then submitted to W3C. IBM and Microsoft founded the Web Services Interoperability Organization (WS-I) which works to achieve interoperability between different Web Services implementations. WS-I is supported by 100+ members, which is divided into working groups.

Web Services have the propensity to provide a new platform both for b2b integration, and integration between new and legacy applications in the vertical market.

Another even more interesting poll at WebServices.org: *“What language do you use to develop your WebServices?”*

C#	8%
C++	2%
Java	75%
Perl	4%
Visual Basic	5%
Other	5%

Table 7-2 Web Services implementation language poll [33]

This emphasizes java’s position as the implementation language for Web Services. Even the lack of optimism from Sun Microsystems, which actually was quite skeptic to Web Services in the start, seems not to have limited this. This is probably caused by IBM’s position in the Web Services evolution. Now IBM wants Sun to join WS-I as a founder member because java seems to be one of the largest developer languages for Web Services. But Microsoft seems to dislike this since Sun was rather late in the game.

The Gartner Group thinks that web services might be the opportunity for creating new growth for service companies in the short to midterm.

> *”Web services have the potential to be the next key demand generator in the IT marketplace. No services firm can afford to ignore them”* [33]

IBM delivers Web Services solutions for Storebrand ASA [34]. Storebrand ASA, Norway’s largest insurance company has been using Web Services to replace the manual process of calculating personal benefits for 390,000 employees at 6,500 different customers. Now is the information extracted directly from the customer’s payroll system, and processed in Storebrand’s mainframe [35].

AvantGo announce support of Web Services in their M-Business solution [36] and almost any company with java enabled application servers announces support for Web Services in their server solutions e.g. IBM WebSphere [37], Microsoft .net [38], Bea WebLogic [39] and Sybase [40].

7.3.5.2 Conclusion

Web Services seems to have got a strong position already, there are however some doubters. Web Services is mainly adopted in the vertical market, where it's mainly used for business-to-business integration, and integration between new and legacy applications.

7.4 JXTA

7.4.1 Service location transparency

7.4.1.1 Test execution

The simplicity of starting the JXTA test is interesting. The service was started first at computer 1 and the client was first started at computer 2. There is no lookup service or other form of registration storage which had to be started.

7.4.1.2 Test result

When we started our service we first had to implement our username and password. This is mandatory with every startup of JXTA peers but it can be implemented in the script instead. When the service was started it waited for incoming clients to register.

Starting the client we had to go through the same process writing username and password. After this process was finished the client discovered the service and the communication started.

We experienced no problems in executing this test.

7.4.1.3 Test discussion

The JXTA platform support service discovering in a transparent environment, the process is controlled by advertisement in form of XML. This advertisement is sent to all known peers the searching peer knows about; - they will again distribute the request to all peers they know about. The Peer Discovery Protocol consists of only two messages; a request format to use to discover advertisements, and a response format for responding to a discovery request.

These advertisements are the basic unit of data exchanged between peers to provide information on available services, peers, peer groups, pipes and endpoints.

The elements of the Discovery Query Message describe the discovery parameters for the query. Only those advertisements that match all of the requirements described by the query's discovery parameters are returned by a peer.

The fact that there is no central form of registration of services worked amazingly well. It must be said that in our test the two computers were on the same network and only separated by a switch. We have however examples of similar network in Gnutella, even if some discovery processes can take awhile and request are sent more than one time, the ability to cover the entire world is there.

7.4.1.4 Test conclusion

JXTA supports service location transparency. Searches are done through advertisement in XML and peers that find a mach will send a response.

7.4.2 Service replication transparency

7.4.2.1 Test execution

We started two services, one at computer 1 and the second at computer 3. The client was started at computer 2.

7.4.2.2 Test result
After we had filled in login information and started the two services, we could start our client. The discovery process worked fine and the client chose one of the services.

7.4.2.3 Test discussion
We can conclude that replication is supported in a JXTA network. Actually there is a great advantage that there is more than one service present. This increases the chance of discovering, and at the same time it makes sure that not only one server has to take all the traffic. The technology is a natural extension of the Internet's philosophy of robustness through decentralization.

In many communities that share resources there is a risk of suffering from The Tragedy of the Commons: the overuse of a shared resource to the point of its destruction. The Tragedy of the Commons originally referred to the problem of over-grazing on public lands, but the term can apply to any public resource that can be used without restriction. In some P2P systems, peers can use the resources (bandwidth, storage space) of others on the network without making resources of their own available to the network, thereby reducing the value of the network. As more users choose not to share their resources, those peers that do share resources come under increased load, and in many ways the network begins to revert to the classic client/server architecture. Taken to its logical conclusion, the network eventually collapses, benefiting no one.

7.4.2.4 Test conclusion
JXTA network is dependent of replication of services. This increase the possibility of finding services as it also makes sure to divide the traffic through out the network.

7.4.3 Service migration transparency

7.4.3.1 Test execution
The test of this test is no different than the test of location transparent. See 7.4.1.1

7.4.3.2 Test result
Not relevant

7.4.3.3 Test discussion
The fact that there is no central registration where the object reference is stored makes the system very resistant to problems of migration. Every service is responsible of its own advertisement, and if it is moved to another location the advertising will be moved at the same time.
If a client was connected to the moved service it must once more go through a discovery process to relocate the new location.

7.4.3.4 Test conclusion
JXTA supports migration of services. Since there is no central registration of services, old references is not a problem

7.4.4 Service failure transparency

7.4.4.1 Test execution

We have based our test on persistent data. Since this is not present in a JXTA environment we couldn't test this feature.
The closest to persistent data is the storing of discovery data to disk. Discovery data is the data of other nodes and their location.

7.4.4.2 Test result

Not tested

7.4.4.3 Test discussion

JXTA has no support of failure transparency, if this should be supported it must fully be implemented by the developer. Failure to services will however have small effect on the network if there is more than one available.

The main advantage of P2P networks is that they distribute the responsibility of providing services among all of the peers on the network; this eliminates service outages due to a single point of failure, and provides a more scalable solution for offering services.

7.4.4.4 Test conclusion

JXTA does not support persistent data of any kind. Failure to services will however have small effect on the network if there is a redundant supply.

7.4.5 Technology's market position

7.4.5.1 Discussion

Almost overnight, users everywhere have adopted innovative technologies that enable them to participate in a larger vision of the Internet. Distributed computing applications like SETI@home have empowered millions of users to contribute their computing resources to work on a common computational analysis. Instant messaging services have enabled users to communicate and collaborate with their peers in real time. And true peer-to-peer (P2P) computing, embodied by applications like Napster, Gnutella, and Freenet, has offered a compelling and intuitive way for Internet users to find and share resources directly with each other, often without requiring a central authority or server.

P2P has proven its popularity and its place in the future. Still the use is often limited to search for stolen music and movies, but the fact is that those searching mechanisms could as easily been used to search for web pages or any other kind information.

The vision of the JXTA organization is to gather all the different applications that will be developed on one platform, JXTA. If this will succeed is difficult to predict, but two years later there is still no big applications built on this platform. Rael Dornfest said this in an article he wrote in oreillynet;

"*All told, JXTA provides some remarkable vision and a bare bones yet well-thought-out framework. Its Sun developers and community regulars are a smart, enthusiastic, and hard-working crowd. Yet two years since its inception, still much of what we're seeing coming out of the JXTA project has been at the level of experimentation rather than any real applications. To be fair, JXTA has, from the start, been designated a research project -- occasional media-clip prognostications notwithstanding. And research, in* *our opinion, is what JXTA is and shall remain for the foreseeable future.*"[43]

I doubt the JXTA team will agree to this statement, but there is probably a great deal of truth in it.

JXTA is not currently a “complete” product. In truth, it may never be complete. Like many open-source projects, JXTA is constantly evolving, as open source developers augment the existing reference implementation and specification to address new problems in peer-to-peer computing. This is probably the hope and reason for survival of this technology. The open source foundation gives the possibility of fast evolving the technology into new directions. - Directions which by now are not thought about, but maybe will have great influence on the market.

Sun has already logged 50,000 downloads of the code, and says it has seen considerable interest in helping move the Jxta service beyond Java technology. Maybe JXTA just need some time to mature before it shows its real future [51].

7.4.5.2 Conclusion

There is by now no indication of success of this technology. But the technology is still in progress and may soon evolve into something new and path-breaking.

7.5 Summary

7.5.1 Jini

7.5.1.1 Technical criteria

Criteria	Result of test	Comments
Service location transparency	Supported	The Jini network support service location transparency Templates provide a way to search for a service given its unique ID, the interface the service-proxy implements or the attributes attached to it
Service migration transparency	Supported	Jini support migration of services. Old references are deleted over time with help of a leasing mechanism. To some extend there is also possible to move services in runtime, this future must however be implemented by the developer.
Service replication transparency	Supported	Jini supports replication both to the service and lookup service. JavaSpace gives the opportunity to make a better load balanced system.
Failure transparency	Not supported	Jini core have implemented transaction mechanisms. The lookup service has support of persistent. JavaSpace can be implemented with a persistent space. Client and service has however no support of failure transparency.

Table 7-3 Summary of Jini application tests

7.5.1.2 Market criterion

Table 7-4 Summary Market criterion

Criteria	Result of discussion	Comments
Technology’s market position	Positive future	Jini has been embraced by the mobile companies, Cisco has used it in their switches, DaimlerChrysler is predicting future where their cars are using the Jini network. Problems like a too big memory footprint are solved with the JMatos and, together with J2ME the millions of small devices have now the option to be a part of the Jini community. The open software license Sun is using on this product should also give the foundation of a rich product line maybe as much as 70% as the Gartner group is predicting.

7.5.2 Web Services

7.5.2.1 Technical criteria

Criteria	Result of test	
Service location transparency	Supported	Supported by use of the UDDI registry for lookup.
Service migration transparency	Supported	Services could be migrated between host, but it would require a location update in the UDDI registry
Service replication transparency	Not Supported	Not supported in the scope of web services, but could be incorporated by use of well known cluster technologies developed for web servers.
Service failure transparency	Not Supported	Not supported in the scope of web services, but partly supported by the increased availability gained by using cluster methods mentioned in the replication transparency part of the tests.

Table 7-5 summary of Web Services application tests

7.5.2.2 Market criterion

Criteria	Result of discussion	Comments
Technology's market position	Growing strong	Large companies stake a lot on providing Web Services support in their portfolio.

7.5.3 JXTA

7.5.3.1 Technical criteria

Criteria	Result of test	Comments
Service location transparency	Supported	JXTA supports service location transparency. Searches are done through advertisement in XML and peers that find a mach will send a response.
Service migration transparency	Supported	JXTA supports migration of services. Since there is no central registration of services, old references is not a problem
Service replication transparency	Supported	JXTA network is dependent of replication of services. This increase the possibility of finding services as it also makes sure to divide the traffic through out the network.
Service failure transparency	Not supported	JXTA does not support persistent data of any kind. Failure to services will however have small effect on the network if there is a redundant supply.

Table 7-6 summary of JXTA application tests

7.5.3.2 Market criterion

Criteria	Result of discussion	Comments
Technology's market position	Unclear	There is by now no indication of success of this technology. But the technology is still in progress and may soon evolve into something new and path-breaking.

8 Discussion

The support of dynamic discovery is probably the strongest ability in a Jini network. Services and clients can instantly start communication with the lookup service when they are plugged to the network, this ability makes it a strong candidate in a mobile network.

By using a centralized registry such as a Lookup service, you are guaranteed a match as long as the service is registered; the problem is that you can experience the “single point of failure”. To overcome this problem Jini has introducing the possibility of a redundant source of lookup services. Redundancy is central ability in the Jini network, every entity such as the service, web server and lookup service have the ability to be multiplied, without having negative effect on the network. This makes it possible to divide the traffic through the network and make it more stable in a failure situation.

It does, however come with a cost. The network is built around the multicast protocol. This makes it possible, without any knowledge, to locate every entity in you local area network. Without it you would be bound to a more static system. The problem is, however, that the multicast system doesn’t scale well when it grows to larger network. There are several reasons for this. The multicast protocol is based on UDP and there is no guarantee that they will reach the destination. Packets are often trashed during transport through overloaded network. The second problem is the TTL in the UDP packet. This is a counter which is decreased every time the packet passes a router, once the TTL count is exceeded the message is dropped.

While most modern IP routers support multicast, there is still some machines on the internet that do not. If a router connecting two network segment does not support multicast, then multicast traffic originate on one segment will not be forwarded on to the second segment. Other routers are configured to control the TTL and make sure to drop packets which have to large TTL number. This will effetely reduce the radius were interested parties will hear the messages, and also the possibility of the network to grow larger.
There is however solutions to the problem, a bridge can be built between the two segments such a way that multicast traffic passes between them, it is called a “multicast tunneling” and is fairly easy to set up.

The Jini structure is based on RMI; this gives it the ability of an object orientated platform support. Java object are exchange much in the same way as in a regular java program. The draw back of this is that there is not possible to implement a Jini network in another language than java. This problem can however be overcome by letting the Jini network exchange the proxy object between the client and service, which again can use an another communication platform such as CORBA [44] or wrapping Jini together with ActiveX [45].

Web Services by its XML messaging is well suited for business-to-business applications and integrating new systems with legacy-applications. And it is these types of applications where early adoptions of web services have been seen.

UDDI provides Web Services with a lookup registry which to some extent supports location transparency. The UDDI registry was originally developed for locating services like finding a plumber in the Yellow-pages. Because of the design the registry isn’t suited for more mobile services. This is mainly a result of missing “garbage-collection” in the registry, and the need of a manual re-registering of the service when it has moved to a new location.

Web Services have obtained much attention, around dynamic location and invocation of services based on service descriptions. The problem is that dynamic invocations require common knowledge on how the system has to be invoked. Just agreeing on a set of interfaces isn’t enough. Both parties must have common understanding of what each entity relates to and how the methods should be invoked. Neither the UDDI registry nor the WSDL document supports behavior description.

JXTA strongest and at the same time, weakest ability is the lack of central registration mechanisms, the Lookup mechanism is decentralized to each peer. This gives JXTA solution great freedom when objects are moved or not regularly connected to the network. The drawback is the uncertainty when searching after services. Take a practical example, someone using a JXTA solution for downloading music in Norway may experience a boast of ability to retrieve any old favorite piece at any time. Yet on the very same system, someone living in South Africa may not be able to find any music at all, no matter how frequently they try.

JXTA is built to be flexible; seldom has a system managed this to the extreme which JXTA has. The communication platform is built on XML; this makes it possible to implement the JXTA application in virtually any language, this again will make it possible to implement it on any operative system platform. The fact is that JXTA even allows the developer to choose which communication platform to base it on, from IP to bluetooth or infrared.

JXTA has a lot to offer, but it is still uncertain when and where the first killer application from P2P may come from – and therefore who will ultimately define the “true” meaning of P2P computing, JXTA having a full stack deck, stands a good chance of being the place where exciting future action will occur.

9 Conclusion

9.1 Thesis conclusion

This thesis does a comparative study of ODP distribution transparencies in Jini, Jxta and Web Services. Large parts of the thesis have dealt with the process of identifying the test criteria and develop the prototype for each test. The comparison focuses on the transparency level seen by the developer.

The target area is different for the technologies and because of this a direct comparison of each technology against each other would not be practical. In spite of this, the technologies might overlap in some areas. With this as a starting point, we conclude with the technologies in two main groups: Mobile services and fixed-location services.

A network of mobile entities:
Jini is most suitable for distributed systems based on mobile objects, solely by its naming system. The mixture of dynamic discovery together with garbage collection of old entities makes this system self maintaining and robust.

The latest implementation of the Jini version, JMatos with small memory footprint makes it also a strong candidate in networks of mobile units like PDA's and cellular phones.

Jini offers replication of both the lookup service and the service in order to increase availability. Some fault tolerance is provided by transaction, persistent lookup registry and persistent tuple space.

It is easy to plug new services or resources into the JXTA network. This a result of the distributed search algorithm used to locate services.

Fixed-location services
Web Services is the most suitable technology for systems with fixed-location of services. This technology is fully based on well known technologies used in regular document-based distributed systems. Web Services target area appears to be medium to large business-to-business applications.

Web Services doesn't provide any functions for replication or failure transparency, but depends upon functions implemented in application server where the services are deployed.

9.2 Future Work

There are mainly four subjects left out in this thesis, and thus suitable for future work.

1. The three transparencies, access, relocation and persistence haven't been cover enough in this thesis. In addition to this it would be interesting to do a more thorough study of the transparences at different levels such as the user and application level.

2. Web Services lacks support of failure transparency. Some failure avoidance can be achieved through transaction support which isn't supported yet. There are companies working on implementations of transaction support in Web Services. It would be interesting to study different approaches and solutions for achieving transaction support in Web Services.

3. The Jini network would be suitable for communicating between small devices. In this scope it's interesting to look into if the JMatos is adequate for small devices. Is the memory footprint small enough, and is the computer power on these machines strong enough to support the network?

4. JXTA has a potential to become the platform in P2P networks in the years to come. An interesting study would be to cover what type of services P2P network could cover? Is the potential of P2P as huge as the supporters want it to be?

Abbreviations

API	Application Program Interface
b2b	business-to-business
CORBA	Common Object Request Broker Architecture
DCOM	Distributed Component Object Model
DNS	Domain Name System
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
J2ME	Java 2 Platform, Micro Edition
JDK	Java Development Kit
LUS	Lookup Service
RM-ODP	Open Distributed Processing - Reference model
OSGI	Open System Gateway Initiative
P2P	Peer to Peer
PDA	personal digital assistant
RMI	Remote Method Invocation
SMTP	Simple Mail Transfer Protocol
SOAP	Simple Object Access Protocol
TCP	Transmission Control Protocol
TTL	Time to Live
UDDI	The Universal Description, Discovery, and Integration
UDDI4J	UDDI for Java
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WSDL	Web Services Description Language
WS-I	Web Services Interoperability Organization
WSIF	Web Service Invocation Framework
XML	Extensible Markup Language

Bibliography

[1] Subrahmanyam Allamaraju, Cedric Buest, John Davies, Tyler Jewell, Rod Johnson, Andrew Longshaw, Ramesh Nagappan, Dr P. G. Sarang, Alex Toussaint, Samer Tyagi, Gary Watson, Mark Wilcox, Alan Williamson, Daniel O'Connor
Professional Java Server Programming – J2EE 1.3 Edition,
Wrox Press, ISBN 1-86100-537-7, 2001

[2] James Snell, Dough Tidwell, Pavel Kulchenko
Programming Web Services with Soap,
O'Reilly, ISBN 0-596-00095-2, 2002

[3] **UDDI.ORG** – Web site
http://www.uddi.org
2002.05.10

[4] **SOAP Envelope namespace**
http://www.w3.org/2001/06/soap-envelope
2002.05.10

[5] Mack Hendricks, Ben Galbraith, Romin Irani, James Milbery, Tarak Modi, Andre Tost, Alex Toussaint, S. Jeelani Basha, Scott Cable
Professional Java Web Services,
Wrox Press, ISBN 1-861003-75-7, 2001

[6] **W3C, World Wide Web Consortium**
http://www.w3.org
2002.05.10

[7] Vinosky, Steve
Web Services and Dynamic Lookup
http://www.webservices.org/index.php/article/articleview/66/
2002.04.02

[8] ISO/IEC 10746-1 **Open Distributed Processing – Reference Model**: **Overview**
1996.09.15

[9] W. Keith Edwards
Core Jini second edition
Prentice Hall PTR, ISBN 0-13-089408-7, 2001

[10] W. Keith Edwards, Tom Rodden
Jini example by example
Prentice Hall PTR, ISBN 0-13-033858-3, 2001

[11] Sing Li
Professional Jini
Wrox Press Ltd, ISBN 1-861003-55-2, 2000

[12] Bill Pierce
Building Service Based Architectures with Jini
Dr. Dobb's journal June issue 2001

[13] Steve Hashman, Steven Knudsen
The Application of Jini™ Technology to Enhance the Delivery of Mobile Services

December 2001
http://wwws.sun.com/software/jini/whitepapers/PsiNapticMIDs.pdf
2002.05.06

[14] PsiNaptic press release
PsiNaptic's JMatos™ Software makes Jini™ technology possible on Symbian OS for mobile devices
http://www.symbian.com/news/2002/pr020321.html
2002.05.06

[15] **Spanish Inquisition**
http://wwws.sun.com/software/jini/whitepapers/si_whitepaper.pdf
2002.5.6

[16] **IT Cruiser Telematics Concept**
DaimlerChrysler Corporation, Auburn Hills, Michigan
DaimlerChrysler Research and Technology North America, Palo Alto, California Sun Microsystems, Southfield, Michigan
DaimlerChrysler IT Cruiser Telematics Concept
January 2001
http://java.sun.com/products/consumer-embedded/automotive/whitepapers/ITCruiser-Whitepaper.pdf
2002.05.06

[17] **International Business Machines Corporation and Microsoft Corporation**,
UDDI Technical White Paper
September 2000

[18] Andrew S. Tanenbaum, Maarten Van Steen
Distributed Systems Principles and Paradigms
Prentice-Hall, ISBN 0-13-088893-1, 2002

[19] W3C
Simple Object Application Protocol, working drafts overview
http://www.w3.org/2002/ws
2002.05.14

[20] W3C
Web Services Description Language version 1.1, specification
http://www.w3.org/TR/wsdl
2002.05.14

[21] W3C
Extensible Markup Language, working drafts overview
http://www.w3.org/XML/
2002.05.14

[22] Max Goff
Sun Technology Audio casts
http://developer.java.sun.com/developer/onlineTraining/webcasts/20plus/mgoff.html
2002.05.15

[23] **Colorado software summit**
http://www.softwaresummit.com/
2002.05.10

[24] Department of Computer Science Rice University
The ScalaServer project

http://www.cs.rice.edu/CS/Systems/ScalaServer/
2002.05.13

[25] Coulouris, Dollimore, Kindberg,
Distributed Systems, Concepts and Design, Third edition
Addison-Wesley, ISBN 0-201-61918-0, 2001

[26] **Bea WebLogic Server Cluster Paper**
http://www.bea.com/products/weblogic/server/paper_wls_clustering.pdf
2002.05.15

[27] UBEANS Web Site
Apache/Tomcat cluster solution http://www.ubeans.com/tomcat/
2002.05.12

[28] Peter Deutsch
The eight fallacies of distributed computing
http://java.sun.com/people/jag/Fallacies.html
2002.05.12

[29] Jim Waldo, Geoff Wyant, Ann Wollrath, Sam Kendall
A note on Distributed Computing
Sun Microsystems, 1994

[30] **Leslie Lamport's home page**.
http://lamport.org
2002.05.2002

[31] WebServices.org
Do you think Web Services are just hype, or have huge potential?
http://www.webservices.org/index.php/poll/result/5/
2002.05.15

[32] WebServices.org
What language do you use to develop your Web Services?
http://www.webservices.org/index.php/poll/result/10/
2002.05.15

[33] Ben Pring
Web Services, The Changing Nature of IT Service Opportunity
The Gartner Group
http://www4.gartner.com/DisplayDocument?id=353343&acsFlg=accessBought
2002.05.15

[34] International Business Machines (IBM)
Case study on Storebrand ASA
http://www-3.ibm.com/software/success/cssdb.nsf/CS/NAVO-4ZMTDR?OpenDocument&Site=dmmain
2002.05.14

[35] Scott Durchslag
Beyond the hype... the Reality of Early Web Services Adoption
Web Services Journal, *March 2002 issue*
http://www.wsj2.com
2002.05.14

[36] **AvantGo**

http://avantgo.com/products/mbus_server_app.html
2002.05.14

[37] International Business Machines (IBM)
WebSphere Application Server
http://www-3.ibm.com/software/webservers/
2002.05.18

[38] Microsoft,
.net technology
http://www.microsoft.com/net/
2002.05.18

[39] Bea
Bea WebLogic Server
http://www.bea.com/products/weblogic/server/index.shtml
2002.05.18

[40] Sybase
Application Servers,
http://www.sybase.com/products/applicationservers
2002.05.18

[41] Kammie Kayl
JOY POSES JXTA INITIATIVE Pushing the Boundaries of Distributed Computing
http://java.sun.com/features/2001/02/peer.p.html
19.05.2002

[42] **Project JXTA: An Open, Innovative Collaboration**
http://www.jxta.org/project/www/docs/OpenInnovative.pdf
19.05.2002

[43] Rael Dornfest
Emergin Technology Review, JXTA
http://www.oreillynet.com/pub/a/webservices/2002/03/12/jxta.html
22.03.2002

[44] **JINI Network technology, datasheet**
http://wwws.sun.com/software/jini/whitepapers/jini-datasheet0601.pdf
22.05.2002

[45] **JavaSpaces Technology**
http://java.sun.com/products/javaspaces/
22.05.2002

[46] **Linda Systems**
http://www.cs.yale.edu/HTML/YALE/CS/Linda/linda.html
22.05.2022

[47] **IBM T Space**
http://www.almaden.ibm.com/cs/TSpaces/index.html
22.05.2022

[48] **JXTA Takes Its Position**
http://www.openp2p.com/pub/a/p2p/2001/04/25/jxta_position.html
22.05.2002
[49] Sing Li
A hands-on, working introduction to the latest P2P technology
http://www-106.ibm.com/developerworks/library/j-p2pint1.html

22.05.2002

[50] Sing Li
JXTA, Peer-to-Peer Computing with Java
Wrox Press, ISBN 1-81006-35-7, 2001

[51] John Borland
Sun's Joy rapturous over JXTA
http://news.com.com/2100-1001-267963.html
22.05.2002

Appendix A Test environment

The test environment is put together by 4 machines. The operation system is divided between Win2000 and Linux. System information is given below.

A.1 Computer configuration

Computer 1

Intel Celeron 330 Mhz, 256 MB Ram
ReadHat Linux 7.2 (Enigma) Kernel 2.4.7-10
Java platform is J2SDK 1.4.0
JINI platform is Jini 1.2

Computer 2

Intel Pentium-2 400 Mhz, 392 MB Ram
Windows 2000, workstation
Java platform is J2SDK 1.4.0
JINI platform is Jini 1.2

Computer 3

Intel Pentium-2 400 Mhz, 392 MB Ram
Windows 2000, workstation
Java platform is J2SDK 1.4.0
JINI platform is Jini 1.2
Jakarta-tomcat 4.0.3
Apache SOAP 2.2

Computer 4

K6-2 350 Mhz, 300 MB Ram
RedHat Linux 7.2 (Enigma) Kernel 2.4.7-10
Java platform is J2SDK 1.4.0
JINI platform is Jini 1.2
Jakarta-tomcat 4.0.3
Apache SOAP 2.2

Appendix B – Jini development environment

B.1 Software

B.1.1 Jini Technology Starter Kit v1.2.1

The Jini starter kit can be downloaded for free from sun. It contains everything that is needed to support at Jini network, including install instructions.

http://developer.java.sun.com/developer/products/jini/

B.2 Building test application

B.2.1 Jini prototype

To make the service able to find the lookup service it had to implement a discovery listener. The same process has to be done with the client.

Fortunately the Jini developer pack makes this process really easy you actually just have to implement an interface "DiscoveryListener" and the Jini process takes care of the rest.

```
Public interface DiscoveryListener extends EventListener{
       public void discovered(DiscoveryEvent evt);
       public void discareded(DiscoveryEvent evt);
}//end interface
```

Listing 6 1 DiscoveryListener example

Of course you have to process the information the DiscoveryListener returns. The discovered method will be called whenever a new lookup service is found via the discovery protocols. And the discovery protocol is controlled by a LookupDiscoveryManager. This class is controlling both multicast and unicast forms of discovery, and for doing general housekeeping on the discovery process. So in other words you will also need to implement a LookupDiscoveryManager class to get a Jini network up and running.
Anyway by implementing this you will have an independent Jini network up and running, and the clients and service will find the LUS without any previous configurations.

The leasing mechanism makes the Lookup service more stable. If a service fails it doesn't have to keep record of it for more than the leasing time. And the network will heal it self. This however makes the service depend of renewing the leas time over and over again.

To make this process work we had to implement a LeaseRenewalManager, which handle the whole process. (The leasing time is option by the developer)

Sometimes the service is not available when the clients are searching through the network. The Jini network has implemented an interesting solution to this problem. The clients have the opportunity to be registrar at the lookup service and be given response if the wanted service arrives. This is done with making a class implementing RemoteEventListener.

Since this is based on RMI we had to send the class thorough RMIC to produce the stubs and skeletons file which is required to make a RMI communication to work. The RMIC is a standard implementation of the JDK distribution pack.

One problem that arises is the requirement of the client to have access to the stub file. In a Jini network this is done by letting the LUS (which are the client in this example) download the stub file from a HTTP server. Most of the file transaction in the Jini network is done by HTTP servers.

When the LUS has got the stub object it has the opportunity to send information to the client about newly discovered services.

After a successful discovery process the client will have a list of wanted services which it can choose between. Of course this depends if there are more than one service available!

We wanted the client to be able to switch between the services if the one in use is failing.

To solve this problem we made an object handler which stored all the incoming proxy objects. Among this the client could choose the one it wants to get time reference from. If the one in use fails it simply deleted the object and went for the next. The problem arises when the entire stock of stored object fails, what then?

To solve this problem the discovery process starts over again and this process will continue to on or more suitable service is found.

Making the service able return the exact time of the server, we had to make a new class file. This class implemented the interface BackendProtocol which again meant that the class had to implement the function getServerTime(). This function returns a string that is built up by server address and time reference.

B.2.2 JavaSpace prototype

In one of the criteria we came to the problem where load balancing is an issue. We were struggling by the idée how to solve this in a Jini network. After looking around and reading the Jini newsgroup, - which by the way is excellent, we came to the conclusion that we should use JavaSpace.

Developing the model gave us some problems, mostly because none of the examples that we found in the books worked as they should. - This again gave some frustration since finding the errors can be difficult when references are missing. (The reasons for failure are the differences in Jini distributions)
The JavaSpace prototype became however a reality after some hour with frustration.

We went for a model where the client sends a challenge and the workers/servers are listening at space fetching the object and process them, and sending them back into space where the client could again could fetch them for download.

```
public class TimeTuple implements Entry{
 public String TimeStamp     = null;
 public InetAddress addr     = null;

 public TimeTuple() {}//end constructor


 public TimeTuple(String TimeStamp, InetAddress addr) {
   this.TimeStamp  = TimeStamp;
   this.addr       = addr;
 }//end constructor

 public String getHost(){
   return addr.getHostName();
 }//end getHost

}//end class
```

The TimeTuple, as we chose to call it, is the object that will be used as a messenger in the space. It has implemented Entry as every object in space must have one way or another.

We decided to implement a string which contains the timestamp and an InetAddress. When the client writes a new object to space both of them will have null values. This is also the search criteria for the workers; - any tuple which contains null values must be handled.

The workers will when they have time for it download the objects and update them with a timestamp and the DNS address of the worker and return it to space.

The search criteria for clients fetching TimeTuple from space are when TimeStamp and InetAddress contain no null values. When a client has downloaded an object it will also feed the space with a new empty one, - and the circle can start over again.

B.3 Script

Staring the Jini network you have to run a Lookup service the client and the service, the JavaSpace need even more components to run. Below the scripts will be listed.

B.3.1 Lookup service

B.3.1.1 RMI Activation Daemon (RMID) (Linux)

```
#!/bin/sh
echo Staring RMID with NO execPolicy
echo Log dir is /usr/java/RunScript/JINI/Log

rmid -J-Dsun.rmi.activation.execPolicy=none -log /usr/java/RunScript/JINI/Log/
```

B.3.1.2 Web Server (Linux)

```
#!/bin/sh

echo Starting Web server at port 8081
echo Source at $JINI_HOME/lib

java -jar $JINI_HOME/lib/tools.jar -port 8081 -dir $JINI_HOME/lib/ -trees -verbose
```

B.3.1.3 Reggie (Linux)

```
#!/bin/sh
echo Staring Reggie....
```

```
HOSTNAME=128.39.203.37

POLICY=$JINI_HOME/policy/policy.all
JARFILE=$JINI_HOME/lib/reggie.jar
CODEBASE=http://$HOSTNAME:8081/reggie-dl.jar
LOG_DIR=/usr/java/RunScript/JINI/Log/reggie_log
GROUP=public

java -Dcom.sun.jini.reggie.proxy.debug=true -jar $JARFILE $CODEBASE $POLICY
$LOG_DIR $GROUP
```

B.3.2 JavaSpace

Staring the JavaSpace we need two services; Mahalo and Outrigger. Mahalo is the Sun's implementation of the transaction manager service while Outerigger is the Sun's implementation of JavaSpace.
Outrigger is in two different implementations one transient and one persistent. The persistent version will make sure to store the entire stock of object that is written to space, to disk. If the system should fail a reboot, the object will still be present.

B.3.2.1 Mahalo (Linux)

```
#!/bin/sh

echo Staring Mahalo......

# Set this to wherever the webserver is running
HOSTNAME=128.39.203.37

# everything below should work with few changes
POLICYFILE=$JINI_HOME/policy/policy.all
JARFILE=$JINI_HOME/lib/mahalo.jar
CODEBASE=http://$HOSTNAME:8081/mahalo-dl.jar
TXN_POLICYFILE=$POLICYFILE
LOG_DIR=/usr/java/RunScript/JINI/Log/txn_log
GROUP=public


java -jar -Djava.security.policy=$POLICYFILE -
Dcom.sun.jini.mahalo.managerName=TransactionManager $JARFILE
$CODEBASE $TXN_POLICYFILE $LOG_DIR $GROUP
```

B.3.2.2 Transient Outrigger (Linux)

```
#!/bin/sh

echo Starting transient Outrigger.....

# Set this to wherever the webserver is running
HOSTNAME=128.39.203.37


# everything below should work with few changes
POLICYFILE=$JINI_HOME/policy/policy.all
JARFILE=$JINI_HOME/lib/transient-outrigger.jar
CODEBASE=http://$HOSTNAME:8081/outrigger-dl.jar
LOG_DIR=/usr/java/RunScript/JINI/Log/js_log
GROUP=public


java -jar -Djava.security.policy=$POLICYFILE -
Djava.rmi.server.codebase=$CODEBASE -
Dcom.sun.jini.outrigger.spaceName=JavaSpaces $JARFILE $GROUP
```

B.3.2.3 Persistent Outrigger (Linux)

```
#!/bin/sh

echo Starting transient Outrigger.....

# Set this to wherever the webserver is running
HOSTNAME=128.39.203.37


# everything below should work with few changes
POLICYFILE=$JINI_HOME/policy/policy.all
JARFILE=$JINI_HOME/lib/outrigger.jar
CODEBASE=http://$HOSTNAME:8081/outrigger-dl.jar
LOG_DIR=/usr/java/RunScript/JINI/Log/js_log
GROUP=public


java -jar -Djava.security.policy=$POLICYFILE -
Djava.rmi.server.codebase=$CODEBASE -
Dcom.sun.jini.outrigger.spaceName=JavaSpaces $JARFILE $GROUP
```

B.3.3 Client starting script

B.3.3.1 Clock client Jini (Linux)

```
#!/bin/sh

HOSTNAME=128.39.202.156:8086
CLASSFILES=    $JINI_HOME/lib/jini-core.jar:$JINI_HOME/lib/jini-ext.jar:\
               $JINI_HOME/lib/sun-util.jar:/usr/java/RunScript/JINI

java -cp $CLASSFILES -Djava.rmi.server.codebase=http://$HOSTNAME/ -
Djava.security.policy=$JINI_HOME/policy/policy.all prototypejini.ClockClient
```

B.3.3.2 Clock client JavaSpace (Linux)

```
java -Djava.security.policy=d:\java\jini1_2\policy\policy.all -
Doutrigger.spacename=JavaSpaces -Dcom.sun.jini.lookup.groups=public -cp
d:\java\jini1_2\lib\jini-core.jar;d:\java\jini1_2\lib\jini-ext.jar;d:\java\jini1_2\lib\sun-
util.jar;d:\Test\Space2 javaspacetimecontrol.TimeClient.ClientStart
```

B.3.4 Service starting script

B.3.4.1 Clock service Jini (Linux)

```
#!/bin/sh

echo Java home: $JAVA_HOME
echo Jini home: $JINI_HOME

CP=$JINI_HOME/lib/jini-core.jar:$JINI_HOME/lib/jini-ext.jar:$JINI_HOME/lib/sun-
util.jar:service

$JAVA_HOME/bin/java -cp $CP -
Djava.rmi.server.codebase=http://sgrm348.grm.hia.no:8085/ -
Djava.security.policy=$JINI_HOME/policy
```

B.3.4.2 Clock service JavaSpace (Windows)

```
java -Djava.security.policy=%JINI_HOME%\policy\policy.all -
Doutrigger.spacename=JavaSpaces -Dcom.sun.jini.lookup.groups=public -cp
%JINI_HOME%\lib\jini-core.jar;%JINI_HOME%\lib\jini-
ext.jar;%JINI_HOME%\lib\sun-util.jar;.
javaspacetimecontrol.TimeService.ServiceStart
```

B.4 Source code

B.4.1 Clock Client Jini

B.4.1.1 ClockClient

```
import net.jini.discovery.*;
import net.jini.core.lookup.*;
import net.jini.core.event.*;
import java.io.*;
import java.rmi.*;
import java.util.*;
public class ClockClient implements Runnable{

  private ServiceTemplate          template;
  private LookupDiscovery         disco;
  private final int               LEASE_TIME = 10*60*1000;
  private MyEventListener         eventCatcher;
  private objectHandler           objecthandler;
  private MyListener              listener;
  private ClockFrame              clientFrame;

  public ClockClient() throws IOException{
        Class[] type  = {ClockServiceInterface.class};
        template      = new ServiceTemplate(null,type,null);
        objecthandler = new objectHandler();

        if(System.getSecurityManager() == null){
            System.setSecurityManager(new RMISecurityManager());
        }//end if


        disco         = new LookupDiscovery(new String[] {""});
        listener      = new MyListener(this,template,objecthandler);
        disco.addDiscoveryListener(listener);
        eventCatcher  = new MyEventListener(objecthandler);

        clientFrame   = new ClockFrame();
  }//end constructor

  public void run(){
      while(true){
          try{
              Thread.sleep(1*1*1000);
              getTime();
          }catch(InterruptedException ex){}
      }//end while
  }//End run

  public void getTime(){
      Vector vector = objecthandler.getVector();
      if(vector.size()>0){
          try{
              ClockServiceInterface testInter = (ClockServiceInterface)vector.lastElement();
              clientFrame.setLabelTekst(testInter.getTime());

          }catch(Exception e){
              System.out.println("Error in Vector.. element removed!! " + e);
              vector.removeElement(vector.lastElement());
          }//End try/catch

      }//end if
      else{
        disco.removeDiscoveryListener(listener);
        listener      = new MyListener(this,template,objecthandler);
        disco.addDiscoveryListener(listener);
      }//end else
  }//end getTime


  public void removeDisco(ServiceRegistrar lusvc){
      disco.discard(lusvc);
  }//end temoveDisco


  public void registerForEvents(ServiceRegistrar lu) throws RemoteException{

lu.notify(template,ServiceRegistrar.TRANSITION_NOMATCH_MATCH,eventCatcher,null,LEASE_TIME);
  }//end registerForEvents


  public static void main(String[] args) {
    try{
        ClockClient clockClient1 = new ClockClient();
        Thread thread = new Thread(clockClient1);
        thread.start();
    }catch(IOException e){
          System.out.println("Error in constructor: "+ e.getMessage() );
    }//end try/catch
  }//end main
}//end class
```

B.4.1.2 ClockFrame

```
import java.awt.*;
import javax.swing.*;
import javax.swing.border.*;

public class ClockFrame extends JFrame{
  private JPanel jPanel1 = new JPanel();
  private JLabel jLabel1 = new JLabel();
  private TitledBorder    titledBorder1;
  private String          labelTekst;

  public ClockFrame() {
      makeFrame();

      this.setSize(400,100);
      this.setLocation(200,200);
      this.show();


  }//end constructor

  public void setLabelTekst(String tekst){
      jLabel1.setText(tekst);
  }//end setLabelTekst

  public void makeFrame(){
      titledBorder1 = new TitledBorder("JINI Clock client");
      jPanel1.setBorder(titledBorder1);
      jLabel1.setText(labelTekst);
      this.getContentPane().add(jPanel1, BorderLayout.CENTER);
      jPanel1.add(jLabel1, null);
  }//end makeFrame
}//end class
```

B.4.1.3 ClockServiceInterface

```
public interface ClockServiceInterface {
    public String getTime() throws java.rmi.RemoteException;
}//end interface
```

B.4.1.4 MyEventListener

```
import java.rmi.server.*;
import java.rmi.RemoteException;
import net.jini.core.event.*;
import net.jini.core.lookup.*;
import java.util.*;

public class MyEventListener extends UnicastRemoteObject implements RemoteEventListener{
  private objectHandler objecthandler;
  public MyEventListener(objectHandler p_objecthandler) throws RemoteException{
      objecthandler = p_objecthandler;
  }//end constructor

  public void notify(RemoteEvent evt)throws RemoteException, UnknownEventException{
        System.out.println("Got an event from: " + evt.getSource());

        if(evt instanceof ServiceEvent){
            ServiceEvent sev          = (ServiceEvent)evt;
            ServiceItem item          = sev.getServiceItem();
            ClockServiceInterface hws = (ClockServiceInterface)item.service;
            System.out.println("Got a service!");
            if(hws != null){
                objecthandler.addObject(hws);
            }//end if
        }//end if
        else{
            System.out.println("Not a service event, ignoring");
        }//end else
  }//end notify
}//end class
```

B.4.1.5 MyListener

```
import net.jini.discovery.*;
import net.jini.core.lookup.*;
import java.io.*;
import java.rmi.*;

public class MyListener implements DiscoveryListener{
    private ClockClient       parent;
    private ServiceTemplate   template;
    private objectHandler     objecthandler;

    public MyListener(ClockClient p_parent,ServiceTemplate p_template,objectHandler
p_handler) {
        parent        = p_parent;
        template      = p_template;
        objecthandler = p_handler;
    }//end constrctor


    public void discovered(DiscoveryEvent evt){
        ServiceRegistrar[] newregs = evt.getRegistrars();
        for(int a=0;a<newregs.length;a++){
            if(lookForService(newregs[a]) != null){
                objecthandler.addObject((ClockServiceInterface)lookForService(newregs[a]));
                System.out.println("New element added:!");
            }//end if
        }//end for
    }//end discovered

    public void discarded(DiscoveryEvent evt){

    }//end discarded


  protected Object lookForService(ServiceRegistrar lusvc){
      Object object = null;
      try{
          object    = lusvc.lookup(template);
      }catch(RemoteException e){
          return null;
      }//end try/catch

      if(object == null){
          System.out.println("No matching services");
          try{
              parent.registerForEvents(lusvc);
          }catch(RemoteException e){
              System.out.println("Error when registrating event: " + e.getMessage());
              parent.removeDisco(lusvc);
          }//end try/catch
          return null;
      }//end if
      return object;
  }//end lookupService
}//end class
```

B.4.1.6 objectHandler

```
import java.util.*;


public class objectHandler {
  private Vector objectVector;
  public objectHandler() {
      objectVector = new Vector();
  }//end constructor

  public Vector getVector(){
      return objectVector;
  }//end getVector

  public ClockServiceInterface getFirst(){
      return (ClockServiceInterface)objectVector.firstElement();
  }//end getFirst

  public void addObject(ClockServiceInterface object){
      objectVector.addElement(object);
  }//end addObject

  public void removeObject(ClockServiceInterface object){
      objectVector.remove(object);
  }//end removeObject
}//End class
```

B.4.2 Clock Service Jini

B.4.2.1 Backend

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.*;
import java.net.*;

public class Backend extends UnicastRemoteObject
  implements BackendProtocol {

  public static final boolean DEBUG = true;
  private long count = 60;

  public Backend() throws RemoteException {
    if(DEBUG)
      System.out.println("Backend created");
  }

  public String getServerTime() throws RemoteException {
    String hostname = null;

    try {
      InetAddress addr = InetAddress.getLocalHost();
      hostname = addr.getHostName();
    } catch (Exception ex) {}
    if(DEBUG & (count++ % 60 == 0))
      System.out.print("#");

    Date now = new Date(System.currentTimeMillis());
    return now.toString() + " " + hostname;
  }
}
```

B.4.2.2 BackendProtocol

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface BackendProtocol extends Remote {

  public String getServerTime() throws RemoteException;
}
```

B.4.2.3 ClockService

```
import net.jini.discovery.*;
import net.jini.core.discovery.LookupLocator;
import net.jini.core.lookup.*;
import net.jini.lease.*;
import net.jini.core.lease.*;
import java.util.*;
import java.io.*;
import java.rmi.*;


public class ClockService implements Runnable, LeaseListener {

  private final int LEASE_TIME = 1*1000;
  private HashMap registrations = new HashMap();
  private ServiceItem item;
  private LookupDiscovery disco;
  protected LeaseRenewalManager lrm;


  public ClockService() throws IOException {
    item = new ServiceItem(null, createProxy(), null);
    if(System.getSecurityManager() == null)
      System.setSecurityManager(new RMISecurityManager());

    disco = new LookupDiscovery(new String[] {""});
    disco.addDiscoveryListener(new Listener());
    lrm = new LeaseRenewalManager();
  }

  public ClockServiceInterface createProxy() {

    try {
      Backend backend = new Backend();
      return new ClockServiceProxy(backend);
    } catch (RemoteException ex) {
      System.out.println("Error creating backend: " +
        ex.getMessage());
      System.exit(1);
      return null;
    }
  }

  synchronized void registerWithLookup(ServiceRegistrar registrar) {

    ServiceRegistration registration = null;
    try {
      registration = registrar.register(item, LEASE_TIME);
    } catch (RemoteException ex) {
      System.out.println("Couldn't register: " + ex.getMessage());
      return;
    }

    if(item.serviceID == null) {
      item.serviceID = registration.getServiceID();
      System.out.println("set service id to " + item.serviceID);
    }

    registrations.put(registrar, registration);
    lrm.renewFor(registration.getLease(), Lease.FOREVER, this);
  }

  public void run() {

    while(true) {
      try {
        Thread.sleep(1000000);
      } catch (InterruptedException ex) {
      }
    }
  }

  public void notify(LeaseRenewalEvent evt) {

    System.out.println("Couldn't renew lease: " + evt.getLease());
  }

  public static void main(String args[]) {

    try {
      ClockService clockService = new ClockService();
      new Thread(clockService).start();
    } catch (IOException ex) {
      System.out.println("Failed to start service: " + ex.getMessage());
    }
  }


  class Listener implements DiscoveryListener {
    public void discovered(DiscoveryEvent evt) {

      System.out.println("Discovered lookup service: ");
      ServiceRegistrar[] newRegs = evt.getRegistrars();

      for(int i = 0;i < newRegs.length;i++) {
        try {
          LookupLocator loc = newRegs[i].getLocator();
          System.out.println(loc.getHost() + ":" + loc.getPort());
        } catch (RemoteException ex) {
        }

        if(!registrations.containsKey(newRegs[i]))
          registerWithLookup(newRegs[i]);
      }
    }

    public void discarded(DiscoveryEvent evt) {

      ServiceRegistrar[] deadRegs = evt.getRegistrars();
      for(int i = 0;i < deadRegs.length;i++)
        registrations.remove(deadRegs[i]);
    }
  }
}
```

B.4.2.4 ClockServiceInterface

```
public interface ClockServiceInterface {
    public String getTime() throws java.rmi.RemoteException;
}
```

B.4.2.5 ClockServiceProxy

```
import java.io.Serializable;
import java.rmi.Remote;
import java.rmi.RemoteException;

public class ClockServiceProxy implements Serializable, ClockServiceInterface {

  protected BackendProtocol backend;

  public ClockServiceProxy() {
  }

  public ClockServiceProxy(BackendProtocol backend) {
    this.backend = backend;
  }

  public String getTime() throws RemoteException {

      return backend.getServerTime();
  }
}
```

B.4.3 Clock Client JavaSpace

B.4.3.1 ClientStart

```
import net.jini.lookup.ServiceDiscoveryManager;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.core.lookup.ServiceItem;
import net.jini.space.JavaSpace;
import net.jini.lease.LeaseListener;
import net.jini.lease.LeaseRenewalEvent;
import net.jini.core.lease.Lease;
import com.sun.jini.lease.landlord.LandlordLease;
import net.jini.core.transaction.server.TransactionManager;
import net.jini.core.transaction.Transaction;
import net.jini.core.transaction.TransactionFactory;

import java.rmi.RMISecurityManager;
import java.util.Vector;

import java.io.*;

import net.jini.discovery.*;
import net.jini.core.lookup.*;

import javaspacetimecontrol.TimeTuple;
import java.net.*;

public class ClientStart implements Runnable,ListenerInterface{
  private ServiceTemplate     template;
  private MyListener          listener;
  private LookupDiscovery     disco;
  private boolean Registrer = true;
  public ClientStart() throws IOException{
      if(System.getSecurityManager() == null){
          System.setSecurityManager(new RMISecurityManager());
      }//end if

      Class[] type  = new Class[] {JavaSpace.class };
      template      = new ServiceTemplate(null,type,null);

      disco         = new LookupDiscovery(new String[] {""});
      listener      = new MyListener(template,this);
```

```
        disco.addDiscoveryListener(listener);

    }//end constructor

    public void HandleSpaceObject(JavaSpace jsp){
        if(Registrer){
              System.out.println("Starting Thread");
              ThreadController controller = new ThreadController(jsp);
              Thread thread               = new Thread(controller);
              thread.start();
              Registrer = false;
        }//end if
        else{
              System.out.println("drop to registrar");
        }//end else
    }//end HandleSpaceObject


    public void run(){
        while(true){
            try{
                Thread.sleep(10*1*1000);
            }catch(InterruptedException ex){}
        }//end while
    }//end run

    public static void main(String[] args) {
      try{
          ClientStart serviceStart1 = new ClientStart();
          Thread thread = new Thread(serviceStart1);
          thread.start();
      }catch(IOException e){
            System.out.println("Feil i constructor: "+ e.getMessage() );
      }//end try/catch
    }//end main
}//end class
```

B.4.3.2 ListenerInterface

```
import net.jini.space.JavaSpace;
public interface ListenerInterface {
    public void HandleSpaceObject(JavaSpace jspace);
}//end interface
```

B.4.3.3 MyListener

```
import net.jini.discovery.*;
import net.jini.core.lookup.*;
import java.io.*;
import java.rmi.*;
import net.jini.space.JavaSpace;


public class MyListener implements DiscoveryListener{
    private ServiceTemplate   template;
    private JavaSpace         javaSpace = null;
    private ListenerInterface parent;
    public MyListener(ServiceTemplate p_template,ListenerInterface p_parent) {
        template  = p_template;
        parent    = p_parent;
    }//end constrctor


    public void discovered(DiscoveryEvent evt){
        ServiceRegistrar[] newregs = evt.getRegistrars();
        for(int a=0;a<newregs.length;a++){
              JavaSpace js = findJavaSpace(newregs[a]);
              if(js !=null){
                  try{
                     System.out.println("URL:" + getURL(newregs[a]));
                     System.out.println("Groups:" + getGroups(newregs[a]));
                  }catch(RemoteException e){
                      System.out.println("Feil med henting av URL: "+ e);
                  }//end try/catch
                  parent.HandleSpaceObject(js);
              }//end if
        }//end for
    }//end discovered

    public void discarded(DiscoveryEvent evt){}//end discarded

    public String getURL(ServiceRegistrar reg) throws RemoteException{
        return reg.getLocator().toString();
    }//end getURL

    public String getGroups(ServiceRegistrar reg) throws RemoteException{
        String[] groups = reg.getGroups();
        if(groups == DiscoveryGroupManagement.ALL_GROUPS){
            return "<ALL GROUPS>";
        }//end if
        if(groups == DiscoveryGroupManagement.NO_GROUPS){
            return "<NONE>";
        }//end if

        StringBuffer buf = new StringBuffer();

        for(int a=0;a<groups.length;a++){
            if(groups[a] == null){
                buf.append("NULL");
            }//end if
            else if(groups[a].equals("")){
                buf.append("PUBLIC");
            }//End else if
            else{
              buf.append(groups[a]);
            }//end else
        }//end for
        return buf.toString();
    }//end getURL

    public JavaSpace findJavaSpace(ServiceRegistrar reg){
      if(javaSpace != null){
          return null;
      }//end if

      try{
          JavaSpace midJavaSpace = (JavaSpace)reg.lookup(template);
          if(midJavaSpace != null){
              System.out.println("Found a Space!");

              return midJavaSpace;
          }//end if
      }catch(RemoteException e){
          System.out.println("Error doing lookup: " + e.getMessage());
      }//end try/catch
      return null;
    }//end findJavaSpace


}//end class
```

B.4.3.4 ThreadController

```
import net.jini.space.*;
import javaspacetimecontrol.TimeTuple;
import java.net.*;
import java.util.*;
import net.jini.core.lease.Lease;

public class ThreadController implements Runnable{
  private JavaSpace jspace;
  private TimeTuple time;
  private TimeTuple tuple;

  public ThreadController(JavaSpace jspace) {
      this.jspace = jspace;
      try {
          jspace.write(new TimeTuple(),null, Lease.FOREVER);
          System.out.println("Object er registrert");
      } catch (Exception e) {
              System.out.println ("write to Space exception: " + e);
      }//end try/catch
  }//end constructor

  public synchronized void readFromSpace(){
    try {
          if((tuple = (TimeTuple)jspace.read(new
TimeTuple(null,null),null,JavaSpace.NO_WAIT)) != null){
              if(tuple.TimeStamp != null){
                  jspace.take(tuple,null,JavaSpace.NO_WAIT);
                  System.out.println("Time found: " + tuple.TimeStamp);
                  System.out.println("Host: "       + tuple.getHost());
                  jspace.write(new TimeTuple(),null, Lease.FOREVER);
              }//end if
              else{
                  System.out.println("Fant ingen som ikk er null");
              }//end else
          }//end if
          else{
              System.out.println("Fant ikke objekt");
          }//end else
      } catch (Exception e) {
              System.out.println ("read from Space exception: " + e);
      }//end try/catch
  }//End readFromSpace


  public void run(){
      while(true){
          try{
              Thread.sleep(1*1*1000);
              readFromSpace();
          }catch(InterruptedException ex){}
      }//end while
  }//end run
}//end class
```

B.4.4 Clock Worker JavaSpace

B.4.4.1 ListenerInterface

```
import net.jini.space.JavaSpace;
public interface ListenerInterface {
    public void HandleSpaceObject(JavaSpace jspace);
}//end interface
```

B.4.4.2 MyListener

```
import net.jini.discovery.*;
import net.jini.core.lookup.*;
import java.io.*;
import java.rmi.*;
import net.jini.space.JavaSpace;


public class MyListener implements DiscoveryListener{
    private ServiceTemplate   template;
    private JavaSpace         javaSpace = null;
    private ListenerInterface parent;
    public MyListener(ServiceTemplate p_template,ListenerInterface p_parent) {
        template  = p_template;
        parent    = p_parent;
    }//end constrctor


    public void discovered(DiscoveryEvent evt){
        ServiceRegistrar[] newregs = evt.getRegistrars();
        for(int a=0;a<newregs.length;a++){
              JavaSpace js = findJavaSpace(newregs[a]);
              if(js !=null){
                  try{
                     System.out.println("URL:" + getURL(newregs[a]));
                     System.out.println("Groups:" + getGroups(newregs[a]));
                  }catch(RemoteException e){
                      System.out.println("Feil med henting av URL: "+ e);
                  }//end try/catch
                  parent.HandleSpaceObject(js);
              }//end if
        }//end for
    }//end discovered

    public void discarded(DiscoveryEvent evt){}//end discarded

    public String getURL(ServiceRegistrar reg) throws RemoteException{
        return reg.getLocator().toString();
    }//end getURL

    public String getGroups(ServiceRegistrar reg) throws RemoteException{
        String[] groups = reg.getGroups();
        if(groups == DiscoveryGroupManagement.ALL_GROUPS){
            return "<ALL GROUPS>";
        }//end if
        if(groups == DiscoveryGroupManagement.NO_GROUPS){
            return "<NONE>";
        }//end if

        StringBuffer buf = new StringBuffer();

        for(int a=0;a<groups.length;a++){
            if(groups[a] == null){
                buf.append("NULL");
            }//end if
            else if(groups[a].equals("")){
                buf.append("PUBLIC");
            }//End else if
            else{
              buf.append(groups[a]);
            }//end else
        }//end for
        return buf.toString();
    }//end getURL

    public JavaSpace findJavaSpace(ServiceRegistrar reg){
      if(javaSpace != null){
          return null;
      }//end if

      try{
          JavaSpace midJavaSpace = (JavaSpace)reg.lookup(template);
          if(midJavaSpace != null){
              System.out.println("Found a Space!");

              return midJavaSpace;
          }//end if
      }catch(RemoteException e){
          System.out.println("Error doing lookup: " + e.getMessage());
      }//end try/catch
      return null;
    }//end findJavaSpace


}//end class
```

B.4.4.3 ServiceStart

```
import net.jini.lookup.ServiceDiscoveryManager;
import net.jini.core.lookup.ServiceTemplate;
import net.jini.core.lookup.ServiceItem;
import net.jini.space.JavaSpace;
import net.jini.lease.LeaseListener;
import net.jini.lease.LeaseRenewalEvent;
import net.jini.core.lease.Lease;
import com.sun.jini.lease.landlord.LandlordLease;
import net.jini.core.transaction.server.TransactionManager;
import net.jini.core.transaction.Transaction;
import net.jini.core.transaction.TransactionFactory;
import java.rmi.RMISecurityManager;
import java.util.*;

import java.io.*;

import net.jini.discovery.*;
import net.jini.core.lookup.*;
import java.net.*;

import javaspacetimecontrol.TimeTuple;


public class ServiceStart implements ListenerInterface,Runnable{
  private ServiceTemplate   template;
  private MyListener        listener;
  private LookupDiscovery   disco;

  public ServiceStart() throws IOException{
      if(System.getSecurityManager() == null){
          System.setSecurityManager(new RMISecurityManager());
      }//end if

      Class[] type  = new Class[] {JavaSpace.class };
      template      = new ServiceTemplate(null,type,null);

      disco         = new LookupDiscovery(new String[] {""});
      listener      = new MyListener(template,this);
      disco.addDiscoveryListener(listener);
  }//end constructor

  public void HandleSpaceObject(JavaSpace jsp){
      ThreadController controller = new ThreadController(jsp);
      Thread thread               = new Thread(controller);
      thread.start();
      System.out.println("Started a new Thread to handler Space");
  }//end HandleSpaceObject


  public void run(){
      while(true){
          try{
              Thread.sleep(10*1*1000);
          }catch(InterruptedException ex){}
      }//end while
  }//end run

  public static void main(String[] args) {
    try{
        ServiceStart serviceStart1  = new ServiceStart();

        Thread thread               = new Thread(serviceStart1);
        thread.start();
    }catch(IOException e){
          System.out.println("Feil i constructor: "+ e.getMessage() );
    }//end try/catch
  }//end main
}//end class
```

B.4.4.4 ThreadController

```
import net.jini.space.*;
import javaspacetimecontrol.TimeTuple;
import java.net.*;
import java.util.*;
import net.jini.core.lease.Lease;

public class ThreadController implements Runnable{
  private JavaSpace jspace;
  private TimeTuple time;
  public ThreadController(JavaSpace jspace) {
      this.jspace = jspace;
  }//end constructor

  public synchronized void registarInSpace(){
        try {
           if  (jspace != null){
                InetAddress addr  = InetAddress.getLocalHost();
                Date now          = new Date(System.currentTimeMillis());

                if((time = (TimeTuple)jspace.read(new TimeTuple(),null,JavaSpace.NO_WAIT))
!= null){
                    if(time.TimeStamp == null){
                        jspace.take(time,null,JavaSpace.NO_WAIT);
//                        System.out.println("Empty TimeTuple, removed: " + time.TimeStamp);
                        jspace.write(new TimeTuple(now.toString(),addr),null, Lease.FOREVER
);
                        System.out.println("New TimeTuple added: " + now.toString());
                    }//end if
                }//End if
           }//end if
           else{
              System.out.println("Object is null!!!");
           }//end else
        } catch (Exception e) {
                System.out.println ("Roombuilder:main() exception: " + e);
        }//end try/catch
  }//end registarInSpace


  public void emptySpace(){
        try {
           if  (jspace != null){
                InetAddress addr  = InetAddress.getLocalHost();
                Date now          = new Date(System.currentTimeMillis());

                if((time = (TimeTuple)jspace.take(new TimeTuple(),null,JavaSpace.NO_WAIT))
!= null){
                    System.out.println("Empty TimeTuple, removed: " + time.TimeStamp);
                }//End if
           }//end if
           else{
              System.out.println("Object is null!!!");
           }//end else
        } catch (Exception e) {
                System.out.println ("Roombuilder:main() exception: " + e);
        }//end try/catch
  }//end registarInSpace

  public void run(){
      while(true){
          try{
              Thread.sleep(1*1*1000);
              registarInSpace();
//              emptySpace();
          }catch(InterruptedException ex){}
      }//end while
  }//end run
}//end class
```

Appendix C – Web Services development environment

C.1 Software

Jakarta Tomcat v4.0.3 (jakarta.apache.org/tomcat)
Apache SOAP v2.2 (xml.apache.org/soap)
UDDI4J v2 beta (www.uddi4j.org)
WSIF v1.0 (alphaworks.ibm.com/tech/wsif)
Jakarta Ant v1.4.1 (jakarta.apache.org/ant)

C.2 Building test application

The implementation in this thesis uses Apache SOAP v2.2 toolkit and IBM UDDI for Java v2.0b (UDDI4J).

First we started using Sun Microsystems Java Web Services Development Pack ea2, which had everything from a SOAP implementation to a UDDI registry and a tomcat server bundled. But we struggled a lot because of tool immaturity and lack of documentation. The pack actually contained a lot of documentation, but at a very course art.

Jakarta-ant is used for building all sources, deploying the service and running the samples. In this section we only refer to the different build tasks. See the `build.xml` make file for details on each build task.

The service is a simple clock which returns the current time plus the name of the computer where the service is located. The clock is implemented in a class `Clock` which includes one method `getTime`.

Apache SOAP runs as a servlet inside a Java HTTP server. To invoke the web service, the class files have to be copied into the web server's classpath and deployed using the SOAP Admin application[9]. Information about implementation class and methods exposed is described in a simple deployment descriptor; `clock_dd.xml`.

This ant task was used for web service deployment.

```
  ant deploy-clock
```

Now the service is deployed and ready for receiving SOAP calls through the soaprcp router.

The client uses UDDI4J, WSIF and SOAP for respectively lookup, bind and invocation. Two different clients were developed, and yet a third was used for testing the WSIF dynamic invocation.

The first client uses a proxy system to invoke the web service. First a generic `SOAPProxy` class was developed. This class could make SOAP calls based on names i.e.

```
    Object soapCall(String method, Vector params)
```

The SOAPProxy does wrap standard soap calls, and is only developed for convenience, since a SOAP call requires quite a lot of code.

[9] Apache SOAP actually supports two different methods for deployment, but we only use the simple command line method in this thesis.

Then a proxy matching the Clock service was developed. This proxy extends the `SOAPProxy` and exposes the `getTime` method from the Clock service.

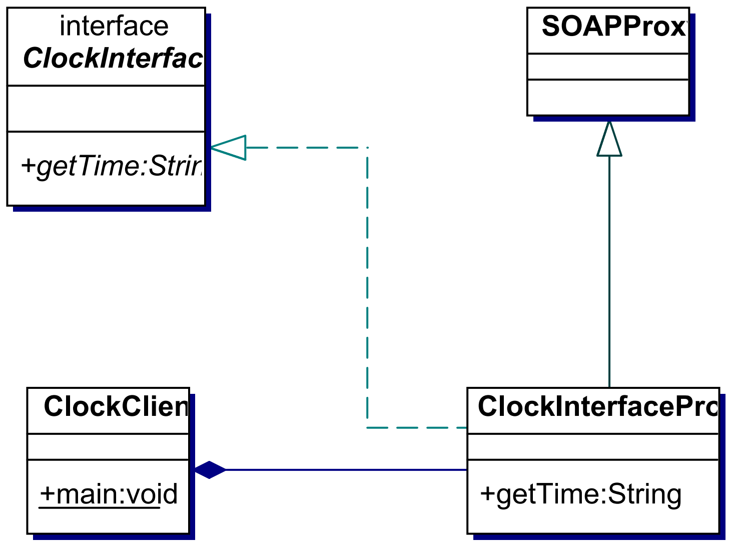


Figure 9-1 - Clock client, conceptual model

The second client is nearly identical to the first one, but the SOAP proxy is redesigned to be a "UDDI-aware" proxy which searches for the service in a UDDI register before the first invocation.

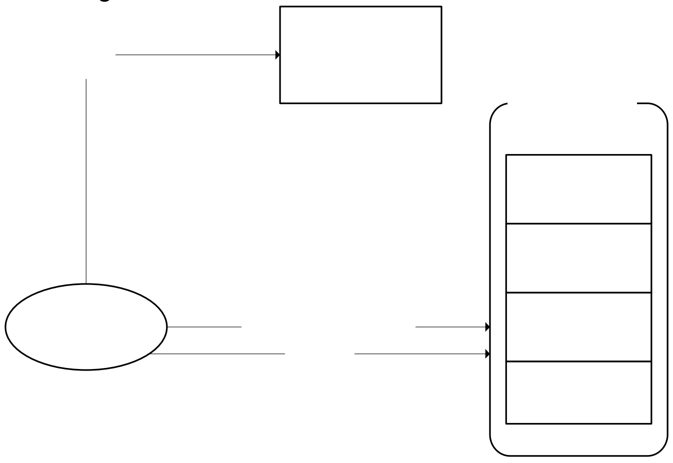


Figure 9-2 - Invocation procedure for the clock client (WSIF)

C.3 Script

C.3.1 Clock service's WSDL document

```
<?xml version="1.0" ?>
<!--
  $Id$

  WSDL description for the ClockService
-->
<definitions name="ClockService"
  targetNamespace="urn:ClockService"
  xmlns:tns="urn:ClockService"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <!--
    Definition of user defined types used
    (no-one in our case; using only built in string type)
  -->
  <types/>

  <!--
    Definition of the messages used when invoking methods
  -->
  <message name="getTime" />
  <message name="getTimeResponse">
    <part name="result" type="xsd:string"/>
  </message>

  <!--
    Port definitons
    Messages are mapped to methods
  -->
  <portType name="ClockIF">
    <operation name="getTime">
      <input message="tns:getTime"/>
      <output message="tns:getTimeResponse"/>
    </operation>
  </portType>

  <!--
    Bindings maps port operations to network invokation
    protocols such as SOAP
  -->
  <binding name="ClockIFBinding" type="tns:ClockIF">
    <operation name="getTime">
      <input>
        <soap:body
encodingStyle=http://schemas.xmlsoap.org/soap/encoding/”
use="encoded" namespace="urn:ClockService"/>
      </input>
      <output>
        <soap:body
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
use="encoded" namespace="urn:ClockService"/>
      </output>
      <soap:operation soapAction=""/>
    </operation>
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
        style="rpc"/>
  </binding>

  <!--
    Service elements constains the locataion(s) of a service
    implementation on the network.
  -->
  <service name="urn:ClockService">
    <port name="ClockIFPort" binding="tns:ClockIFBinding">
      <soap:address
        location="http://diplom.mine.nu:8080/soap/servlet/rpcrouter"
/>
    </port>
    <port name="ClockIFPort2" binding="tns:ClockIFBinding">
      <soap:address
        location="http://diplom2.mine.nu:8080/soap/servlet/rpcrouter"
/>
    </port>
  </service>
</definitions>
```

C.3.2 Deployment descriptor

```
<!--
  $Id$
  Deployment descriptior used for deployment with Apache SOAP
  (http://xml.apache.org/soap/)
-->

<dd:service xmlns:dd="http://xml.apache.org/xml-soap/deployment"
            id="urn:ClockService">
  <dd:provider type="java"
               scope="Application"
               methods="getTime">

    <dd:java class="prototype.ws.service.Clock" static="false" />
  </dd:provider>
  <dd:faultListener>
    org.apache.soap.server.DOMFaultListener
  </dd:faultListener>
  <dd:mappings />
</dd:service>
```

C.3.3 Build and run scripts

C.3.3.1 Ant file

```
<?xml version="1.0"?>
<!-- $Id$ -->
<project name="Clock Web Service" default="build" basedir=".">

  <property file="build.properties" />
       <property environment="env" />
  <path id="classpath">
    <fileset dir="${dir.commonlib}">
      <include name="*.jar" />
    </fileset>
  </path>

  <target name="prep">
    <mkdir dir="${dir.classes}" />
    <mkdir dir="${dir.lib}" />
  </target>

  <target name="compile" depends="prep">
    <javac srcdir="${dir.src}" destdir="${dir.classes}">
      <include name="**/*.java" />
      <classpath refid="classpath" />
    </javac>
  </target>


  <!--
    This target deploys the ClockService as a Apache SOAP web
service.
  -->
  <target name="deploy-clock" depends="compile">
    <copy file="clock.wsdl" todir="${env.TOMCAT_HOME}/webapps/ROOT"
/>
    <copy todir="${env.TOMCAT_HOME}/webapps/soap/WEB-INF/classes/">
      <fileset dir="${dir.classes}">
        <include name="**/Clock*.class" />
        <exclude name="**/*Client*.class" />
      </fileset>
    </copy>

    <!-- deploy using admin client -->
    <java classname="org.apache.soap.server.ServiceManagerClient"
fork="yes">
      <classpath refid="classpath" />
      <arg line="http://diplom.mine.nu:8080/soap/servlet/rpcrouter"
/>
      <arg line="deploy" />
      <arg line="clock_dd.xml" />
    </java>
  </target>


  <!--
    Run the client using dynamic invocation with the WSIF
    environment
  -->
  <target name="run-dynamic">
    <java classname="clients.DynamicInvoker" fork="yes">
      <classpath refid="classpath" />
      <arg line="http://diplom.mine.nu:8080/clock.wsdl getTime" />
    </java>
  </target>


  <!--
    This target runs the clock client
  -->
  <target name="run-clock-client" depends="compile">
    <java classname="prototype.ws.client.ClockClient" fork="yes">
      <classpath refid="classpath" />
      <classpath path="${dir.classes}" />
      <arg line="http://diplom.mine.nu:8080/soap/servlet/rpcrouter"
/>
      <arg line="urn:ClockService" />
      <arg line="10" />
      <arg line="1000" />
    </java>
  </target>

  <!--
    UDDI test target
  -->
  <target name="run-uddi-client">
    <java classname="prototype.ws.uddiclient.UddiClockClient"
fork="yes">
      <classpath refid="classpath" />
      <classpath path="${dir.classes}" />
      <classpath path="." />
      <arg line="ClockService" />
      <arg line="10" />
      <arg line="1000" />
    </java>
  </target>

  <!--
    Simple clean target... removes created class files and jars
  -->
  <target name="clean">
    <delete dir="${dir.classes}" />
    <delete dir="${dir.lib}" />
  </target>
</project>
```

C.3.3.2 Ant properties

```
# build.properties
dir.src=${basedir}/src
dir.classes=${basedir}/classes
dir.lib=${basedir}/lib

# libraries
dir.commonlib=${basedir}/common/lib
```

C.3.3.3 Properties file for UDDI enabled client

```
## uddi.properties
queryManagerURL=http://www-3.ibm.com/services/uddi/v2beta/inquiryapi

## service description
ClockService.org.name=Diploma RF & PW
```

C.4 Source code

C.4.1 Client

C.4.1.1 ClockClient.java

```
package prototype.ws.client;

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;
```

```
/**
 * Test client implementation for invoking the <code>
 * Clock</code> Web Service.
 *
 * @see prototype.ws.client.SOAPProxy SOAP call impl.
 * @see prototype.ws.service.Clock The web service
 */
public class ClockClient {

  /**
   * Ivokes the call to the web service
   */
  private ClockInterfaceProxy proxy;

  /**
   * Simple constructor; instaniate the proxy
   */
  public ClockClient(URL endpoint, String uri) {
    proxy = new ClockInterfaceProxy(endpoint, uri);
  }

  /**
   * Invoke the method
   */
  public String makeCall() throws Exception {
    return proxy.getTime();
  }


  /**
   * main method
   */
  public static void main(String[] args) throws Exception {

    URL url;
    String serviceName;
    int n;
    int delay;

    // parse command line
    System.out.println("parsing command line paramters... found " +
args.length + " parameters");
    if(args.length >= 2) {

      url = new URL(args[0]);
      serviceName = new String(args[1]);

      try {
        n = Integer.valueOf(args[2]).intValue();
      } catch (Exception e) {
        n = 1;
      }

      try {
        delay = Integer.valueOf(args[3]).intValue();
      } catch (Exception e) {
        delay = 1000;
      }

      System.out.println("Using endpoint: '" + url + "'");
      System.out.println("Invoking service: " + serviceName + " " + n
+ " time(s) with " + delay +"ms delay");
      System.out.println();

      // invoke the web service
      ClockClient app = new ClockClient(url, serviceName);
      while(n-- > 0) {

        System.out.println("\tTime is: " + app.makeCall());
        Thread.sleep(delay);
      }
    } else {

      System.out.println("Usage:");
      System.out.println("\tprototype.ws.client.ClockClient
<endpoint> <service> <n> <delay>");
      System.exit(1);
    }
  }
}
```

C.4.1.2 ClockClientInterfaceProxy.java

```
// $Id$
package prototype.ws.client;

import prototype.ws.service.ClockInterface;
import java.net.URL;

/**
 * Custom SOAP proxy class supporting the ClockInterface
 */
public class ClockInterfaceProxy extends SOAPProxy implements
ClockInterface {

  /**
   * Simple constructor just invoking the Proxy
   *
   * @param endpoint SOAP rpc endpoint
   * @param uri Service uri
   */
  public ClockInterfaceProxy(URL endpoint, String uri) {
    super(endpoint, uri);
  }

  /**
   * Redirecting the getTime method to the
   * proxy
   */
  public String getTime() throws Exception {

    Object result = soapCall("getTime", null);

    return (String) result;
  }
}
```

C.4.1.3 SOAPProxy.java

```
// $Id$
package prototype.ws.client;

import prototype.ws.service.ClockInterface;
import java.net.URL;

/**
 * Custom SOAP proxy class supporting the ClockInterface
 */
public class ClockInterfaceProxy extends SOAPProxy implements
ClockInterface {

  /**
   * Simple constructor just invoking the Proxy
   *
   * @param endpoint SOAP rpc endpoint
   * @param uri Service uri
   */
  public ClockInterfaceProxy(URL endpoint, String uri) {
    super(endpoint, uri);
  }

  /**
   * Redirecting the getTime method to the
   * proxy
   */
  public String getTime() throws Exception {

    Object result = soapCall("getTime", null);

    return (String) result;
  }
}
```

C.4.2 UDDI enabled client

C.4.2.1 UddiClockClient.java

```
package prototype.ws.uddiclient;

import java.io.*;
import java.net.*;
import java.util.*;
import org.apache.soap.*;
import org.apache.soap.rpc.*;


/**
 * Test client implementation for invoking the <code>
 * Clock</code> Web Service.
 *
 * @see prototype.ws.client.SOAPProxy SOAP call impl.
 * @see prototype.ws.service.Clock The web service
 */
public class UddiClockClient {

  /**
   * Ivokes the call to the web service
   */
  private UddiClockInterfaceProxy proxy;

  /**
   * Simple constructor; instaniate the proxy
   */
  public UddiClockClient(String uri) {
    proxy = new UddiClockInterfaceProxy(uri);
  }

  /**
   * Invoke the method
   */
  public String makeCall() throws Exception {
    return proxy.getTime();
  }


  /**
   * main method
   */
  public static void main(String[] args) throws Exception {

    URL url;
    String serviceName;
    int n;
    int delay;

    // parse command line
    System.out.println("parsing command line paramters... found " +
args.length + " parameters");
    if(args.length >= 1) {


      serviceName = new String(args[0]);

      try {
        n = Integer.valueOf(args[1]).intValue();
      } catch (Exception e) {
        n = 1;
      }

      try {
        delay = Integer.valueOf(args[2]).intValue();
      } catch (Exception e) {
        delay = 1000;
      }
      System.out.println("Invoking service: " + serviceName + " " + n
+ " time(s) with " + delay +"ms delay");
      System.out.println();

      // invoke the web service
      UddiClockClient app = new UddiClockClient(serviceName);
      while(n-- > 0) {

        System.out.println("\tTime is: " + app.makeCall());
        Thread.sleep(delay);
      }
    } else {

      System.out.println("Usage:");
      System.out.println("\tjava prototype.ws.client.ClockClient
<service> <n> <delay>");
      System.exit(1);
    }
  }
}
```

C.4.2.2 UDDIClockInterfaceProxy.java

```
// $Id$
package prototype.ws.uddiclient;

import prototype.ws.service.ClockInterface;
import java.net.URL;

/**
 * Custom SOAP proxy class supporting the ClockInterface
 */
public class UddiClockInterfaceProxy extends
    UddiSoapProxy implements ClockInterface {

  /**
   * Simple constructor just invoking the Proxy
   *
   * @param uri Service uri
   */
  public UddiClockInterfaceProxy(String uri) {
    super(uri);
  }

  /**
   * Redirecting the getTime method to the
   * proxy
   */
  public String getTime() throws Exception {

    Object result = soapCall("getTime", null);

    return (String) result;
  }
}
```

C.4.2.3 UDDISoapProxy.java

```
// $Id$
package prototype.ws.uddiclient;

import org.uddi4j.*;
import org.uddi4j.response.*;
import org.uddi4j.client.*;
import org.uddi4j.datatype.*;
import org.uddi4j.datatype.binding.*;
import org.uddi4j.datatype.business.*;
import org.uddi4j.datatype.service.*;
import org.uddi4j.datatype.tmodel.*;
import org.uddi4j.util.*;
import org.uddi4j.transport.*;

import org.apache.soap.Constants;
import org.apache.soap.Fault;
import org.apache.soap.rpc.Call;
import org.apache.soap.rpc.Response;
import org.apache.soap.rpc.Parameter;

import java.util.Properties;
import java.util.Vector;
import java.util.Enumeration;
import java.io.FileInputStream;
import java.net.URL;
import java.net.MalformedURLException;

public class UddiSoapProxy {

  /**
   * Loads uddi properties from file
   */
  static {

    Properties props = new Properties(System.getProperties());
    try {
      FileInputStream in = new FileInputStream("uddi.properties");
      props.load(in);
      in.close();
      System.setProperties(props);
    } catch (Exception e) {
      System.out.println("Failed to load properties file");
    }
  }

  /**
   * Service identification
   */
  private String uri;

  /**
   * Service url
   */
  private URL endpoint;

  /**
   * @param uri Used for service identification
   */
  public UddiSoapProxy(String uri) {

    this.uri = uri;

    try {
      findService();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Generic soap call
   *
   * @param method The name of the method to invoke
   * @param params Vector with parameters to the call
   * @return Result of method call
   */
  public Object soapCall(String method, Vector params) throws
Exception {

    if(endpoint == null)
      throw new RuntimeException("endpoint not set, can't invoke
method");
    if(uri == null)
  throw new RuntimeException("uri not set, can't invoke method");
    Call call = new Call();
    call.setTargetObjectURI("urn:"+uri);
    call.setMethodName(method);
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);
    call.setParams(params);

    Response resp = call.invoke(endpoint, "");
    if(resp.generatedFault()) {
      Fault fault = resp.getFault();
      System.out.println("Fault code: " + fault.getFaultCode());
      System.out.println("Fault string: " + fault.getFaultString());
      return null;
    } else {
      Parameter result = resp.getReturnValue();
      return result.getValue();
    }
  }

  /**
   */
  private void findService() throws
      MalformedURLException, UDDIException, TransportException {

    UDDIProxy proxy;
    BusinessEntity businessEntity;
    String queryManagerURL = System.getProperty("queryManagerURL");

    System.out.println("finding service, using uddi service at\n\t" +
      queryManagerURL);

    proxy = new UDDIProxy(new URL(queryManagerURL), null);

    // find business
    // Create a vector of Name objects, which is used for the inquiry
    Vector names = new Vector();
    names.addElement(new Name(System.getProperty(uri+".org.name")));

    // Makes the call on the proxy
    BusinessList list =
      proxy.find_business(names, null, null, null, null, null, 0);
    BusinessInfos infos = list.getBusinessInfos();

    // If the returned vector is empty, no business could be found
    if(infos.getBusinessInfoVector().size() == 0) {
      throw new RuntimeException("No business found");
    }

    // Obtain the unique identifier for this entry in the registry
    BusinessInfo info =
      (BusinessInfo) infos.getBusinessInfoVector().elementAt(0);
    String businessKey = info.getBusinessKey();

    // Uses the unique identifier to obtain the full record
    businessEntity =
      (BusinessEntity) proxy.get_businessDetail(businessKey).
        getBusinessEntityVector().elementAt(0);

    // find service
    BusinessServices bs = businessEntity.getBusinessServices();
    if(bs != null) {

      boolean found = false;
      Vector services = bs.getBusinessServiceVector();
      Enumeration enum = services.elements();
      while(enum.hasMoreElements() && !found) {

        BusinessService service = (BusinessService)
enum.nextElement();
        System.out.println("Service: " +
service.getDefaultNameString());
        if(service.getDefaultNameString().equals(uri)) {
          found = true;
          System.out.println("found service: " +
            service.getDefaultNameString());

          BindingTemplates bTemplates =
service.getBindingTemplates();
          Vector bTempVector = bTemplates.getBindingTemplateVector();
          if(bTempVector.size() > 0) {
            BindingTemplate bTemp =
              (BindingTemplate) bTempVector.elementAt(0);
            AccessPoint ap = bTemp.getAccessPoint();
            endpoint = new URL(ap.getText());
            System.out.println("Using endpoint: " + endpoint);
          } else
            throw new RuntimeException("No endpoint registered");
        }
      }
    }
  }
}
```

C.4.3 Service

C.4.3.1 ClockInterface.java

```
// $Id$
package prototype.ws.service;

/**
 * Interface used for a simple method specification
 * for the web service
 */
public interface ClockInterface {

  /**
   * Gets current time
   * @return the time plus hostname
   */
  public String getTime() throws Exception;
}
```

C.4.3.2 ClockService.java

```
// $Id$
package prototype.ws.service;

import java.util.Date;
import java.net.InetAddress;

/**
 * Simple implementation of the Clock Web Service
 */
public class Clock implements ClockInterface {

  /**
   * @see prototype.ws.service.ClockInterface#getTime
   */
  public String getTime() {
    String hostname = null;

    try {
      InetAddress addr = InetAddress.getLocalHost();
      hostname = addr.getHostName();
    } catch (Exception ex) {
      ex.printStackTrace(System.err);
    }

    Date now = new Date(System.currentTimeMillis());
    return now.toString() + " " + hostname;
  }
```

Appendix D – JXTA development environment

D.1 Software

D.1.1 Project JXTA starting kit

The starting kit for JXTA is free, and can be downloaded from the URL below. The installation is made simple and is in many cases almost 100% independent of user interaction.

http://download.jxta.org/easyinstall/install.html

D.2 Building test application

D.2.1 Creating the service

The first that had to be made was a module class advertisement associated with the service.
The Module class advertisement is a small advertisement that only advertises the existence of service. In order to access the service, a peer will have to discover the associated module spec advertisement.
Then we had to create a pipe advertisement for the Service. The client must use the same pipe advertisement to talk to the service. When the client discovers the module advertisement it will extract the pipe advertisement to create its pipe.
To make sure that the service always creates the same service we are reading the pipe advertisement from a file. The file is an XML document containing an id and a service name associated with it.
When this is done we can publish into the local cache the netpeergroup.
The next step is to make an output pipe which is used to send data to the clients when they are looking for time data.

D.2.2 Creating the client

The first that is done in the client is to start searching for the service; this search can be done locally and remote. The local search is based on earlier discovery that has been done before and then stored to the disk. If this search is not a success the next step is a remote search.
A remote discovery is asynchronous, and we do not know how long it is going to take before we get an answer, if we get an answer at all. Since we have no guarantee that we will find a service this process will loop until success.
When the service is discovered the pipe advertisement must be collected. The pipe advertisement is used to create an input pipe which the service can collect the information that the service is sending out.

D.3 Script

The deployment of a JXTA network is easy and does not require other components as a lookup server or similar.

D.3.1 Starting the service

```
#!/bin/sh

echo Staring Service......

# everything below should work with few changes

CLASSFILES=$JXTA_HOME/lib/jxta.jar:$JXTA_HOME/lib/jxtacms.jar:$JXTA_H
OME/lib/jxtasecurity.jar:$JXTA_HOME/lib/instantp2p.jar\
:$JXTA_HOME/lib/log4j.jar:$JXTA_HOME/lib/beepcore.jar:$JXTA_HOME/lib/
jxtashell.jar:$JXTA_HOME/lib/cryptix-asn1.jar\
:$JXTA_HOME/lib/cryptix32.jar\:$JXTA_HOME/lib/minimalBC.jar:$JXTA_HOM
E/lib/jxtaptls.jar
```

D.3.2 Starting the client

```
#!/bin/sh

echo Staring Service......

# everything below should work with few changes

CLASSFILES=$JXTA_HOME/lib/jxta.jar:$JXTA_HOME/lib/jxtacms.jar:$JXTA_H
OME/lib/jxtasecurity.jar:$JXTA_HOME/lib/instantp2p.jar\
:$JXTA_HOME/lib/log4j.jar:$JXTA_HOME/lib/beepcore.jar:$JXTA_HOME/lib/
jxtashell.jar:$JXTA_HOME/lib/cryptix-asn1.jar\
:$JXTA_HOME/lib/cryptix32.jar\:$JXTA_HOME/lib/minimalBC.jar:$JXTA_HOM
E/lib/jxtaptls.jar



java -classpath $CLASSFILES:. prototypejxta.StartClient
```

D.3.3 Pipe service advertisement

```
<?xml version="1.0"?>
<!DOCTYPE jxta:PipeAdvertisement>
<jxta:PipeAdvertisement xmlns:jxta="http://jxta.org">
        <Id>
                urn:jxta:uuid-
9CCCDF5AD8154D3D87A391210404E59BE4B888209A2241A4A162A10916074A9504
        </Id>
        <Type>
                JxtaUnicast
        </Type>
        <Name>
                JXTA-TIMESERVICE
        </Name>
</jxta:PipeAdvertisement>
```

D.4 Source code

D.4.1 Client

```
import java.io.IOException;
import java.io.StringWriter;
import java.util.Enumeration;
import net.jxta.document.Advertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.StructuredTextDocument;
import net.jxta.document.MimeMediaType;
import net.jxta.document.TextElement;
import net.jxta.endpoint.Message;
import net.jxta.pipe.PipeService;

import net.jxta.protocol.ModuleSpecAdvertisement;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.discovery.DiscoveryService;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupFactory;
import net.jxta.exception.PeerGroupException;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.PipeMsgEvent;
import net.jxta.pipe.PipeMsgListener;

public class clientStart implements PipeMsgListener, Runnable{
    static PeerGroup netPeerGroup = null;
    private DiscoveryService discoSvc;
    private PipeService pipeSvc;
    private InputPipe myPipe; // input pipe to connect the service
    private Message msg; // message to be sent
    private final static String SERVICE = "JXTASPEC:JXTA-TIMESERVICE"; // service name
    private final static String TAG = "DataTag"; // tag in message

    public clientStart() {
          try {
              // create, and Start the default jxta NetPeerGroup
              netPeerGroup = PeerGroupFactory.newNetPeerGroup();
          } catch (PeerGroupException e) {
              // could not instantiate the group, print the stack and exit
              System.out.println("fatal error : group creation failure");
              e.printStackTrace();
              System.exit(1);
          }//end try/catch
          // get the discovery, and pipe service
          System.out.println("Getting DiscoveryService");
          discoSvc  = netPeerGroup.getDiscoveryService();
          System.out.println("Getting PipeService");
          pipeSvc   = netPeerGroup.getPipeService();
          startClient();
    }//end constructor

    private void startClient() {
        // Let’s initialize the client
        System.out.println("Start the Client");
        // Let’s try to locate the service advertisement SERVICE
        // we will loop until we find it!
        System.out.println("searching for the " + SERVICE + " Service advertisement");
        Enumeration enum = null;
        while (true) {
            try {
                // let’s look first in our local cache to see
                // if we have it! We try to discover an advertisement
                // which has the (Name, JXTASPEC:JXTA-TIMESERVICE) tag value
                //
                enum = discoSvc.getLocalAdvertisements(DiscoveryService.ADV, "Name",
SERVICE);
                // Found it! Stop searching and go send a message.
                if ((enum != null) && enum.hasMoreElements()) break;
                // We could not find anything in our local cache, so let’s send a
                // remote discovery request searching for the service advertisement
                discoSvc.getRemoteAdvertisements(null, DiscoveryService.ADV, "Name",
SERVICE,1, null);
                // The discovery is asynchronous as we do not know
                // how long is going to take
                  try { // sleep as much as we want. Yes we
                      // could implement asynchronous listener pipe...
                      Thread.sleep(2000);
                  } catch (Exception e){}
            } catch (IOException e){
              /* found nothing! move on*/
            }//end try/catch
            System.out.print(".");
        }//end while
        System.out.println("We found the service advertisement:");
        // Ok get the service advertisement as a ModuleSpecAdvertisement
        ModuleSpecAdvertisement mdsadv = (ModuleSpecAdvertisement)enum.nextElement();
        try {
            // let’s print the advertisement as a plain text document
            StructuredTextDocument doc = (StructuredTextDocument)
            mdsadv.getDocument(new MimeMediaType("text/plain"));
            StringWriter out = new StringWriter();
            doc.sendToWriter(out);
            System.out.println(out.toString());
            out.close();
            // Get the pipe advertisement -- need it to talk to the service
            PipeAdvertisement pipeadv = mdsadv.getPipeAdvertisement();
            if (pipeadv == null){
                System.out.println("Error -- Null pipe advertisement!");
                System.exit(1);
            }
            // create the input pipe endpoint to connect
            // to the server, try 3 times to bind the pipe listening endpoint to
            // the endpoint pipe of the service
            myPipe = null;
            for (int i=0; i<3; i++) {
                System.out.println("Trying to bind to pipe...");
                try {
                    myPipe = pipeSvc.createInputPipe(pipeadv,this);
                    break;
                } catch (java.io.IOException e) {}//end try/catch
            }//end for
            if (myPipe == null) {
                System.out.println("Error resolving pipe endpoint");
                System.exit(1);
            }//end if

        } catch (Exception ex) {
            ex.printStackTrace();
            System.out.println("Client: Error receiving message from the service");
        }//end try/catch
    }//end startClient

    public void pipeMsgEvent(PipeMsgEvent evt){
        Message msg=null;
        try {
            msg = evt.getMessage();
            if (msg == null)       return;
        }catch (Exception e) {
            e.printStackTrace();
            return;
        }//end try/catch
        // Get message
        String newMessage = msg.getString(TAG);
        if (newMessage == null) System.out.println("null msg received");
        else                    System.out.println("Received message: " + newMessage);
    }//end pipeMsgEvent


    public void run(){
       while(true){
          try{
              Thread.sleep(10*10*1000);
          }catch(InterruptedException ex){}
        }//end true
    }//end run

    public static void main(String[] args) {
      clientStart clientStart1 = new clientStart();
      Thread thread = new Thread(clientStart1);
      thread.start();

    }//end main
}//end class
```

D.4.2 Service

```
import java.io.*;
import java.net.URL;
import net.jxta.document.*;
import net.jxta.peergroup.*;
import net.jxta.protocol.*;

import net.jxta.discovery.DiscoveryService;
import net.jxta.exception.PeerGroupException;

import net.jxta.endpoint.Message;
import net.jxta.id.IDFactory;
import net.jxta.platform.ModuleClassID;

import net.jxta.pipe.PipeService;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.OutputPipe;

import net.jxta.pipe.OutputPipeListener;
import net.jxta.pipe.OutputPipeEvent;
import java.util.Date;

public class ServiceStart implements Runnable{
    static  PeerGroup group = null;
    private DiscoveryService  discoSvc;
    private PipeService       pipeSvc;
    private OutputPipe        myPipe;   // input pipe for the service
    private Message           msg;      // message received on input pipe
    private final static String SERVICE   = "JXTASPEC:JXTA-TIMESERVICE"; // service name
    private final static String TAG       = "DataTag"; // tag in message
    private final static String FILENAME  = "pipeserver.adv"; // file containing pipe
advert.

    public ServiceStart() {
          try {
              // create, and Start the default jxta NetPeerGroup
              group = PeerGroupFactory.newNetPeerGroup();
          } catch (PeerGroupException e) {
              // could not instanciate the group, print the stack and exit
              System.out.println("fatal error : group creation failure");
              e.printStackTrace();
              System.exit(1);
          }//end try/catch

          // get the discovery, and pipe service
          System.out.println("Getting DiscoveryService");
          discoSvc = group.getDiscoveryService();
          System.out.println("Getting PipeService");
          pipeSvc = group.getPipeService();

          startServiceDaemon();
    }//end constructor

    public void startServiceDaemon(){
          System.out.println("Start the Server daemon");
          try {
              // Create the Module class advertisement associated with the service
              // We build the module class advertisement using the Advertisement
              // Factory class by passing it the type of the advertisement we
              // want to construct. The Module class advertisement is a
              // a very small advertisement that only advertises the existence
              // of service. In order to access the service, a peer will
              // have to discover the associated module spec advertisement.
              ModuleClassAdvertisement mcadv =
(ModuleClassAdvertisement)AdvertisementFactory.newAdvertisement(ModuleClassAdvertisement.get
AdvertisementType());
              mcadv.setName("JXTAMOD:JXTA-TIMESERVICE");
              mcadv.setDescription("Time advertisement service");
              ModuleClassID mcID = IDFactory.newModuleClassID();
              mcadv.setModuleClassID(mcID);
              // Ok the Module Class advertisement was created, just publish
              // it in my local cache and to my peergroup. This
              // is the NetPeerGroup
              discoSvc.publish(mcadv, DiscoveryService.ADV);
              discoSvc.remotePublish(mcadv, DiscoveryService.ADV);
              // Create the Module Spec advertisement associated with the service
              // We build the module Spec Advertisement using the advertisement
              // Factory class by passing in the type of the advertisement we
              // want to construct. The Module Spec advertisement will contain
              // all the information necessary for a client to contact the service
              // for instance it will contain a pipe advertisement to
              // be used to contact the service
              ModuleSpecAdvertisement mdadv =
(ModuleSpecAdvertisement)AdvertisementFactory.newAdvertisement(ModuleSpecAdvertisement.getAd
vertisementType());

              // Setup some of the information field about the servive. In this
              // example, we just set the name, provider and version and a pipe
              // advertisement. The module creates an input pipes to listen
              // on this pipe endpoint.
              mdadv.setName(SERVICE);
              mdadv.setVersion("Version 1.0");
              mdadv.setCreator("sun.com");
              mdadv.setModuleSpecID(IDFactory.newModuleSpecID(mcID));
              mdadv.setSpecURI("http://www.jxta.org/Ex1");
              // Create a pipe advertisement for the Service. The client MUST use
              // the same pipe advertisement to talk to the server. When the client
              // discovers the module advertisement it will extract the pipe
              // advertisement to create its pipe. So, we are reading the pipe
              // advertisement from a default config file to ensure that the
              // service will always advertise the same pipefs
              System.out.println("Reading in file " + FILENAME);
              PipeAdvertisement pipeadv = null;
              try {
                  FileInputStream is = new FileInputStream(FILENAME);
                  pipeadv = (PipeAdvertisement)AdvertisementFactory.newAdvertisement(new
MimeMediaType("text/xml"), is);
                  is.close();
              } catch (java.io.IOException e) {
                  System.out.println("failed to read/parse pipe advertisement");
                  e.printStackTrace();
                  System.exit(-1);
              }//end try/catch

              // add the pipe advertisement to the ModuleSpecAdvertisement
              mdadv.setPipeAdvertisement(pipeadv);
              // display the advertisement as a plain text document.
              System.out.println("Created service advertisement:");
              StructuredTextDocument doc = (StructuredTextDocument)
              mdadv.getDocument(new MimeMediaType("text/plain"));
              StringWriter out = new StringWriter();
              doc.sendToWriter(out);
              System.out.println(out.toString());
              out.close();
              // Ok the Module advertisement was created, just publish
              // it in my local cache and into the NetPeerGroup.
              discoSvc.publish(mdadv, DiscoveryService.ADV);
              discoSvc.remotePublish(mdadv, DiscoveryService.ADV);
              // We are now ready to start the service --
              // create the output pipe endpoint clients will
              // use to connect to the service

              for (int i=0; i<10; i++) {
                  System.out.println("Trying to bind to pipe...");
                  try {
                      myPipe = pipeSvc.createOutputPipe(pipeadv, 10000);
                      break;
                  } catch (java.io.IOException e) {
                  // go try again;
                  }//end try/catch
              }//end for
              if (myPipe == null) {
                  System.out.println("Error resolving pipe endpoint");
                  System.exit(1);
              }//end if

          } catch (Exception ex) {
              ex.printStackTrace();
              System.out.println("Server: Error publishing the module");
          }//end try/catch
    }//end startServiceDeamon

    public void writeMessage(){
            if (myPipe == null) {
                System.out.println("Error resolving pipe endpoint");
            }//end if
            else{
                Date now      = new Date(System.currentTimeMillis());
                String myMsg  = "Date from peer " + group.getPeerName() + " Date: " +
now.toString();

                msg = pipeSvc.createMessage();
                msg.setString(TAG, myMsg);
                // send the message to the client pipe
                try{
                    myPipe.send(msg);
                }catch(Exception e){
                    System.out.println("Error when sending message: " + e);
                }//End try/catch
                System.out.println("message \"" + myMsg + "\" sent to the Client");
            }//end else
    }//end writeMessage

    public void run(){
        while(true){
          try{
              writeMessage();
              Thread.sleep(1*1*1000);
          }catch(InterruptedException ex){}
        }//end true
    }//end run

    public static void main(String[] args) {
      ServiceStart serviceStart1 = new ServiceStart();
      Thread thread = new Thread(serviceStart1);
      thread.start();
    }//end main

}//end class
```