# A Jini-based prototype metacomputing framework

Zoltan Juhasz[1 2]

[1] University of Veszprem, Department of Information Systems, Veszprem, Hungary, juhasz@irt.vein.hu
[2] University of Exeter, Department of Computer Science, United Kingdom

**Abstract.** This paper investigates the possible use of Jini technology for building Java-based metacomputing environments. It gives an overview of Jini and highlights those features that can be used effectively for metacompting. Based on this evaluation the paper then presents the structure of a prototype Jini metacomputing system. The overall working mechanism and some details of the implementation are described. It is shown that Jini is a suitable technology that deserves the interest of the metacomputing research community.

## 1 Introduction

The past few years generated tremendous interest in metacomputing. Several research groups world wide are working on how to provide seamless access to the vast number of computers connected by the Internet, to create better and more useful computing services, and to improve the performance of metacomputing systems. Different approaches, languages and paradigms have been used.

Jini technology [5]– developed by Sun Microsystems – is a new Java-based technology to create autonomous, ad hoc network of digital devices. It is a service-based system featuring automatic discovery of the network and its available services, fault tolerance, transactions and distributed event mechanism.

Although there are many Java-based metacomputing projects in progress, at the time of writing, the author is not aware of any metacomputing project building solely on Jini. The purpose of this paper is to report on our ongoing work whose aim is to investigate the suitability of Jini as a metacomputing technology, and outline the structure and underlying working mechnism of one possible jini metacomputing system.

The structure of this paper is the following. Section 2 overviews the basics of metacomputing environments with particular attention to the most important services. Section 3 gives a brief introduction to Jini technology. Section 4 evaulates the potential of Jini technology for metacomputing systems and outlines the structure of our prototype system. Section 5 gives details of our prototype implentation, while Section 6 gives a brief look at the most well-known metacomputing projects. The paper ends with conclusions and a discussion of future research issues.

## 2 Metacomputing systems

The term *metacomputing system* now commonly refers to systems composed of potentially thousands of personal computers, workstations or supercomputers to solve compute-intensive problems or provide better solutions in geographically distributed work environments.

Compared to parallel systems, metacomputing environments open up a set of new problems and issues that must be dealt with. Baker and Fox [8] identify that a successful

"metacomputing environment:

- does not interfere with existing site administration or autonomy
- does not compromise existing security of users or remote sites
- does not need to replace existing operating systems, network protocols or services
- allows remote sites to join or leave the environment whenever they choose
- does not mandate the programming paradigms, languages, tools, or libraries that a user wants
- provides a reliable and fault tolerant infrastructure with no single point of failure
- provides support for heterogeneous components
- uses standards, and existing technologies, and is able to interact with legacy applications
- provides appropriate synchronization and component program linkage."

Consequently, such successful systems must cope with partial failure, geographical distance, large network latencies, different language environments, hardware platforms and operating sytems, and must provide solutions for security problems, fault tolerance and portability.

### 2.1 Minimal service set

Most metacomputing systems are service based. Services provide distinct groups of functionality. Services are generally used for resource management (allocating/scheduling tasks, load-balancing, process management), communication (eg mesage passing), security and authentication, system status (monitoring the availablitiy and system load of components), and accessing remote data. These are nicely represented by the service set of the Globus [1] project, such as the Resource management, Communication, Security, Information, Health and Status, Remote Data Access, and Executable Management services.

## 3 Jini and JavaSpaces

Jini Technology [5] is one of the latest developments of Sun's Java product line. Launched in January 1999, Jini has been invented to simplify the way we use networked equipment and create distributed systems. Its goal is to create impromptu communities or federations of networked devices. A federation is set of software components and devices creating a distributed system. Devices are present and active in the federation when necessary; they can join and leave the community when they wish.

Jini is based on the Java software infrastructure. It relies heavily on Java's Remote Method Invocation (RMI) mechanism that facilitates the passing of objects between Java Virtual Machines. This feature allows Jini devices to move executable program code around the network and as such, create networks of digital devices without planning, installation and human intervention.

As Jini is a service-centred technology, a participant of the federation is either a service provider offering a service for the community or a client that has become a participant in order to find and use a suitable service. A service is an entity (a piece of computation, a hardware device or software) that other programs, services or users, can use. Jini devices acting as services carry their own software and – once connected to the network – announce themselves and their services to the other members of the federation. Clients then can locate and use these services when they need them. It is a dynamic, demand-driven environment. Clients join the network when they need a service and services are only accessed and used when needed.

The central element of a Jini federation is the service registrar or lookup service. The role of the registrar is to keep a record of all the available services. This role determines the operation of the Jini systems, which follows a simple pattern.

Devices offering services to the network first must find lookup services on the network in order to register their services. This is carried out using a discovery mechanism. The service then registers with the located registrar in a join process. With this step performed, the object representing the service is uploaded to the registrar and the service becomes available to interested clients.

Clients also have to contact the available registrars and search for the required or suitable set of services. Finding a service, the lookup process, in the registrar is based on template matching. Once the appropriate service is located, its code is loaded into the client for execution. Depending on its functionality, a service can run entirely in the clients virtual machine or implement only a proxy to the real service running in the service provider's VM.

This pattern keeps repeating during the existence of the federation. New services, as they join the network, register with the lookup service(s). Similarly, new clients search the lookup service(s) to find services they need then load and execute the service.

### 3.1 Important properties of Jini

Jini has some important properties that make it an appealing candidate for creating distributed systems. We list here only the most important ones.

**Code mobility.** The ability to move code around the network is in the heart of Jini. This enables us to allocate and migrate processes transparently and efficiently.

**Security and authentication.** Jini uses all the security features of Java to avoid the dangers of moving potentially intruder code into a system.

**Leasing.** Jini is designed for operation in an unreliable environment. Participation in the federation is granted in a time-based manner. If the period of the grant for use (lease) is not renewed before its expiration, the grant will be withdrawn at the end of the period. If a lease is not renewed, the participant is considered non-operational and such a service is removed from the system. The leasing mechanism provides graceful degradation in the system.

**Distributed events.** Components of a Jini system can notify each other when some change in their state occurs. Participants can register their interest in such state changes and they will be automatically notified when this event occurs.

**Transaction.** Jini can provide execution in a transactional manner. A transaction will only succeed if all operations making up the transaction occur. Transactions provide a secure way of executing distributed operations.

**Minimal system requirement.** Jini systems do not require any operating system to run. They can work without and coexist with any operating system, as long as the system is capable of running the Java virtual machine.

### 3.2 JavaSpaces

One particular service Sun have implemented and provide with the Jini software distribution is the JavaSpaces [**?**] service. This service is an improved Java-based implementation of the Linda coordination language [11]. Relying on the Java RMI mechanism, Jini participants can pass around objects with great simplicity through the "javaspace" using only write, read and take operations. Participants can also notify each other about changes in the space with the notify operation.

## 4 A minimal metacomputing system

This sections looks at through what features Jini technology can support the creation of a metacomputing environment, and as a result of this analysis, outlines the structure of one such possible system.

From the requirement list of Baker and Fox [8], a pure Java/Jini system will certainly satisfy the administration, security and operating system requirements. In such a system, remote sites will be able to join or leave whenever they choose – this is a central property of Jini. Also, it can create a reliable system, and it uses existing standards. It will require, however, programs to be written in Java, therefore it mandates the type of language to be used, although by using the Java Native Interface, it is possible to integrate code written in other languages including legacy applications.

In addition, the following key middleware features are directly supported in Jini. By using groups, it is possible to create a hierarchical system of resources. Communication is supported by the underlying Java system including TCP/IP, UDP unicast and multicast communication, and by RMI. Jini provides directory, naming and registration services through its registrar. Security and authorisation is also provided by Java. Fault tolerance is achieved by the by lease mechanism. Services and clients can easily restart and join the Jini federation after a crash. Registrars will also automatically restart themselves and all registered services after a registrar crash. Jini provides the use of transactions that can be used to create secure applications and provide Quality of Service.

There is no direct support, however, for distributed time and clock services, for distributed file system or for resource management and scheduling. These features must be added additionally by developing suitable services. It can be seen that most required features are provided by Jini and this makes it a very appealing choice for the underlying technology. Besides the above features, its object-oriented nature and platform independence are additional benefits.

In the rest of this section, we look at how these services can be created and how they can be combined to form a simple metacomputing environment.

### 4.1 System structure

Our current plan is to create minimal system (a testbed), which we can use to experiment with various features of Jini and consequently, evaluate its general suitability for building metacomputing systems. We expect the prototype system to be able to *(i)* monitor the current state of the computing resources, *(ii)* identify which resources it should use for parallel tasks and *(iii)* map and execute the tasks on the selected resources.

We use the following three participants, Client, Broker and Host. The *Client* represents the parallel application to be run on the system. A client first has to find available brokers then request the execution of the program from one or more. A *Broker* is responsible for two main actions. It must be able to monitor resources and select a subset of them for executing the client's tasks (based on requirements received from the client with the execution request). Monitoring is performed through the *Host* services that provide crucial state information to the brokers and are able to execute tasks allocated to them by the broker service.

We expect each cluster or site to have at least one Jini registrar available. Since the Broker and Host are Jini services, they will have to register with the registrar. The Client will only use other services, therefore it is a Jini client. It is important to note that Broker, at the same time, is also acting as a Jini client, since it is a client when it uses the Host service.

### 4.2 The operation of the system

The system is depicted in Fig 1. The main components are the hosts, brokers and clients as described above. Thus, the system is a collection of services and clients that are registered in a number of registrars.

Each parallel host or cluster runs a host service that implements a generic compute engine and contains methods that manage processes. The service also has static attributes describing the type of computer it represents. The service is lease based, and the attributes are updated on each lease renewal.

Brokers discover potential resources by finding Host services in the registrar and after dowloading they extract information from the host services. In the simplest form of implementation, the Host service will run entirely in the broker's virtual machine. Note that it is possible to create host services that provide dynamic, time-varying information (e.g. change in load level). In this case, the

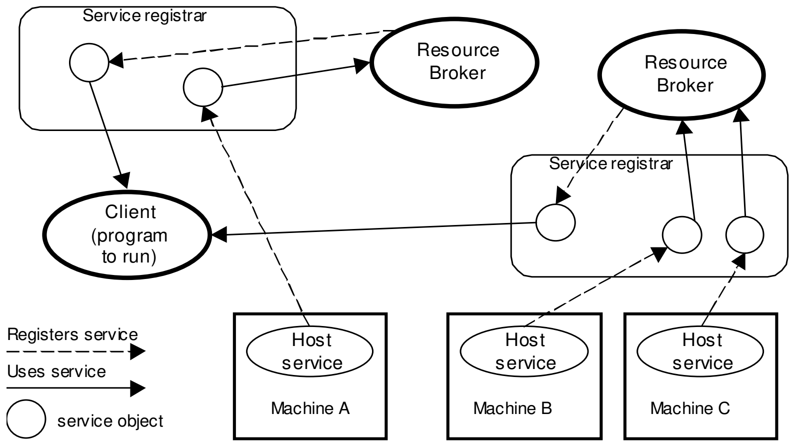


**Fig. 1.** Movement of service and application objects during the operation of the system

## 5 Implementation of the prototype system

After the general operation of the system, we briefly describe the three main participants. The client as well as the broker and host services are implemented by Java objects. To understand the details, we look at first the process of registering a Jini service.

The code sample below illustrates the necessary steps. We describe the operation starting with line 5 working backwards. Each service registers with the registrar through the registrar object's `register()` method (Line 5). This has two input parameters, a `ServiceItem` object describing the service and a lease period given in milliseconds. A `ServiceItem` object comprises of the following parts

1. the service ID, which is assigned by the registrar after the successful registration,
2. the actual service object representing the service, and
3. an attribute set describing the service for clients during the template-matching based service lookup operation. The ServiceItem object is created in line 4 and lines 1-3 show the creation of the attribute set.

```
1. Entry[] attributeSet = new Entry[2];
2. attributeSet[0] = new OneAttribute("field_1",
                     " field_2", " field_3");
3. attributeSet[1] = new AnotherAttribute("field_1",
                     " field_2");
4. ServiceItem hostServiceItem = new ServiceItem(null,
                     new HostService(), attributeSet);
// register HostService for 100 sec with the two atributes
5. registration = registrar.register(hostServiceItem, 100000L);
```

In the remaining part of this section we outline the structure of the two prototype services, Host and Broker. Note that there are several other possibilities to create the framework.

### 5.1 The Host service

The role of this service is to provide host system and status information to brokers and to execute allocated client tasks, i.e. to implement a generic compute engine. System information stored are host URL, the number and types of CPUs, the number and types of network interfaces of the host, network latency and bandwidth, topology, memory size, location. The most important status information is whether the host is alive and its current load (not implemented in the prototype).

Each Host service in the framework will implement a `public Object execute(Task t)` method to execute client task `t` of type `Task` interface. There are also other methods for managing processes.

The system and state information will be represented in the registered service as attributes, as Jini uses attribute-based mathcing when searching for services. We have abstracted out attributes such as Processor, Network and Memory for our ptototype. Here we show the details of the Processor attribute.

```
public class Processor extends AbstractEntry {
   Integer number;  // 1..n
   Double speed;    // in MFlop
   String type;     // e.g. Pentium
}
```

### 5.2 The Broker service

The responsibility of the Broker service is to take an execution request from the Client, find suitable machines for execution, then allocate and run tasks on these machines. Allocation is based on requirements received with the execution request (e.g. required performance, maximum number of tasks, maximum allowed cost of execution to be payed for, etc) as well as on machine information extracted from Host services. Scheduling the tasks is a non-trivial operation and its full treatment is beyond the scope of this paper. In our prototype the priorities on scheduling is to minimize the number of processors used and maintain geographical locality of the selected machines. This can only be violated if such a parallel supercomputer is available that can handle the execution of the complete problem itself.

When the broker executes it first looks for available Host services, then registers with the lookup service to become available to clients. The main methods of the Broker are `findMachines()` and `execute()`. The method `findMachines()` takes a list of execution requirements from the Client and returns a set of machine list–cost pairs. If the set is not empty, the Client can invoke the `execute()` method on either a specific set or by specifying allocation priorities such as `minimum_cost`, `maximum_speed`.

## 6 Related work

The two most well-known and most ambitious metacomputing projects are Globus [1] and Legion [2]. Globus is a layered, modular architecture offerinf a range of services to users. Legion on the other hand is an object-oriented metacomputing framework written in MPL.

There are also a number of Java-based metacoputing projects, such as Javelin [4], Webflow [10], Bayanihan [9] and Harness [3]. These all rely on Java features, exploit object-orientation and web technology. They follow a service-based structure similar to Globus and Legion. None of them uses Jini, and as a consequence, special implementations are required to e.g. create and lookup services and provide fault tolerance.

## 7 Conclusions

This paper looked at Jini technology as a possible means of building successful metacomputing environments. It overviewed the main features of metacomputing systems and compared them with built-in Jini features. Building on Jini's discovery mechanism it outlined the structure of a possible Jini service-based metacomputing framework. It also described some details of the Host and Broker service implementation.

The results of our investigation supports the view that Jini can be a very successful technology for building metacomputing systems. Its underlying infrastructure provides very powerful and effective support for distributed systems.

The results reported here are results of an ongoing project. Our short term plan is to complete the full implementation of the framework and experiment with it. There are also several issues that will be have to be addressed in the future. Scalability of the system requires that increasing the size of the system does not create bottlenecks in service access and use. We will explore possibilities of creating a network of registrars as well as an array of brokers to avoid the system relying on a single or small number of registrars and brokers.

The problem of allocating tasks is crucial to the success of such systems. The already complex mapping and scheduling problem is further complicated by the fact that financial measures (cost of computing) should be incorporated in the decision. Charging for resource usage is important to increase users' willingness in participating in metacomputing systems.

There are some complicated areas of Jini, such as the use of transactions, remote events, that make the programming of the framework non-trivial. Since the framework is used for communicating objects, it is our future aim to examine a possible JavaSpaces imlementation of it. JavaSpaces could provide a very elegant, clean and high-level abstraction of the same system.

## References

1. I. Foster and C. Kesselman, The Globus project: a status report,*Future Generation Computer Systems* 15 (1999) pp 607-621.
2. A. Grimshaw, W. Wulf et al. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, vol. (40)1, January 1997.
3. The Harness project, `http://www.harness.org`
4. M.O. Neary, B.O.Christiansen, P. Capello and K.E.Schauser, Javelin: Parallel Computing on the Internet, *Future Generation Computer Systems* 15 (1999) pp 659-674.
5. Sun's Jini home page, `http://java.sun.com/products/jini`
6. Sun's JavaSpaces home page, `http://java.sun.com/products/javaspaces`
7. I. Foster and C. Kesselman, eds. *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kauffamn, 1998.
8. M. Baker and G. Fox, Metacomputing: Harnessing Informal Supercomputers in *R. Buyya [ed.] High Performance Cluster Computing* Vol. 1, Prentic Hall, 1999.
9. L.F.G Sarmenta and S. Hirano, Bayanihan: building and studying a web-based volunteer computing systems usign Java, *Future Generation Computer Systems* 15 (1999) pp 675-686.
10. T. Haupt, E. Akarsu, G. Fox and W. Furmanski, Wb based metacomputing, *Future Generation Computer Systems* 15 (1999) pp 735-743.
11. D. Gelertner, Generative Communication in Linda, *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 1 (January 1985), pp. 80-112.