INTRINSIC SECURITY IN SORCER

(SERVICE ORIENTED) GRID

by

ABHIJIT RAI, B.Tech.

A THESIS

IN

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

Approved

Michael Sobolewski
Chairperson of the Committee

Hector Hernandez

Yu Zhuang

Accepted

John Borrelli
Dean of the Graduate School

May, 2005

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Michael Sobolewski, the chairperson of my committee, for providing the opportunity to work with him. I am extremely thankful to him for dedicating his time, in spite of his busy schedule towards, successful completion of my thesis research. I am extremely grateful for showing his confidence in me. With his constant encouragement and support in designing the solutions for my thesis problem, I was able to complete the work on time.

I feel glad to thank Dr. Hector Hernandez for accepting to be in my committee. As a committee member he has always shown extreme interest towards my academic work by clearly defining the standards and deadlines. As my committee member and graduate advisor, his advices have guided me to successful completion of my work as a Masters Candidate. I would like to thank him for his full cooperation and support for successful completion of my thesis research and graduation.

I am extremely obliged to have Dr. Yu Zhuang as my committee member who has given me intense moral support during the entire course of my research. I feel grateful in thanking him for all the encouragement and interest he has shown towards my research.

I would also like to thank my manager Mr. James Turnbull (Center for Professional Development, Rawls College of Business Administration) for the support nd confidence he has shown towards me

At last but not the least, I would like to express my heartfelt appreciation to all my friends in SORCER lab for their timely help and encouragement. He gave me the

flexibility to work on my thesis along with the work in CPD which allowed me to divide the time comfortably

TABLE OF CONTENTS

TABLE OF FIGURES

ABSTRACT

A grid is a vast repository of virtual services. SORCER is a computational grid environment based on the Service Oriented Paradigm. Security and trust, in SORCER, are of utmost importance since the grid resources and the requestors connecting to faceless service providers are at high risk. For example, if a virus code is sent for computation, the grid resources are at high risk. Similarly, if rogue services are present on the network, requestor's privacy and security are at risk. A security framework for a grid shall ensure access control to the federated services by authenticated and authorized users so that the requestors and services are able to work with mutual–trust.

Today, grids are being used to build the systems which build up, rather than replace, legacy components. This makes securing virtual services even more difficult.

The task of securing the SORCER grid can be accomplished by incorporating the following security practices into the SORCER environment:

- Requestor (Client/Service) Identification and Authentication
- Proxy Verification (building trust)
- Authorization
- Resource Control and Containment
- Privacy and Integrity
- Non-Repudiation
- Accountability (Auditing)

The security mechanism needs to be intrinsic to the grid, so that secure services can be built without being concerned with security on a per service basis. This will greatly reduce the effort required in patching security of individual services.

Our goal is to achieve Intrinsic Security by developing robust, scaleable, and multi-layered security solutions for federated services.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Network can never be free from the fiendish attempts of malicious users who find pride in causing harm to innocent users. Service Oriented Network (SON) is another such network of Services running on a vast repository of resources. The resources and clients of a SON are very vulnerable to malicious intent. For example a client's privacy is endangered if confidentiality is not ensured in data transfer. Resources running the services are at high risk if a malicious user sends a file with virus for computation or execution.

Although Service Oriented Computing is in earlier stages of research, there is a need of an intrinsic security mechanism. One big advantage of intrinsic security mechanism is that if intrinsic security is already provided, the developers will not have to think of security of each of the different services. This will save huge time and effort in patching security holes / compatibility issues due to difference in security mechanisms for different service providers

SORCER is a grid environment based on Service Oriented Paradigm and thus there is a need for security.

1.2 Challenges

The need for security in SORCER is further augmented since SORCER grid promises zero (SORCER) installation on the client (requestor) side.

Zero installation is possible with the help of easily installable Service Browsers such as IncaX. Like web browsers allow a client to browse various pages on the internet, the service browsers enable the users to browse through the available services on the network. The clients can search for the service they intend to use and get the results back accordingly. Presently, there are not many service browsers available and IncaX is the only one we have worked with.

Zero installation necessitates the use of mobile and dynamically downloaded code. This means that for security, it is very important that the downloaded code comes from trusted services and the services shall know who is trying to use them and what he is authorized to do. Otherwise, there are many possible threats to the client, service and the grid itself. Below gives a few cases that might occur if the grid security is not enabled

1.3 Problem Statement

A grid is a vast repository of services. Grids are being built with systems that build up, rather than replace legacy components. This makes securing virtual services more difficult. We need to develop robust, scaleable and multi layered solutions for federated services without disturbing the legacy code. This is to be achieved by deploying following security requirements in SORCER environment:

- Identification and Authentication
- Proxy Verification (building trust)
- Authorization
- Resource Control and Containment
- Privacy and Integrity

- Non-Repudiation
- Accountability (Auditing)

1.4 Thesis Organization

The Thesis report has been divided into five chapters:

Table 1.1 Thesis Organization

Part	Chapters
Problem Introduction	1
Service Oriented Architectures	2
Jini and Sorcer	3
Security Frameworks	4
Intrinsic Security Framework and	5

Chapter 1 provides an introduction to the aspects related problem statement of the thesis. Chapter 2 gives an introduction to the concept of Service Oriented Architectures (SOAs) and then discusses about Web Services and Service Grid architectures. After an introduction to SOAs, The solution (Intrinsic Security) is basically been designed for Service Oriented Grids (SOGs) and has been implemented for SORCER which is based on Jini architecture. Hence an introduction to Jini and SORCER has been provided in Chapter 3 of the report. After introduction to various parts of Jini and SORCER, we give introduction to Security. Chapter 4 discusses various security frameworks which have been used in implementation of the thesis. These security frameworks include JAAS and Jini security frameworks. Chapter 5 introduces the proposed Security framework (the solution) and its implementation to the SGrid framework.

CHAPTER 2

SERVICE ORIENTED ARCHITECTURES

Before we can go on with our specific solution for monitoring of Service Oriented Programs, we need to understand the architecture of the environment we are dealing with. This chapter discusses about different Service Oriented architectures like Web Services, Grid Services (Globus), Jini, Rio and also looks into some of the work being done currently in the field of Grid Monitoring and Autonomic Provisioning. This chapter also provides a high level structural & operational view of the components. To make things more concrete several ongoing projects are outlined as an illustration

2.1 Web Services

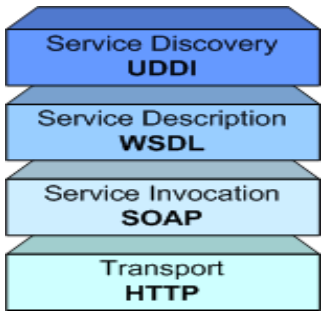

Figure 2.1 Web Service Protocol Stack

Web Service [1] is a protocol based Service Oriented Architecture. The above figure (Figure 2.1) describes the protocol stacks involved in the WS architecture. It consists of the following protocols

2.1.2 Service Discovery

This part of the architecture allows us to find Web Services which meet certain requirements. This part is usually handled by UDDI (Universal Description, Discovery, and Integration). One of the most interesting features of Web Services is that they are self-describing. This means that, once you've located a Web Service, you can ask it to 'describe itself' and tell you what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL).

2.1.3 Service Invocation

Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responds. In theory, we could use other service invocation languages (such as XML-RPC, or even some ad hoc XML language). However, SOAP is by far the most popular choice for Web Services.

2.1.4 Transport

Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responds. In theory, we could

use other service invocation languages (such as XML-RPC, or even some ad hoc XML language). However, SOAP is by far the most popular choice for Web Services.

2.2 WS Interaction Model

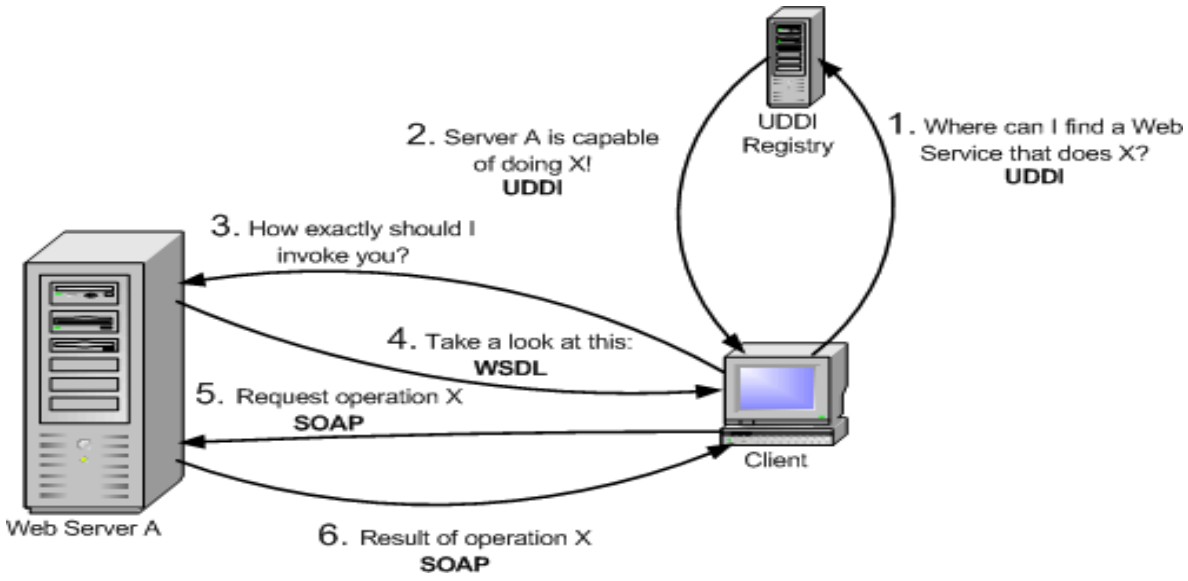

Figure 2.2 WS Component Interaction

The following are the interactions between different web service components

- A client with no knowledge of what Web Service it is going to invoke, contacts a UDDI registry for a particular type of service.
- The UDDI registry will reply, telling us what servers can provide us the service we require (e.g. the temperature in US cities)
- We now know the location of a Web Service, but we have no idea of how to actually invoke it. Hence we have to ask the Web Service to describe itself
- The Web Service replies in a language called WSDL.

- Now that via WSDL, we know the interface and invocation can be done in a language called SOAP.
- The Web Service will reply with a SOAP response.

2.2.2 WS As Service Oriented Architecture

WS is clearly conjunctive as the WSDL defines the interface and different services can be combined together in the form of WSDL. But still it lacks the versatility and richness of CORBA or as a matter of fact other object oriented technologies when it comes to definition of the interface. WS as an architecture may be independent of platform, transport and environment. But still it's heavily protocol dependant. History has shown us that as protocols get used by systems in a large scale, it's almost impossible to change or improve. One of the best known examples is that of SMTP protocol which still remains insecure in spite of the evolution of better protocols.

WS invariably are hosted in Application Severer or cluster of Application Servers. Autonomic Monitoring and Management here means monitoring and management of the applications inside the web servers or web servers themselves. The term monitoring is used with application servers to describe about the monitoring of the applications running inside and they do not address the monitoring of the programs run by the services.

2.3 Grid

The term "Grid" was coined in the mid-90s to refer to an advanced science and engineering compute infrastructure [2]. A simplest analogy to help describe a grid is the electricity grid [3]. When you turn on the power to your television you do not care where

your power comes from or when/how it was generated. While intuitively appealing, there is however, a need for a clearer, more precise definition of a grid and grid computing. This has been detected in literature and numerous formal as well as informal [4][5]definitions have been proposed. This dissertation will stick to the extensive definition presented in [2]:

"A grid is a large scale geographically distributed hardware and software infrastructure composed of heterogeneous networked resources owned and shared by multiple administrative organizations which are coordinated to provide transparent, dependable, pervasive and consistent computing support to a wide range of applications. These applications can perform any of the following: distributed computing, high throughput computing, on-demand computing, data-intensive computing, collaborative computing or multimedia computing"

As the definition implies, there exists different types of grids, most common being [8]:

- Compute grid: distributed compute resources consisting of desktop, server and High Performance Computing systems
- Data grid: distributed storage devices (tape/disk/... devices), along with the necessary software
- Access grid: distributed audio-visual equipment (cameras, microphones, speakers ...) set up to provide a virtual collective presentation room

Since SORCER is basically a computational grid (at east at its present capacity), this thesis focuses mainly on computational grid systems. A computational grid system can be defined as "a type of parallel and distributed system that enables sharing, selection

and aggregation of resources distributed across the multiple administrative domains based on their availability, capability, performance, cost, and users' quality of service requirements" [9]. Grid systems allow one create supercomputer capability out of collections of different computer types.

2.4 Grid Architecture

Since a grid is a virtual architecture its constituent components are individually unimportant. The key concept is how these components work together as a unified resource. The components include [10] (see Figure 2.3):

- Processors and Memory
- Networks and Communications Software
- Virtual Environment or Middleware: the grid computing counterpart of an OS2. Used to configure, manage and maintain the grid environment. Usable by both administrators and individual users.
- Remote Data Access and Retrieval.

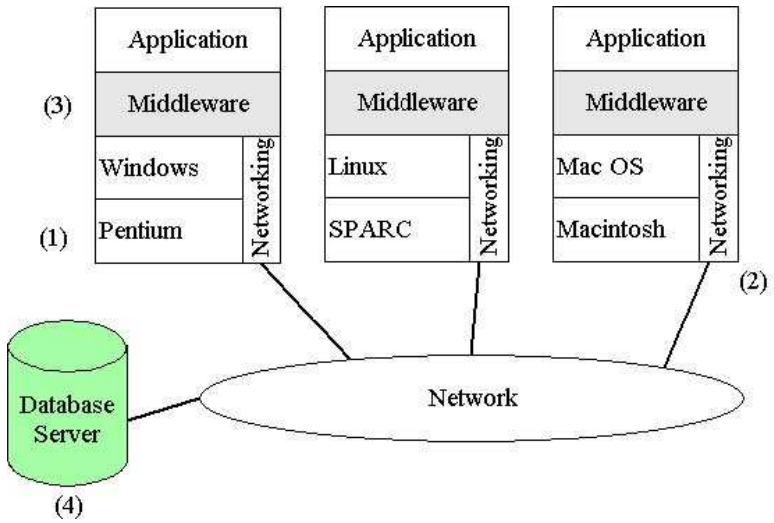

Figure 2.3 Positioning of Middleware in Distributed Systems

The most important component is the virtual environment or middleware. Since a real world distributed system encompasses a plethora of architectures, operating systems, communication protocols... developing a decent distributed application without some intermediate abstraction layer is literally impossible. The middleware provides exactly that abstraction layer. It is responsible for turning a radically heterogeneous environment into a virtual homogeneous one.

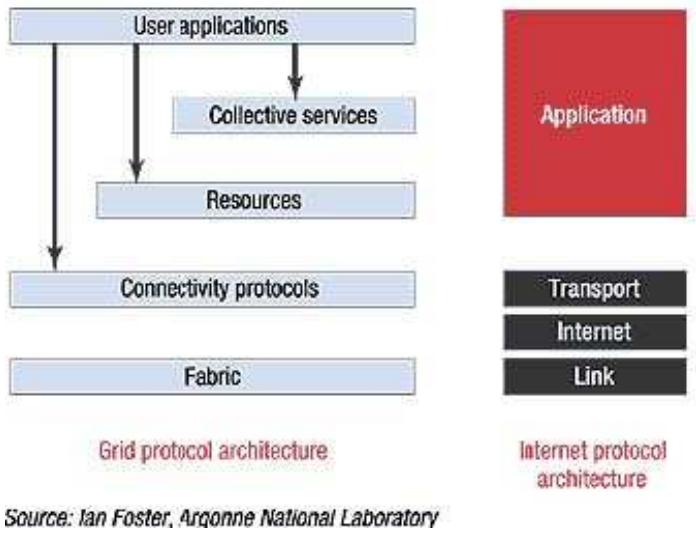

Figure 2.4 Grid Protocol Architecture

The model in Figure 2.5 is also known as the hourglass model. The narrow neck of the hourglass defines a set of core abstractions and protocols onto which many different high-level behaviors can be mapped (the top), and which themselves can be mapped onto many different underlying technologies (the base) [19]

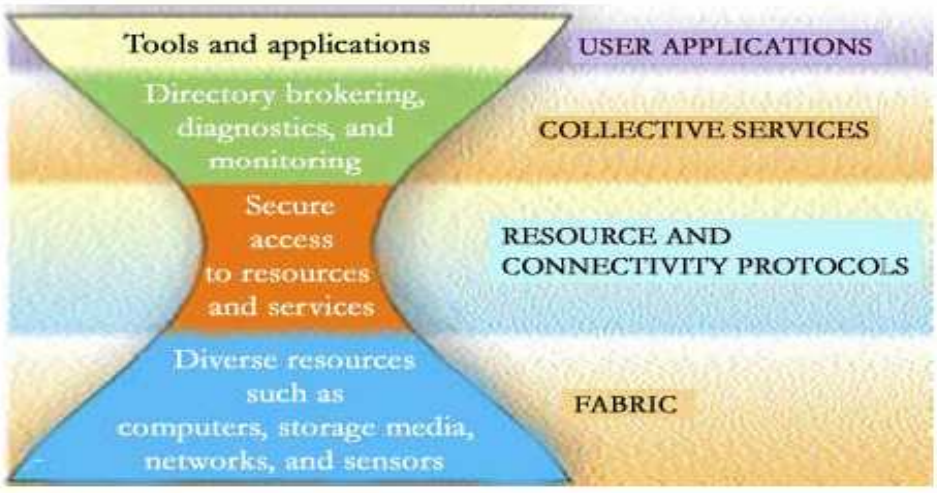

Figure 2.5 The Hour Glass Model

The next chapter covers another Service Oriented Architecture – Jini. Since Jini is the base on which SORCER has been developed, it has been discussed in depth in the coming chapter. Later we also discuss about Security provisions in the latest versions of Jini and how these have been utilized to achieve security in a Service Oriented Grid Environment.

CHAPTER 3

JINI AND SORCER

3.1 Introduction

Jini was the result of evolution of Java technology to make distributed computing easier. Jini aims at making network devices and network computing into standard components of networking environment. In this way all the devices can be available on the network as services and interact with each other. For example, the alarm clock service can ask the coffee maker service to switch on, 5 minutes before it wakes you up. In the era of growing wireless networks, Jini offers higher level of interaction between different services on the network [11].

Jini offers plug and play of devices and services. When a device gets connected with the network, it can announce its presence and another service can locate it to perform specific tasks. Mobile devices, computer hardware and software etc can announce themselves as services available on the network.

Jini is not an acronym and doesn't have a particular. It is basically a federation of services and clients communicating using Jini protocols. The focus of the system is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly. A Jini system consists of the following parts: 1) A set of components that provides an infrastructure for federating services in a distributed system 2) A programming model that supports and encourages the production of reliable distributed services 3) Services that can be made

part of a federated Jini system and which offer functionality to any other member of the federation [13].

The Jini system extends the Java application environment from a single virtual machine to a network of machines. The Java application environment provides a good computing platform for distributed computing because both code and data can move from machine to machine. The environment has built-in security that allows the confidence to run code downloaded from another machine. Although Jini has been written in Java language, it does not put any constraint on the services/clients. The services and clients can be written in any language with proper wrappers.

3.1.1 Services

A service is an entity on the network which can be used by another service, a client (a person or another program). It can be a computation (proth), storage (File store, Database), a communication channel to another user (Chat service), a software filter, a hardware device (a printer), or another user.

Members of a Jini system federate in order to share access to services. The services of a Jini system can be collected together for the performance of a particular task. Services may make use of other services, and a client of one service may itself be a service with clients of its own. The dynamic nature of a Jini system enables services to be added or withdrawn from a federation at any time according to demand, need, or the changing requirements of the workgroup using it.

Services in a Jini system communicate with each other by using a *service protocol*, which is a set of interfaces written in the Java programming language. The set

of such protocols is open ended. The base Jini system defines a small number of such protocols which define critical service interactions.

3.1.2 Service Registration (The Lookup Service)

Services are found and resolved by a *lookup service*. The lookup service is the central bootstrapping mechanism for the system and provides the major point of contact between the system and users of the system. In precise terms, a lookup service maps interfaces indicating the functionality provided by a service to sets of objects that implement the service. In addition, descriptive entries associated with a service allow more fine-grained selection of services based on properties understandable to people.

Objects in a lookup service may include other lookup services; this provides hierarchical lookup. Further, a lookup service may contain objects that encapsulate other naming or directory services, providing a way for bridges to be built between a Jini Lookup service and other forms of lookup service.

A service is added to a lookup service by a pair of protocols called *discovery* and *join*--first the service locates an appropriate lookup service (by using the *discovery* protocol), and then it joins it (by using the *join* protocol).

3.1.3 Java RMI (Remote Method Invocation)

Communication between services can be accomplished using Java Remote Method Invocation. The infrastructure to support communication between services is not itself a service that is discovered and used but is, rather, a part of the Jini technology infrastructure. RMI provides mechanisms to find, activate, and garbage collect object groups.

Fundamentally, RMI is a Java-programming-language-enabled extension to traditional remote procedure call mechanisms. RMI allows not only data to be passed from object to object around the network but full objects, including code. Much of the simplicity of the Jini system is enabled by this ability to move code around the network in a form that is encapsulated as an object.

3.1.4 Leasing

Access to many of the services in the Jini system environment is *lease* based. A lease is a grant of guaranteed access over a time period. Each lease is negotiated between the user of the service and the provider of the service as part of the service protocol: A service is requested for some period; access is granted for some period, presumably taking the request period into account. If a lease is not renewed before it is freed--either because the resource is no longer needed, the client or network fails, or the lease is not permitted to be renewed--then both the user and the provider of the resource may conclude the resource can be freed.

Leases are either exclusive or non-exclusive. Exclusive leases insure that no one else may take a lease on the resource during the period of the lease; non-exclusive leases allow multiple users to share a resource.

3.1.5 Transaction Management

A series of operations, either within a single service or spanning multiple services, can be wrapped in a *transaction*. The Jini Transaction interfaces supply a service protocol needed to coordinate a *two-phase commit*. How transactions are implemented--and

indeed, the very semantics of the notion of a transaction--is left up to the service using the interfaces.

3.1.6 Event Management

The Jini architecture supports distributed *events*. An object may allow other objects to register interest in events in the object and receive a notification of the occurrence of such an event. This enables distributed event-based programs to be written with a variety of reliability and scalability guarantees.

3.1.7 Discovery and Lookup Protocols

Services form the interactive basis for a Jini system, both at the programming and user interface levels. The details of the service architecture are best understood once the Jini Discovery and Jini Lookup protocols are presented.

The heart of the Jini system is a trio of protocols called *discovery*, *join*, and *lookup*. A pair of these protocols--discovery/join--occurs when a device is plugged in. Discovery occurs when a service is looking for a lookup service with which to register. Join occurs when a service has located a lookup service and wishes to join it. Lookup occurs when a client or user needs to locate and invoke a service described by its interface type (written in the Java programming language) and possibly, other attributes. The following diagram outlines the discovery process.

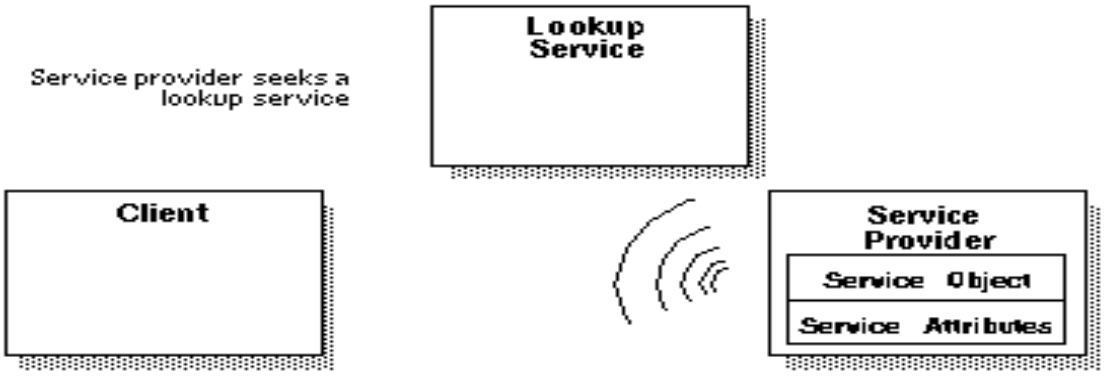

Figure 3.1 Discovery

Discovery/Join is the process of adding a service to a Jini system. A service provider is the originator of the service--a device or software, for example. First, the service provider locates a lookup service by multicasting a request on the local network for any lookup services to identify themselves (Figure 3.1). Then, a service object for the service is loaded into the lookup service (Figure 3.2). This service object contains the Java programming language interface for the service including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes.

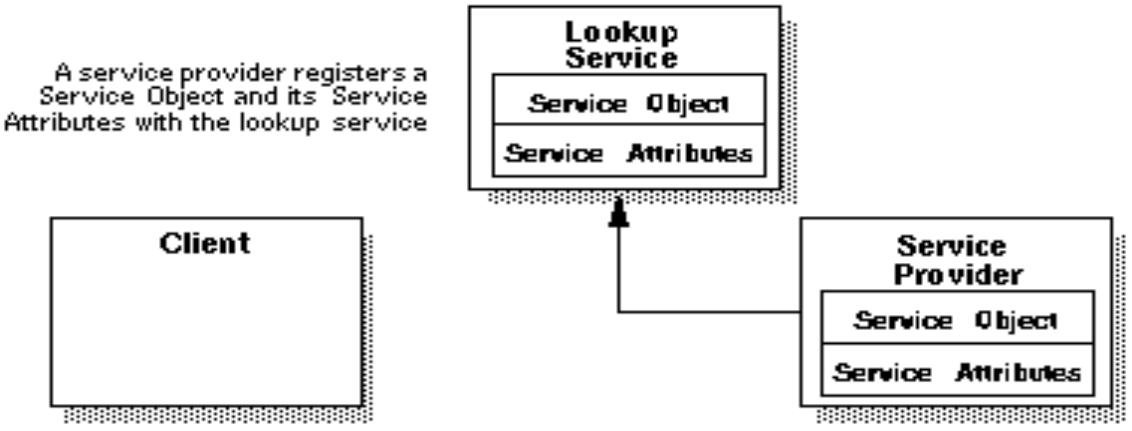

Figure 3.2 Join Protocol

Services must be able to find a lookup service; however, a service may delegate the task of finding a lookup service to a third party. The service is now ready to be looked up and used, as shown in the following diagram (Figure 3.3).

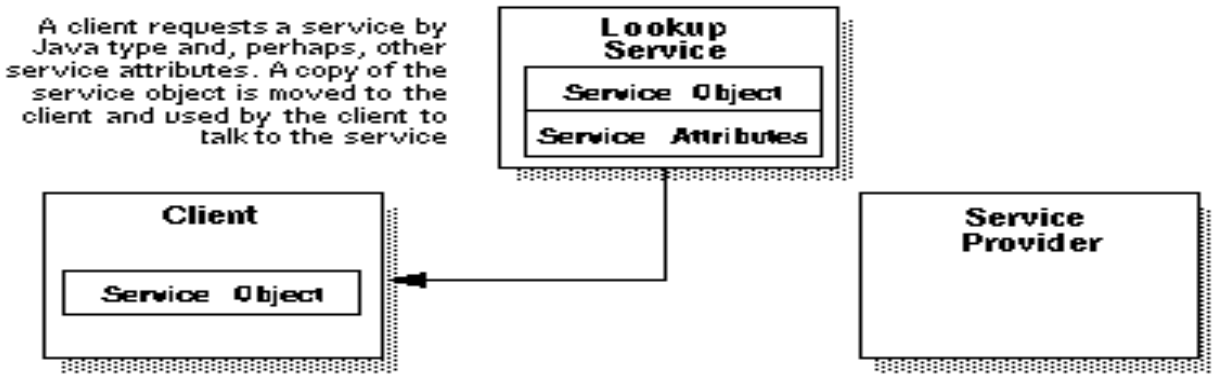

Figure 3.3 Lookup

A client locates an appropriate service by its type--that is, by its interface written in the Java programming language--along with descriptive attributes which are used in a user interface for the lookup service. The service object is loaded into the client.

The final stage is to invoke the service, as shown in the following diagram (Figure 3.4). The service object's methods may implement a private protocol between itself and the original service provider. Different implementations of the same service interface can use completely different interaction protocols.

The ability to move objects and code from the service provider to the lookup service and from there to the client of the service gives the service provider great freedom in the communication patterns between the service and its clients. This code movement also ensures that the service object held by the client and the service for which it is a proxy are always synchronized, because the service object is supplied by the service

itself. The client only knows that it is dealing with an implementation of an interface written in the Java programming language, so the code that implements the interface can do whatever is needed to provide the service. Because this code came originally from the service itself, the code can take advantage of implementation details of the service known only to the code.

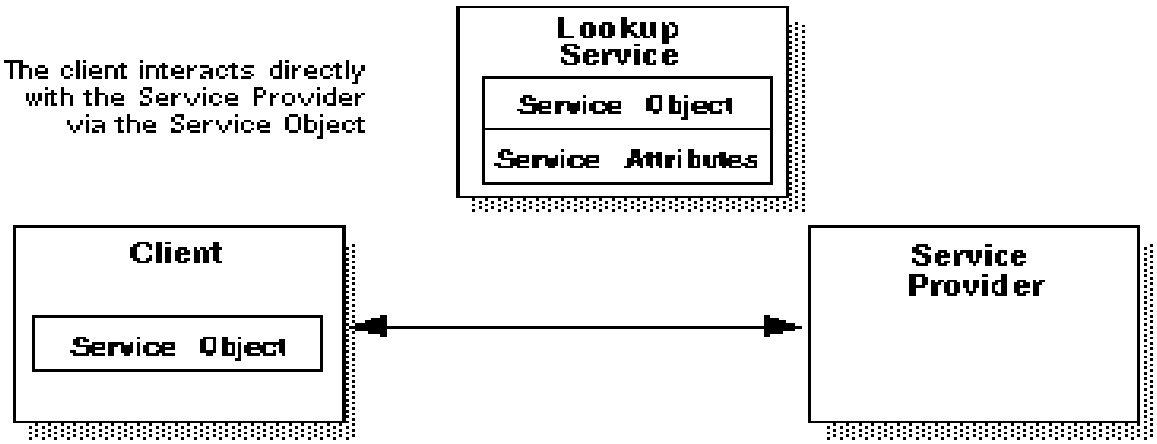

Figure 3.4 Service Invocation

The client interacts with a service via a set of interfaces written in the Java programming language. These interfaces define the set of methods that can be used to interact with the service. Programmatic interfaces are identified by the type system of the Java programming language, and services can be found in a lookup service by asking for those that support a particular interface. Finding a service this way ensures that the program looking for the service will know how to use that service, because that use is defined by the set of methods that are defined by the type.

Programmatic interfaces may be implemented either as RMI references to the remote object that implements the service, as a local computation that provide all of the

service locally, or as some combination. Such combinations, called *smart proxies*, implement some of the functions of a service locally and the remainder through remote calls to a centralized implementation of the service.

A user interface can also be stored in the lookup service as an attribute of a registered service. A user interface stored in the lookup service by a Jini service is an implementation that allows the service to be directly manipulated by a user of the system.

In effect, a user interface for a service is a specialized form of the service interface that enables a program, such as a browser, to step out of the way and let the human user interact directly with a service.

In situations where no lookup service can be found, a client could use a technique called *peer lookup* instead. In such situations, the client can send out the same identification packet used by a lookup service to request service providers to register. Service providers will then attempt to register with the client as though it were a lookup service. The client can select those services it needs from the registration requests it receives in response and drop or refuse the rest.

3.2 Jini Standard Services [24] [32]

Jini framework is included with some standard (default) services. These services have special purpose defined and are critical to the proper functioning of the Jini framework. These services have been outlined below:

3.2.1 The Lookup Service – Reggie

Reggie is the Jini service which is an implementation of "The Registrar". It registers services (including itself), removes registrations when the lease expires and can perform unicast and multicast discovery to find different services available on the network. Reggie is an activatable process which implies that it is started by RMI activation daemon (rmid) only when it is first needed [22].

3.2.2 The Transaction Manager – Mahalo

Mahalo [23] is the transaction manager service provided by Jini framework. It implements Jini TransactionManager interface but not the NestableTransactionManager interface because nested transactions are not supported in mahalo.

3.2.3 The JavaSpaces – Outrigger

Outrigger [23] implements Jini JavaSpaces specification. It can be started either as a transient or as a persistent space. Persistent spaces are useful for mini databases, but storage updates slows it down and being activatable services they are considerably hard to start. Transient spaces, however, are easier to start and if performance is the prime concern they are preferable choices.

3.2.4 Lease Renewal Service – Norm

Norm [23] is another bookkeeping service in Jini Infrastructure which keeps track of service leases. It is an activatable service and keeps a persistent transaction log of

leases and registrations. It is especially needed for the activatable and mobile services which can hand over the lease to Norm and become safely inactive.

3.2.5 The Jini Lookup discovery Service – Fiddler

Fiddler [23] is the lookup discovery service in the Jini framework. Once a client finds a service registrar (Reggie), it performs useful work that doesn't usually include bookkeeping registrars. On the other hand, it is important for a client to talk to existing registrars that actually exist, so the bookkeeping is a necessary activity.

Just as bookkeeping is outsourced to accountants, Jini clients can outsource the bookkeeping of lookup services to lookup discovery service. This is an important pattern in Jini: if you see common behavior, it can be made into a different service.

Fiddler can be started as an activatable service (with persistence) or directly from the command line without persistence.

3.2.6 Event Mailbox Service – Mercury

Mercury [23] is the Jini event mailbox. service. This service has been introduced so that it can collect events on behalf of the clients and send them the envents when the clients specially request for them. This is needed because Activatable (and other) services are not always available, and mobile services may actually be physically removed from the network altogether. Event registrations may be problematic in this case: an activatable service would be activated for every event sent to it and a mobile service simply wouldn't get them.

3.3 SORCER

SORCER, [12] which stands for Service Oriented Computing Environment, is a service-based concurrent engineering project which is based on evolution of the concepts and lessons learned in the FIPER project [14], a $21.5 million program founded by NIST (National Institute of Standards and Technology)[1][2][14[15] at GE Corporate Research and Development. SORCER project aims to develop a means for global communication of product information, data, methods, and tools while satisfying stringent product performance requirements. The architecture of the SORCER system is designed to be flexible enough to handle the needs of almost any product from aircraft engines to manufactured goods such as plastics.

The architecture of the SORCER [31][30][29]system is service-based, network-centric, and web-centric. This architecture houses the large pool of distributed services that execute business logic and integrate tools and applications in the underlying engineering domain. The web-centric architecture enables HTTP communication between a web-based client and the SORCER system, as well as transparent access to the globally distributed data and the pool of federated services. The individual services requested by the SORCER system act on behalf of the web client, both in the role of providing services (provider mode) and requesting services (requestor mode). When requesting services, the SORCER system also brokers the requests, delegating them to the appropriate registered service providers.

3.3.1 Service Oriented Program (SO)

Service Oriented Programming (SO) [31][30][29][12] is a paradigm for distributed computing built over Object Oriented Programming (OOP) paradigm emphasizing the point that problems can be modeled in terms of services rather than objects. SO differs from OOP by focusing on what things can do whereas OOP focuses on what things are and how they are constructed. SO defines set of core principles to maintain interoperability of services over time. This section explains SO in greater detail.

The structured computing paradigm is a method based on a concept that a system has data and functionality (behavior) separated into two distinct parts. A structured program is composed of one or more units or modules such that each module is composed of one or more functions (procedures, routines, subroutines, or methods, depending on programming language). It is possible for a structured program to have multiple levels or scopes, with functions defined inside other functions. Each scope can contain variables, which cannot be seen in outer scopes. Usually the structured computing is based on splitting programs into sub-sections, each with a single point of entry and of exit, by using only structured looping constructs, often named "while", "repeat", "for" with simple, hierarchical program flow structures. Often it is recommended that each loop should only have one entry point and one exit point.

The object-oriented paradigm, on the other hand, defines a system as a collection of interacting active objects. These objects do things and know things, or stated equivalently they have functions and data that complement each other. Usually an object-oriented system creates its own object space instead of accessing a data repository. This

object space constitutes an object-oriented program. The execution of the object-oriented program is a collection of dialoguing objects (sending and receiving messages).

Building on the object-oriented paradigm is the service-oriented paradigm, in which the objects are distributed, or more precisely they are network objects and play some predefined roles. A service provider is an object that accepts messages from service requestors to execute an item of work – a task. The task object is a service request – a kind of elementary service executed by a service provider. A service broker is a specialized service provider that executes a job – a compound request in terms of tasks and other jobs. The job object is a service-oriented program that is dynamically bound to all relevant and currently available service providers on the network. This collection of service providers dynamically identified by a broker is called a job federation. This federation is also called a job space. While this sounds similar to the object-oriented paradigm, it really isn't. In the object-oriented paradigm the object space is a program itself; here the job space is the execution environment for the job itself and the job is a service-oriented program that federates relevant providers at runtime. This changes the game completely. In the former case the object space is a virtual machine, but in the latter case the job space is the virtual federating network. This runtime federation is the jobs' execution environment and the job object is a service-oriented program. In other words, we apply the object-oriented concepts directly to the network in the service-oriented paradigm. Tasks and jobs as elementary and compound service-oriented programs, respectively, are called exertions.

The complexity of the problems to be solved is directly related to the kind and quality of abstraction. The primary network abstraction still requires us to think in terms

of structure of many computing nodes and devices rather than the structure of the problem we are trying to solve. Instead of modeling the multiple computing devices the service-oriented programming provides the paradigm that allows us to model the problem to be solved in terms of services and define the computing processes in terms of exertions.

3.3.2 SO in SORCER

Representation of a SO is the key to any SOC framework. SORCER has a very clear separation in defining a representation for a service, the data to that service and the method to be invoked in that service.

A service is represented by what is called a ServiceMethod. A service method is a reference to a Service Oriented Routine (SOR). A Service Provider runs a set of SORs which are exposed to service requestors via a provider interface methods. Technically, a SOR is represented by the pair (i, s), where i is the name of the provider interface and s is the name of method selector.

Service data is what is to be passed as an argument to the SO in runtime. This argument to SOR is called a context model, for short c. A context model [20] has a tree structure and has name space and data in leaves nodes. The following figure (Figure 3.5) depicts how a context model looks.

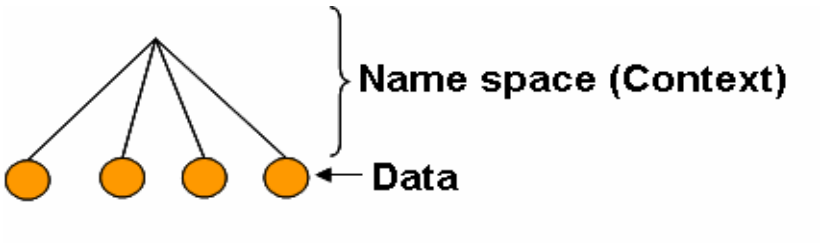

Figure 3.5 Context Model

A task is an elementary grid operation and is defined by data and what to do with the data represented by ServiceMethod i.e. task T = (c, m) where m is the ServiceMethod and c is the context.

A job is a compound exertion which is a collection of tasks and other jobs. The job defines the task graph model and also encapsulates other information like the execution and control strategy for the job. A job J = (c, m) where m is the ServiceMethod and c is the context model in which, the data nodes are the tasks. The context model for job also captures meta-information required for the defining the control strategy of execution of the SO. This meta-model is also called the Control Context Model.

The definition of control strategy by the Control Context adds greater ability to the representation of the SO. Control Context can define not only the task graphs but also represent other conditional and iteration aspects involved in the execution of a SO.

A Job aggregation "oi" can be represented by Control Context model which can support the following operation for the execution of SO: ⏐, ‖, ┤, *, [ ] where,

⏐(e1, …, en) means sequential execution of SO

‖(e1, …, en) means parallel execution

┤ (e1, e2, e3) – conditional

*(e1) - iteration

[ ](e1, …, en) – selected

It can be seen that both the task T and job J has same structure. It contains data represented in the form of a context model and also method which says what to do with that data. Hence both job and task can be represented by an exertion which is a notion for a distributed activity which can be either atomic or compound. The following UML describes the UML representation of a SO in SORCER. This UML shows the client site representation of SO and service site representation of SO. Essentially both represent the same thing except for some added functionalities in terms of additional operations on objects representing the SO in server site.

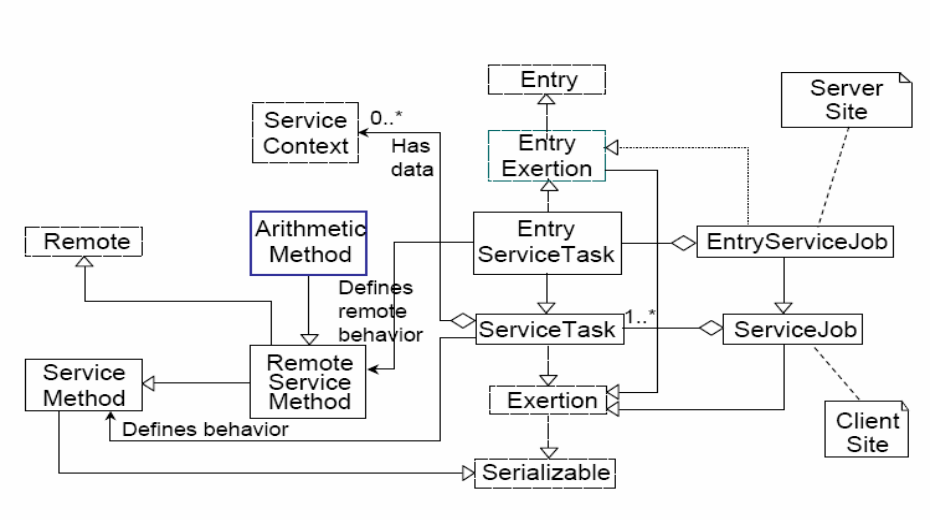

Figure 3.6 SO in Sorcer

3.3.3 Service Oriented Runtime Execution Environment (SOREE)

In SORCER, the SOREE framework is designed using the Jini Network Architecture. Some of the core Jini services used in the SOREE are as follows:

- Lookup Service: For dynamic discovery of services in the network based on rich template matching
- Transaction Manager: For implementing the transactional semantics between service providers.
- Java Spaces: For space based computing where tasks can be dropped and providers can pick up according to their capabilities

3.3.4 Execution of SO in SORCER

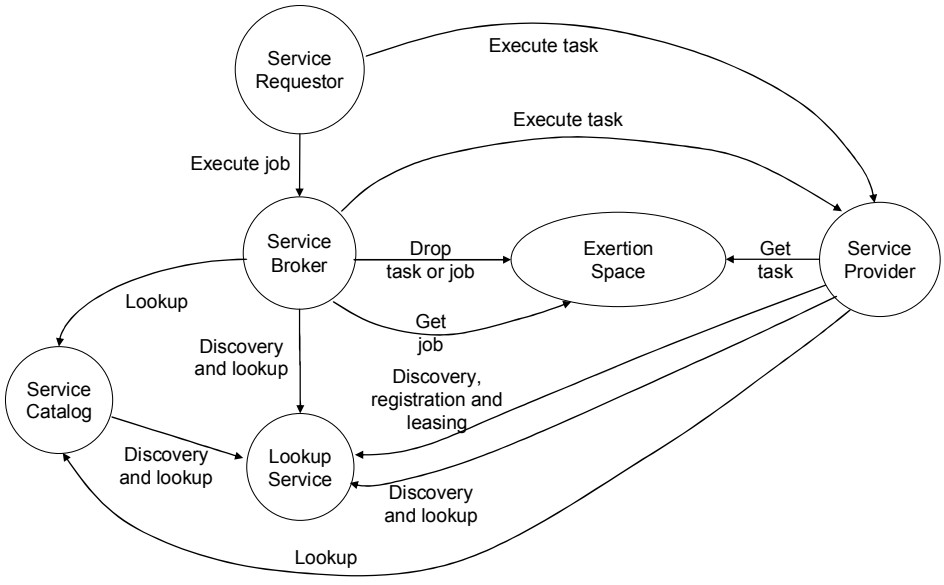

Figure 3.7 Execution of SO in SORCER

The above diagram depicts a typical execution of exertions in an SO framework. The requestor creates a SO (task/job) and submits it to the Service Broker or directly to a Service Provider. The Service Provider / Service provider are found by the Service Requestor in a dynamic fashion with the help of the Service Method defined in the exertion.

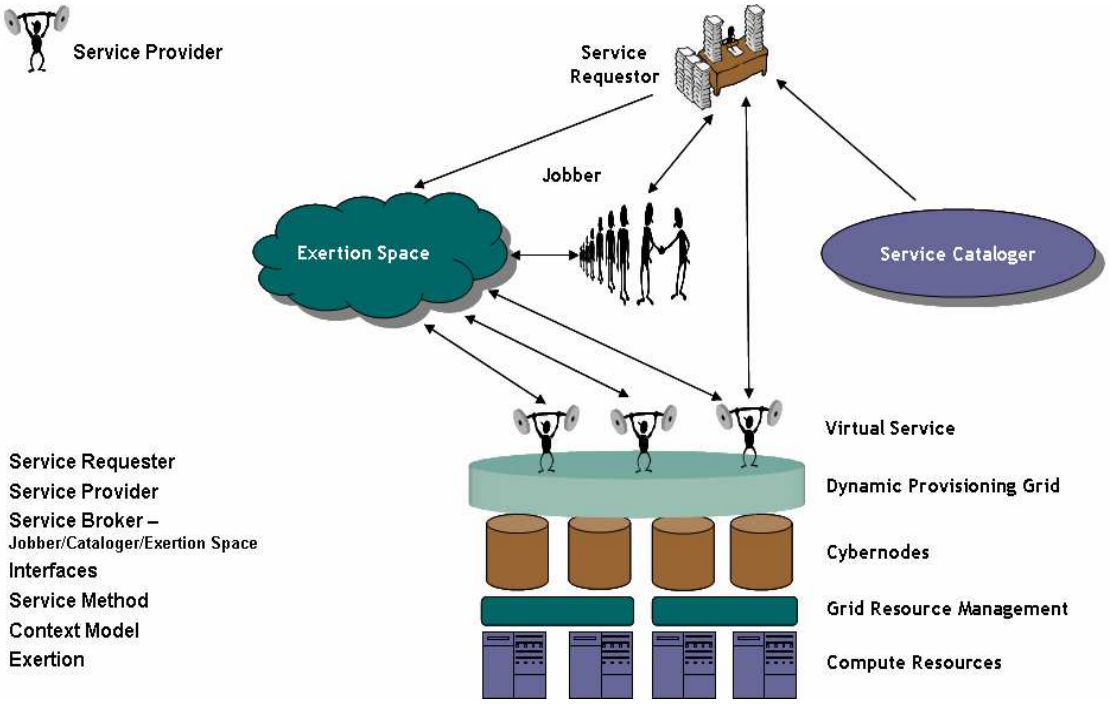

Figure 3.8 SORCER Conceptual View

The lookup service (LUS) and the cataloger help in the dynamic discovery of services in a SO framework. This dynamic discovery can be done by lookup service in many ways. In the specific implementation of JINI, it works in the following way. When Services bootstrap, they either listen for a multi cast announcement made by all the lookup services (Multicast announcement protocol) in the network or sends multicast packets to other LUS requesting for information about the location of the LUS (Multicast announcement protocol). Once it gets a multicast packet, it knows the location of a

lookup service. Then the ServcieProvider registers with the lookup service the proxy for its service. Thus all the lookup service in the network is aware of any particular service. Lookup services require that the client maintain leases so that they can cleanup a stale proxy if a client crashes or does not renew leases due to some reasons.

Though lookup service contains all the proxies for the services in the grid, it's important for the framework to have a Catalog Service. A catalog helps in three ways. The first one is that the catalog is on a constant lookout for changes in all lookup services and keeps a dynamic catalog of all the services in the lookups and maintains a cache to aid fast discovery. The second one is more important one. It functions as a filter to filter out only required services by the framework. To understand the third use, it requires an understanding of how services are requested from a lookup service. Typically a requestor will ask the lookup for a particular proxy based on a top level interface which a service implements and some other attributes. But for this, the requestor requires the class definition of interface. This may not be possible in service oriented grids because it requires that if a new service joins the federation, all other members in the federation must be updated with the class definition of the new interface to be able to talk to the new service. Service Catalogs can be used to solve this interesting problem.

Assume there is a global interface called Servicer which contains one method called "service(Exertion)".

```
public interface Servicer {
// Put into action the specifiedexertion
public Exertion service(Exertion exertion) throws
          RemoteException, ExertionException;
… }
```

All Providers will implement their own interface say XXInterface and extend an abstract ServiceProvider This abstract ServiceProvider implements the global interface Servicer. The implementation of service method of the global interface is same for all providers. They will look into ServiceMethod of the SO and do a "self inspection" and check if it implements the interface and method selector defined in the Service Method. If it does, it will call on itself the method defined by the method selector by passing the data in the SO as argument to it. This way all Service providers know one interface Servicer, and all providers are themselves Servicers forming a true P2P federation. Now a service like Service Cataloger will dynamically find all proxies for all ServiceProviders. Note that at this point the Cataloger does not have the interface definition of any of the services. The cataloger then via mechanism of reflection will find out the name of all interfaces a service implements and the methods in the interfaces. It then keeps a map of this information. Now a requestor can directly ask the catalog to give it a service defined by a ServiceMethod because cataloger knows the name of all the Interfaces and the methods contained in all the interfaces. Once the requestor gets the proxy, it just calls service method on the proxy and passes a SO.

A service broker is a specialized service which knows how to co-ordinate the execution of compound SOS. It might also employ complex scheduling algorithms. It might dispatch the exertions in two ways. It might ask the Cataloger to give a proxy for a particular service defined by the ServiceMethod and request for execution or it might put the SO in space for service providers to pick up from space by themselves.

The collection of Service Providers formed for executing the SO by a broker is called a ***Federation***. The ServiceMethod in the exertion define the matching provider

and thus the exertions get bound to the right service providers during runtime execution of the SO by broker.

The below figure (Figure 3.9) explains the implementation details of the core components in the system which involves in job execution. Jobbers use dispatchers to dispatch jobs to the right providers in the grid and thus create the federation required for job execution. In SORCER environment there are four types of dispatchers that implement different type for control strategies. These include sequential and parallel dispatchers for Catalog and Space. A relevant dispatcher is assigned to a jobber by the dispatcher factory based on the information captured in the ControlContext Model of the job.

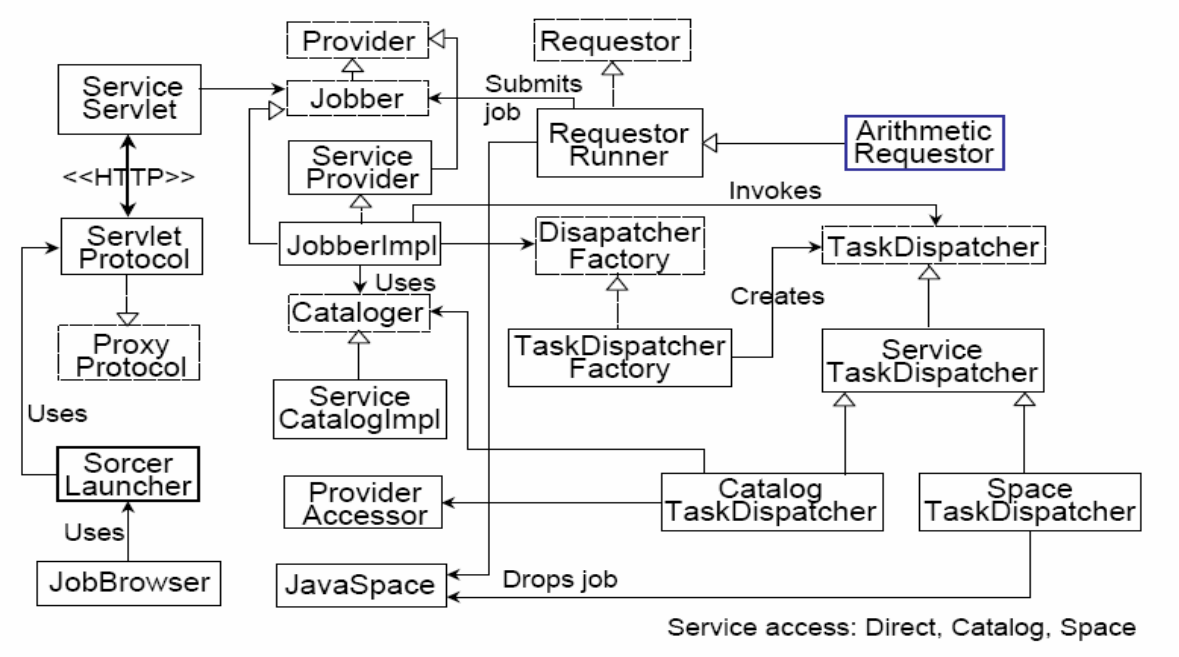

Figure 3.9 Job Execution in SORCER

Dispatchers does service discovery via Cataloger or via ProviderAccessor. While Cataloger is a service in itself whose responsibility is to find other service providers dynamically, ProviderAccessor is a client side utility which helps in discovery of services. For dropping exertions in space, the dispatcher does not require any discovery as providers themselves pick up the exertion and execute them.

3.3.5 SORCER Functional Architecture

Framework

This sections summarizes the description of SO given in SORCER and extends the explanation of capabilities and features of SORCER as a P2P (Peer to Peer) environment.

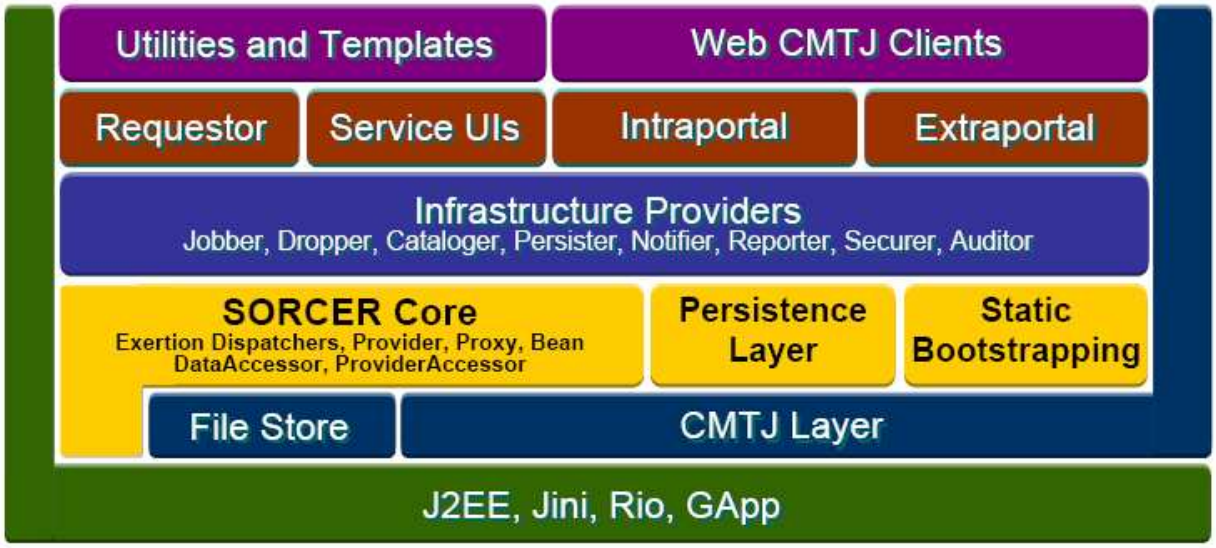

Figure 3.10 SORCER Functional Architecture

The P2P service-oriented framework targets complex business and engineering transactions. A transaction is composed of a sequence of activities with specific precedence relationships. The grid contains service providers that offer one or more services to other peers on the overlay network. Service providers do not have mutual associations prior to the transaction; they come together (federate) dynamically for a specific transaction. Each provider in the federation performs its services in a choreographed sequence. Once the transaction is complete, the federation dissolves and the providers disperse and seek other transactions to join. The architecture is service centric in which a service is defined as an independent self-sustaining entity performing a specific network activity. Each service is defined by a well-known public interface. A service provider that plans to offer a service implements its interface or multiple interfaces (services) to be eligible for participating in federations.

The same provider can provide multiple services in the same federation and different providers can provide the same service in different federations. The service grid is dynamic in which new services can enter the overlay network and existing services can leave the network at any instance. The service-based architecture is resilient, self-healing, and self-managing. The key to the resilience is the transparency of search and seamless substitution of one service with another. The architecture allows services to share data by using specialized data services or a shared data repository (distributed file store). The architecture also allows asynchronous execution of activities such that an activity can wait for a service to be available

The architecture uses *Jini* network technology (Jini Architecture Specification 2000; Jini.org; Edwards, 2000) and *Java Spaces* technology (Freeman, 1999; Halter, 2002) for implementing the service-based overlay network described above. However, the proposed service-oriented architecture is abstract and can be implemented using any distributed network technology that provides support for dynamic discovery of resources and a rich software development environment.

Design

A UML-diagram showing the framework of the system developed is illustrated in Figure 3.11.The core of the architecture consists of service providers and service brokers interacting with lookup registries, a catalog of services, and exertion shared space. In general, a service provider executes a task (elementary exertion), and a service broker executes a job (compound exertion or a nested transaction). While executing a job, the service broker (Figure 3.11) coordinates exertion execution within the nested transaction.

It interprets the transaction map supplied by the service requestor and completes the nested exertions accordingly as presented in the transaction map.

At the start of the transaction the service broker reads all the exertions in the transaction and executes those exertions, which have no precedence relationships. At each step it executes the services for which all the precedence relationships have been satisfied (services complete). Whenever it gets a notification of a service being completed it evaluates the remaining unfinished activities and invokes one or more exertions based on their precedence relationships.

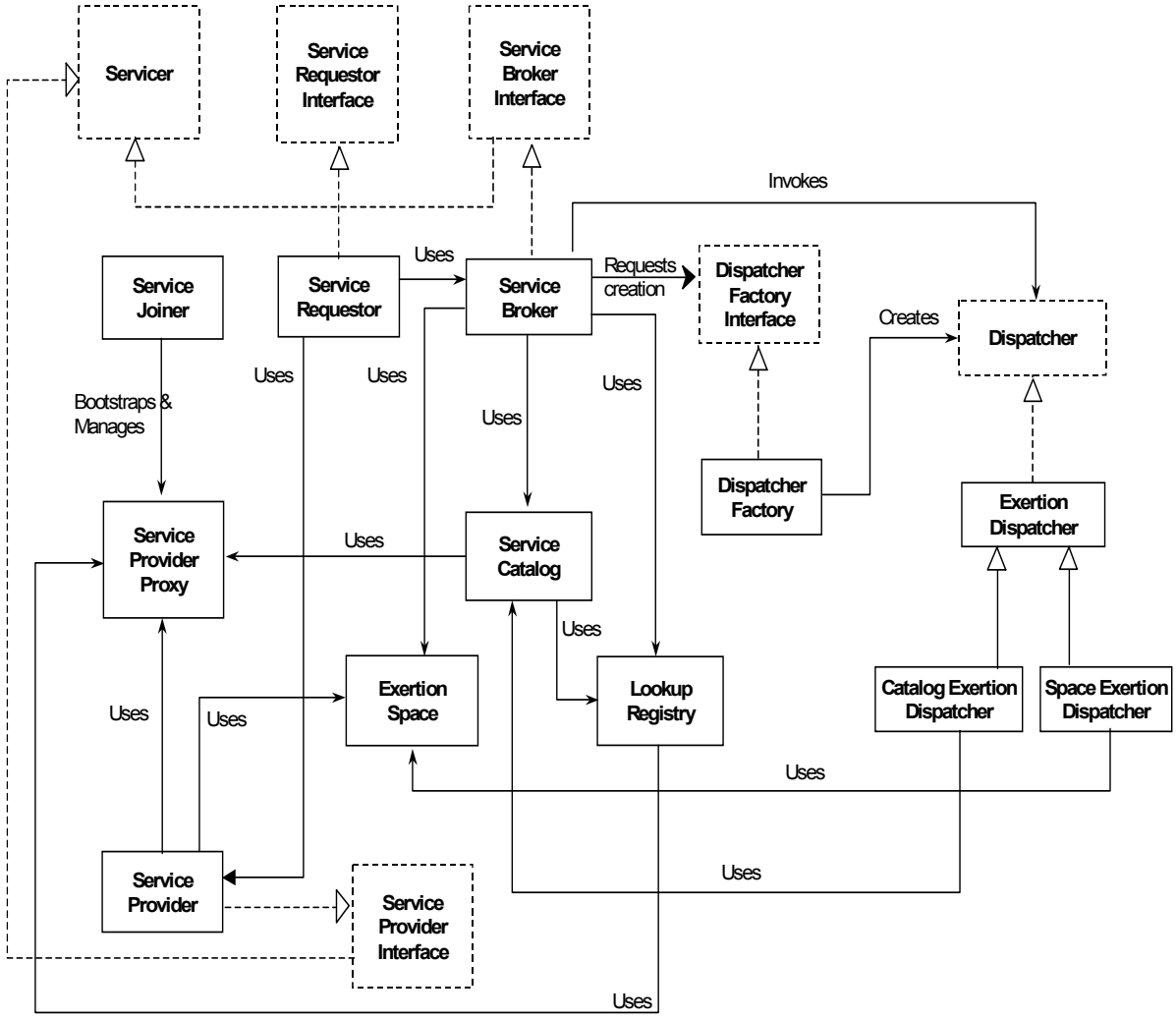

Figure 3.11 Service-based framework to support nested transactions

The service broker, by using an appropriate exertion dispatcher, can directly access the service provider through a service catalog and select a provider or drop the exertion into Exertion Space for the first available provider to process the request. While the service broker is servicing a job, a nested job within the job being currently serviced can be executed locally, or can be dropped into the Exertion Space, or passed on directly to another service broker. Another available service broker can then federate and

collaborate in the job execution by executing the nested job, and so on. Thus not only can the service providers federate to execute a job for a particular service broker, but the service brokers can also federate along with other service providers. The federated brokers with the originating broker execute the nested jobs while the regular service providers execute all the tasks within all jobs including the originating one. A service broker uses a factory design pattern to request a relevant exertion dispatcher that matches the control structure of the executed job.

Two main types of exertion dispatchers are used: a catalog exertion dispatcher and a space exertion dispatcher. The catalog exertion dispatcher finds needed service providers using the service catalog. The space exertion dispatcher drops exertion into the exertion space to be picked up by matching available service providers. When the exertion is picked up from the space and it is executed by a matching provider then the provider returns it into the space and the space exertion dispatcher gets it back from the space for the service broker. The service grid also defines a common service provider interface (*Provider* that extends the top level interface *Servicer*) and a set of utilities to create and register providers with the grid as service peers. A Service Joiner is used for bootstrapping service providers and also for maintaining leases on registered proxies. The grid service brokers can also use dynamic provisioning based on Rio technology (Rio Project) for deploying and maintaining required federation. For a direct connection to the service provider the provider can either use discovery to find a lookup service or use a Service Catalog provider for selecting a service. The lookup service caches all the proxies for services that have registered with it for a particular group(s) of services. The Catalog provider is a service-grid cache that periodically polls all relevant lookup services and

maintains a cache of all the proxies that are registered with the lookup services for a particular group or groups of services.

Thus, multiple service catalogs may be used for different logical overlay sub networks. The provider has to discover lookup services each time it needs to use them where as it finds one of required catalogs only once when it (provider) is instantiated and then the Catalog continues service discovery for the provider. In case the provider finds an available service using a lookup registry or the Catalog, a proxy for the service is downloaded on to the provider who invokes the service by calling *service (Exertion)*. Alternately the provider submits the service request to an Exertion Space that holds the request and waits for a matching service provider to accept the exertion. This is essential so that the transaction does not have to abort due to non-availability of a service. This also helps in better load balancing of the services since available providers will act at their own pace to process the exertions in the space. A notification management framework (Lapinski 2002) based on a notification provider allows federated providers notify the service requestor on their collaborative actions. Additionally the File Store provider (Sobolewski 2003) allows federated providers to share exertion input as well as output data is a uniform service-oriented way.

CHAPTER 4

SECURITY FRAMEWORK

4.1 Introduction

Security in any framework is critical. In the grid based on the Jini (Service Oriented) framework security becomes a big issue because mobile code is involved. Downloadable code poses complicated security issues. A secure grid framework shall be able to provide an environment of mutual trust between requestor and provider.

4.2 Defining trust

4.2.1 Trusted Component

A trusted component is a component which must work correctly to ensure that the security requirements of a particular framework are met.

4.2.2 Trustworthy Component

A trustworthy component is a component which actually has an acceptable level of assurance that the component actually performs its security related responsibilities.

Both the concepts seem similar. However, it is unfortunate and noteworthy that the trusted components may not be trustworthy. A simple real life example is when a person downloads a file from the internet and runs a virus scans on it. The file now becomes a trusted component, however, if the anti-virus software does not have updated definitions, the anti-virus and the downloaded file both are not trustworthy.

A mutual-trust environment has to be built so that both the requestor and providers consider each other trustworthy and shall be able to demonstrate this trust. To design a secure framework we must investigate the grey areas where security can be breached.

4.3 What is Insecure

The breach of security can occur at various points in a Service Oriented environment based on Jini Framework. In the Figure 4.1 grey areas represent the areas where the security lapse may occur when the mobile code is downloaded

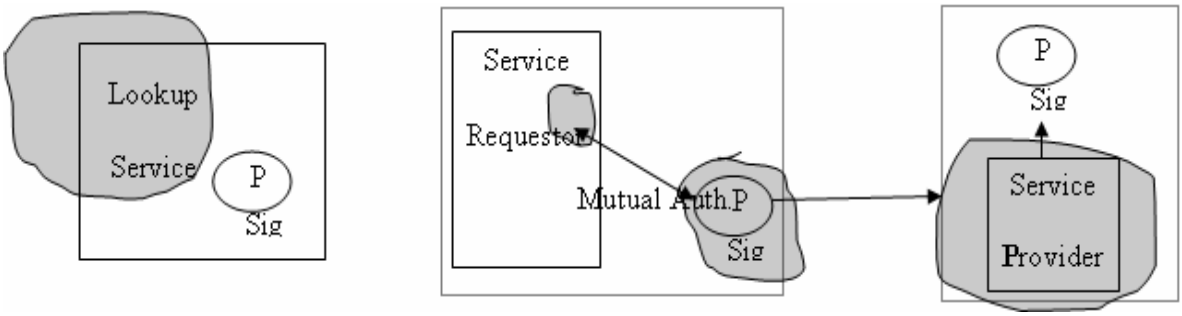

Figure 4.1 Security Considerations

4.3.2 Lookup Service

A requestor locates a service by querying a Lookup Service (LUS, service locator). LUS provides the requestor with a proxy which represents the service. The requestor calls a method on this proxy and the call is transferred to the remote service, where the method is executed.

A malicious lookup service may provide a fake proxy to the client. The client sends or uploads his data assuming that the obtained proxy is trusted. This data can be provided to any malicious user who has started the malicious look up service. Thus the client's identity and private data are at high risk

4.3.3 Service Provider

If a malicious Service Provider sends a proxy to the requestor and doesn't authenticate itself to the requestor, it can get access to requestor's data. This data can be disclosed or misused.

4.3.4 Service Requestor

Service requestor code (Service UI) may not authenticate to the Service provider. It can use requestor's credentials and impersonate the user. Also, it may generate false results without even contacting the service provider.

4.3.5 Proxy

Service proxy shall perform integrity checks so that the 'man in the middle' attacks can be avoided. If this is not done, the grid becomes susceptible to 3rd party attacks (Man in the Middle attacks). In this case, both the client and the grid resources are compromised.

Java and Jini provide API for ensuring security in such scenarios. A trust environment can be built between requestor and provider with the help of the intrinsic security mechanisms given in Chapter 1.

4.4 Authentication

Authentication or Identification refers to the process of establishig trust about the entity which is trying to use the services. Java provides Java API for Authentication and Authorization (JAAS) API to be able to achieve this. Although the main challenge in the case of Service Oriented Grid Computing is that authentication needs to be done using downloaded code.

4.4.1 JAAS

JAAS basically grants permission based on who is executing the code instead of granting permissions based on the location of the code (downloaded / local). This model is similar to the authentication and authorization model used in Unix, Windows and other operating systems.

JAAS uses Pluggable Authentication Modules (PAM) for authentication. The mechanism is pluggable because it allows an authentication mechanism to be dropped into the code environment and which is then used to authenticate users running the java code [14]. This pluggable authentication mechanism can be either a Graphical User Interface (GUI) asking for username and password or a GUI asking for Java card or some other device which contains user's credentials. If the authentication succeeds, the user is given access to that particular code (depending upon the privileges the Subject of the user

is allowed to have). Different modules can be plugged in, including Kerberos and PKI implementations. With the advent of JDK 1.4, JAAS been added to core java APIs.

4.4.2 JAAS Framework for Authentication [36]

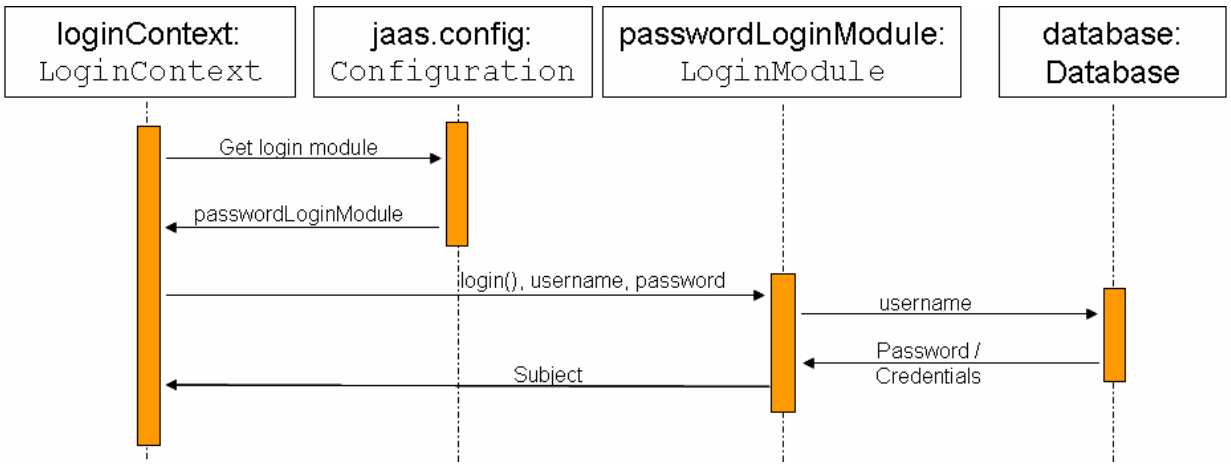

Figure 4.2 Authentication using JAAS

4.5 Authorization

4.5.1 Subject (doAs() and doAsprivileged)

As defined already, a Subject defines a group of related information, for a single entity such as a person. This information can include the person's (Subject's) identifications and security related information such as his passwords, security codes etc. Authorization requires that the persons identity be known and checked for access[14].

The `doAs()` and `doAsPrivileged()` methods in the Subject class asks the code to run as the Subject and this allows the AccessController to check if the particular subject (logged in) has the authorization to run that piece of code.

```
doAs(Subject subject, java.security.PrivilegedAction
     action)

doAs(Subject subject,
          java.security.PrivilegedExceptionAction
action)

doAsPrivileged(Subject subject,
          java.security.PrivilegedAction action,
               java.security.AccessControlContext acc)

doAsPrivileged(Subject subject,
          java.security.PrivilegedExceptionAction
action,
               java.security.AccessControlContext acc)
```

The `doAs()` method This method first retrieves the current Thread's `AccessControlContext` via `AccessController.getContext`, and then instantiates a new `AccessControlContext` using the retrieved context along with a new `SubjectDomainCombiner` (constructed using the provided `Subject`). Finally, this method invokes `AccessController.doPrivileged`, passing it the provided `PrivilegedExceptionAction`, as well as the newly constructed `AccessControlContext`[25].

The `doAsPrivileged()` method behaves exactly as `Subject.doAs`, except that instead of retrieving the current Thread's `AccessControlContext`, it uses the provided `AccessControlContext`. If the provided `AccessControlContext` is

`null`, this method instantiates a new `AccessControlContext` with an empty collection of `ProtectionDomains`[25].

4.5.2 Guarded Objects

Gaurded Objects [39] are represented by the Class `java.security.GaurdedObject`. These objects are used to protect access to other objects. The `getObject()` method provides access control to the protected object. This is done by invoking checkGaurd method of `java.security.Guard Class`. If the access is allowed, the reference is returned, otherwise, `SecurityException` is thrown.

4.5.3 Permissions/ Policy Objects

The version 2.0 (Davis Release) of Jini has provided a wide variety of API for enabling security in framework based on Jini. Following additions have been made in Jini 2.0 from the previous versions of Jini [38] [32] :

- Configuration: Mechanisms for building and deploying configurable and secure services
- Exporter: Exporter provides abstractions for exporting the remote objects and obtaining the server side information for executing the remote call
- Proxy Preparation: The feature of preparing a proxy before a remote call enables the requestor to put constraints on the proxy to ensure communication in a trusted environment.

The Jini API used for building the desired security framework has been outlined in the respective sub-headings given below.

4.6 Invocation Constraints

The security considerations act as *constraints* on normal execution: something that might have been allowed will be restricted after the constraints have been set. For example, a requestor may put the constraint that communications be encrypted (`Confidentiality.YES`). It might not want to know many details of how this has been done (This depends on middleware which can be either the standard or customized based on requirements). But if it hasn't been done, then the requestor will not accept the communication [11].

Jini 2.0 doesn't specify how the constraint is implemented but just what it is. However, in the Jini's default implementation of server endpoints such as `SslServerEndpoint` this has been predefined. An explanation of these constraints has been given below:

4.6.1 Integrity

Integrity constraints are the constraints to enforce Integrity during the communication. `Integrity.YES` enforces detection of alteration of the message by third parties and if detected, the communication is refused and an exception is thrown. `Integrity.NO` constraint is put when the detection is not necessary

4.6.2 Confidentiality

Confidentiality constraints are set to check whether third party listeners can interpret the communication. If `Confidentiality.YES` constraint is specified, the message is bound to be transimitted so that it can not be easily interpreted by third party listeners. If `Confidentiality.NO` constraint is specified, the message is transmited without any use of encryption

4.6.3 ClientAuthentication

`ClientAuthentication.YES` constraint specifies that the requestor shall authenticate to the server. Again it is to be noted that the constraint doesn't specify how the requestor should authenticate to the server. This is implementation dependent. Jini's standard `SslServerEndpoint`, `KerberosServerEndpoint` etc. have their own mechanisms of satisfying this constraint. For example, `SslServerEndpoint` uses the requestor's certificate (with the private credentials) for ClientAuthentication.

`ClientAuthentication.NO` constraint instructs the requestor not to authenticate to the server. This means that the requestor refuses to prove his identity. This is important in the applications where the requestors need to preserve their privacy

4.6.4 ServerAuthentication

`ServerAuthentication.YES` constraint enforces the server to authenticate to the requestor. In many applications the requestor needs to know whether it is dealing with the right service provider.

`ServerAuthentication.NO` constraint enforces the server to remain anonymous.

4.6.5 Delegation

`Delegation.YES` constraint allows the server to authenticate as requestor in remote calls made and received by the server. This is important especially in the cases of a grid technology where one service provider sends a task across to another service provider (third service provider) on behalf of the requestor. Since the requestor's identity may matter to the third service provider, for example for Authorization. Jini's `SslServerEndpoint` does not support direct delegation, since by design SSL doesn't allow requestor's private credentials to be passed. However, `KerbersoServerEndpoint` does support Delegation.

`Delegation.NO` constraint can be set to disallow the server from impersonating the client while talking to a third party.

There are other constraints which control which principals of the server and requestor shall be authenticated and which principals shall not be [16]:

`ClientMinPrincipal`

`ClientMaxPrincipal`

`ClientMinPrincipalType`

`ClientMaxPrincipalType`

`ServerMinPrincipal`

However, there is no `ServerMaxPrincipal` Constraint since requestor shall not need to impose such a constraint

4.7 Proxy-Trust (Proxy-Verification)

The privilege of proxy trust is very important in the frameworks which incorporate mobile code. The requestor must be able to deicide whether it has received a proxy from the right service and also whether this proxy does the right thing. For example if the requestor debits $100 from a bank service how will he trust whether the proxy doesn't debit $1000 from his account and deposit the remaining $1000 somewhere else. The requestor shall be able to explicitly decide if it trusts the proxy especially for communications where security is important.

There are two different cases where proxy trust needs to established:

4.7.1 Local Code

If all parts of the proxy code are local then the client can decide to trust the proxy. Effectively, since the client has established that the proxy code is not downloaded code, he is using his local code which he can trust.

4.7.2 Downloaded Code

The main cause of concern comes when the proxy code is downloaded and the client needs to establish whether the proxy can be trusted. One easy way is to go the service provider and ask the service if it trusts the proxy that the requestor has received.

Note that the integrity of the downloaded code has already bean established in the previous section. So now the "trust" that the downloaded code is not changed by anyone is already in place

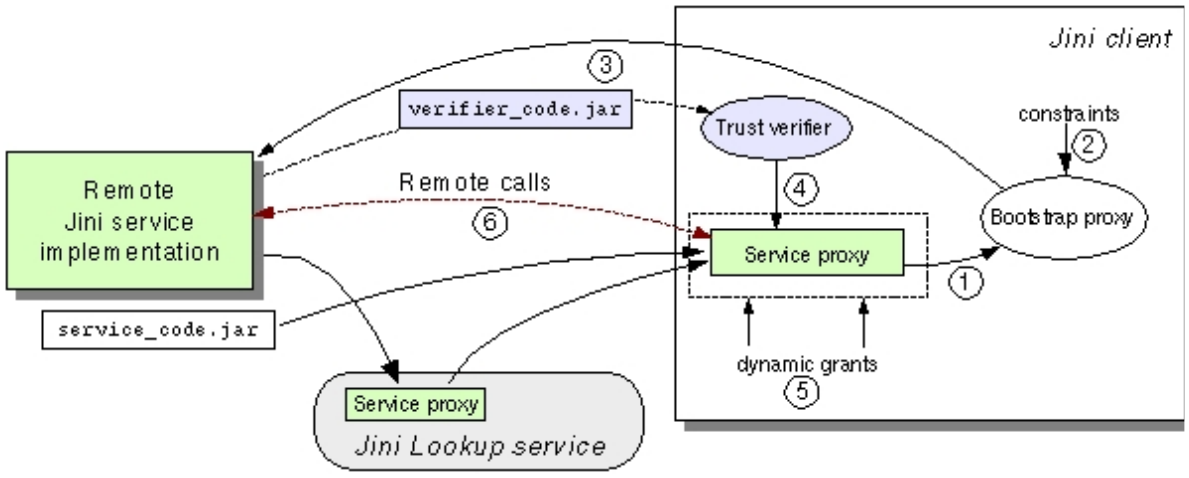

Figure 4.3 Proxy Trust Verification Mechanism [17]

Basically, we are dealing with transitivity here where the requestor trusts the service, if it can establish that the service trusts the proxy then it can also trust the proxy.

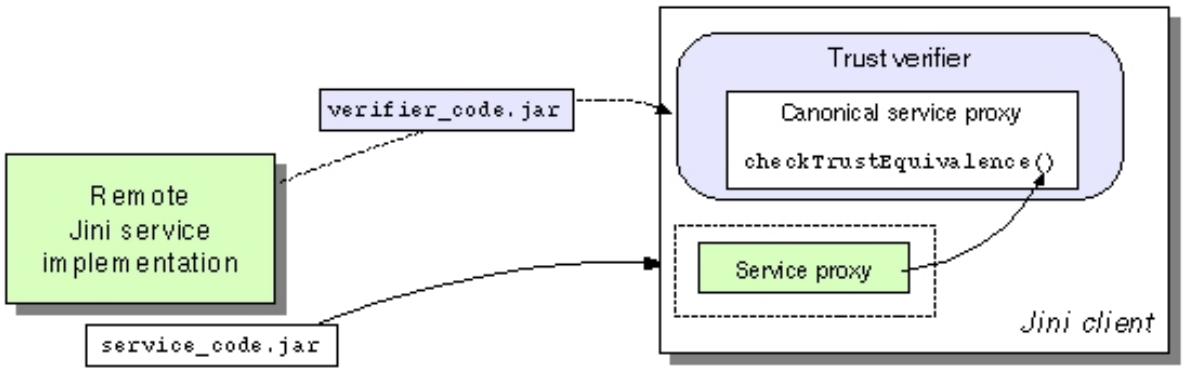

Figure 4.4 Determining Trust Equivalence

However, this is where the "chicken and egg" nature of the problem comes into picture. Since the requestor can contact the service only through a proxy, how can the requestor contact the service via a proxy it doesn't trust (as of now). This is a problem which needs bootstrapping solution. The client uses a bootstrap proxy to verify the trust

of the downloaded proxy. The mechanism (Figure 4.3 and Figure 4.4) used for proxy verification using a bootstrap proxy has been given below:

The requestor asks the downloaded (untrusted) proxy for a bootstrap proxy. After obtaining the bootstrap proxy, it ensures that all the code in the bootstrap proxy can be resolved locally. This is done since the client can explicitly trust only the local classes.

After obtaining a bootstrap proxy, the requestor shall then put `ServerAuthentication` constraint which would enforce the service to authenticate itself to the requestor. Once the service is authenticated, the requestor trusts the service and now can ask the service if it trusts the downloaded proxy.

Requestor now invokes `getProxyVerifier()` method on S which returns a `TrustVerifier` object which it then uses to decide if the service proxy is trusted.

Requestor calls `isTrustedObject(Object ServiceProxy, …)` to verify whether the downloaded service proxy can be trusted. The algorithm which the `TrustVerifier` uses to verify whether the service can trust the service proxy, is dependente upon the service itself. Jini's default algorithm relies on object equality semantics. The `TrustVeirifier` returned by the service contains a canonical service proxy instance. The `isTrustedObject`() method compares the canonical instance and the downloaded instance and if the objects match then the downloaded proxy is considered trusted.

if `isTrustedObject`() returns true, the service proxy is supposed to be completely trusted by the (trusted) service. The requestor and service provider are now free to communicate since the trust of the downloaded code is established.

The Figure 4.5 below gives a pictorial representation of smart proxy verification procedure in Jini framework

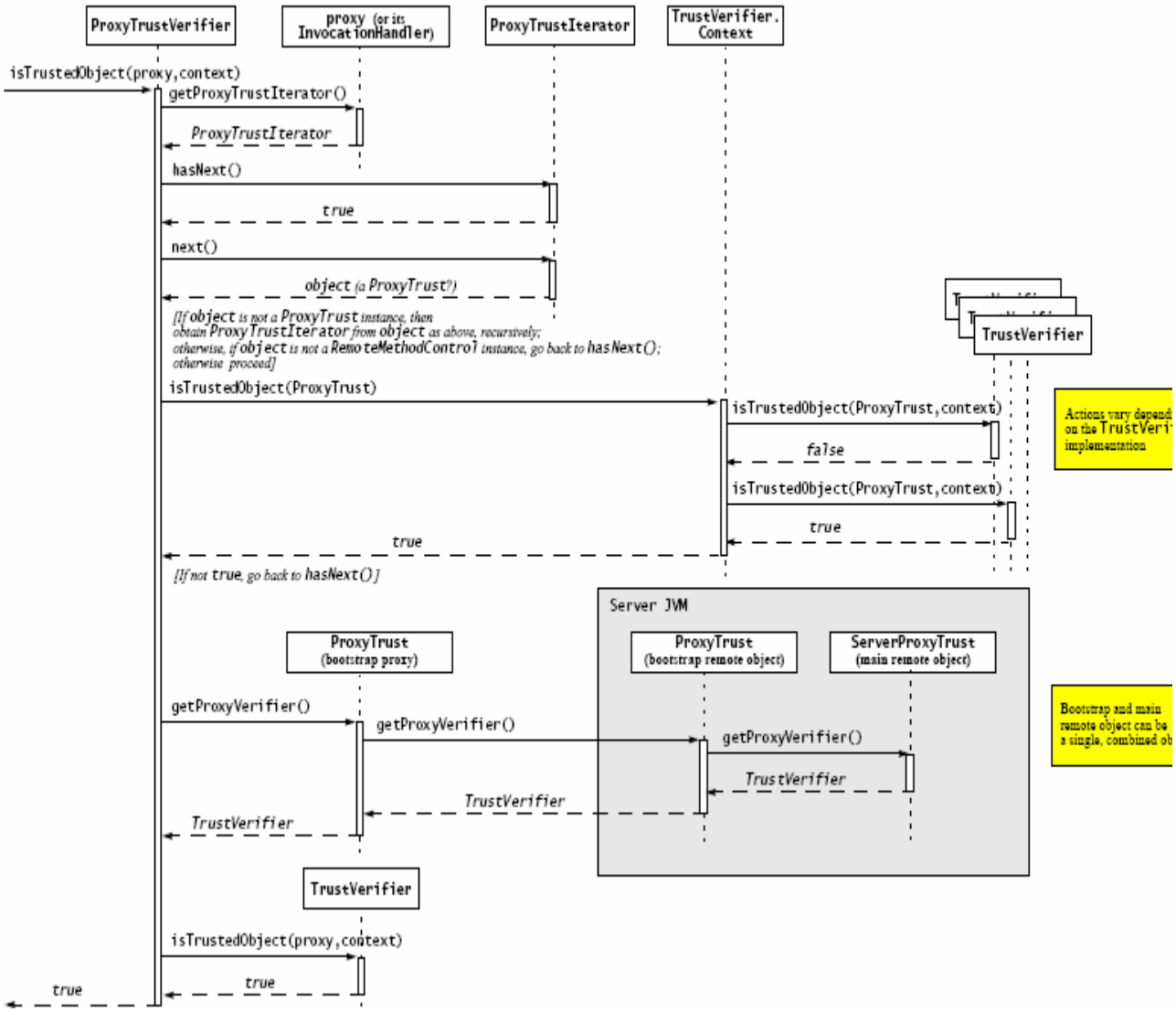

Figure 4.5 Smart Proxy Trust Verification [18]

However, these steps are mostly hidden from the developer. For example, `SslServerEndpoint` automatically takes care of these constraints and this mechanism on setting a flag while preparing the proxy. However, this will be discussed further in the chapter where implementation and validation has been discussed.

4.8 Integrity

Integrity is needed to detect and take measures in case of "Man in the middle" or third party attacks. For example if the requestor sends a message to the bang service to 'withdraw $100', and someone in the middle change the message content to 'withdraw $150 and transfer $50 to Alice". The banking service shall be able to detect this intrusion and act accordingly. Therefore, Integrity is important in two cases – when the code is downloaded and when the communication is taking place.

4.8.1 Downloaded Code

In case of downloaded code, the requestor must be able to check that the code it has downloaded (from a particular URI) was not changed by a third party. For example a malicious user 'A' can change the jar file loaded at the rmic-codebase of a service. When the requestor downloads this file assuming it to be the file pointed to by the proxy of the service, it is at very high risk of compromising sensitive data and even its identity.

Usually, the URIs are resolved to HTTP URLS. HTTP doesn't provide any intrinsic methods to provide integrity [16]. A conceivable solution is HTTPS URLs, which would guaranty confidentiality and server authentication. However there are a few basic disadvantages in using HTTPS in this particular case:

- HTTPS server would require a Private Key for authentication, Similarly, the requestor would require his own private key. Trust-stores would also be required. The whole mechanism is an extra overhead attached to the whole process.
- HTTPS also provides encryption although all the requestor may care about is just integrity.

Jini provides with an easy solution to provide integrity of the downloaded code without requiring overhead of encryption. It provides a new URL scheme type called HTTPMD where MD stands for message digest. So the server now needs to calculate the message digest of all its jar files and specify it's codebase as a list of HTTPMD URLs. An example of HTTPMD URL is:

httpmd://neem.cs.ttu.edu:2044/classes/dispatcher-dl.jar;md5=40c8812dce7b9f8fb0a3b364af

However, this scheme is introduced with jar files only and not with directories and class files on the server.

Once the code is downloaded its Message Digest is calculated using various classes provided in the `net.jini.url.httpmd` package. If the two message digests match, the requestor trusts the integrity of the downloaded code (however, it may not yet trust the service provider), otherwise `WrongMessageDigestException` is thrown.

One important thing to note about HTTPMD URL scheme is that it is an non-standard scheme and hence not recognized by standard Java Virtual Machine. The HTTPMD handler is a part of Jini package and hence it needs to be installed on JVM by specifying `-Djava.protocol.handler.pkgs=net.jini.url` property. It is

better to start both the services as well as the requestor with this property for smooth functioning.

4.8.2 Communication Integrity

Communication Integrity is required for remote calls between the requestor and the service provider. For this, the server shall be started with SSL, HTTPS, Kerberos or an custom endpoint which supports integrity. Protocols like TCP don't support integrity. The `Integrity.YES` constraint shall be set on the proxy at least from either the server or the client side to be able to ensure that the transaction actually takes place securely (integrity enabled). If for example the service provider uses TCP endpoint and the client specifies `Integrity.YES` constraint, `net.jini.io.UnsupportedConstraintException` will be thrown at the client side.

4.9 Privacy

Privacy is same as confidentiality. It basically means that the communication between the requestor and service shall not be understandable by a third party. This is achieved by encryption. Encryption can be symmetric or asymmetric, this is specific to implementation. In Jini, client or the server (preferably client) can enforce confidentiality by injecting `Confidentiality.YES` invocation constraint on the proxy.

Jini doesn't specify how this constraint shall be specified and hence this specification can be customized by the developer. However, Jini's default classes such as

SslServerEndpoint do have inbuilt mechanism to ensure confidentiality, given this constraint.

4.10 Non-Repudiation

Non–repudiation as the name suggests, means that requestor or service provider shall not be able to repudiate a communication once the message is sent. This is ensured when each entity signs the message it sends. Signature of an entity is the message's message digest signed by its private key.

4.11 Accountability/Auditing

Accounting or Auditing means recording all critical communications between Requestor and Services on the grid. The critical communications may include:

- Authenitcation communication (Who tried to log in at what time)
- File update/delete/execute request
- Trial for Un-Authorized access (whenever `AccessException` is thrown)
- Whenever a Security Exception is thrown at the provider or the requestor side
- Other messages which may be critical specific to application design and implementation

This will help the administrators to locate the cause, if a security breach occurs

CHAPTER 5

SECURE FRAMEWORK AND VALIDATION

This chapter will cover the Security framework proposed for Service Oriented Computing Environment and the validation of the proposed framework in SORCER. I have chosen my validation case as SGrid, the computational grid framework created in SORCER.

5.1 SGrid – Introduction

As the exertions (jobs, tasks) are submitted via a service provider, synchronous or asynchronous mechanism can be used to send the jobs from the service providers to callers (explained below) and taskers. In my validation I have secured the channel where the jobs are sent via synchronous mode (jobber-cataloger) to callers.

In SGrid or SORCER in general there are three types of brokers which provide synchronous and asynchronous modes of exertion execution. These three brokers are (refer Figure 5.1):

- Jobber
- Cataloger
- Exertion Space

5.1.1 Jobber (Coordination broker)

Jobber is a service provider in SORCER framework which provides the services of a coordination broker. Jobber can be involved in both synchronous and asynchronous

modes of exertion execution. Figures 5.1 and 5.2 depict the functioning of the various brokers in the synchronous and asynchronous modes.

This is further explained in the sub-sections below.

5.1.2 Cataloger (Synchronous broker)

Cataloger is the Synchronous broker. It is called synchronous because proxy for a particular provider (Caller in this case) can be obtained and the exertion can be sent to it either directly or via a jobber (coordinated). Hence the exertion is executed in a synchronous way. The direct and coordinated synchronous approaches have been shown in the figure.

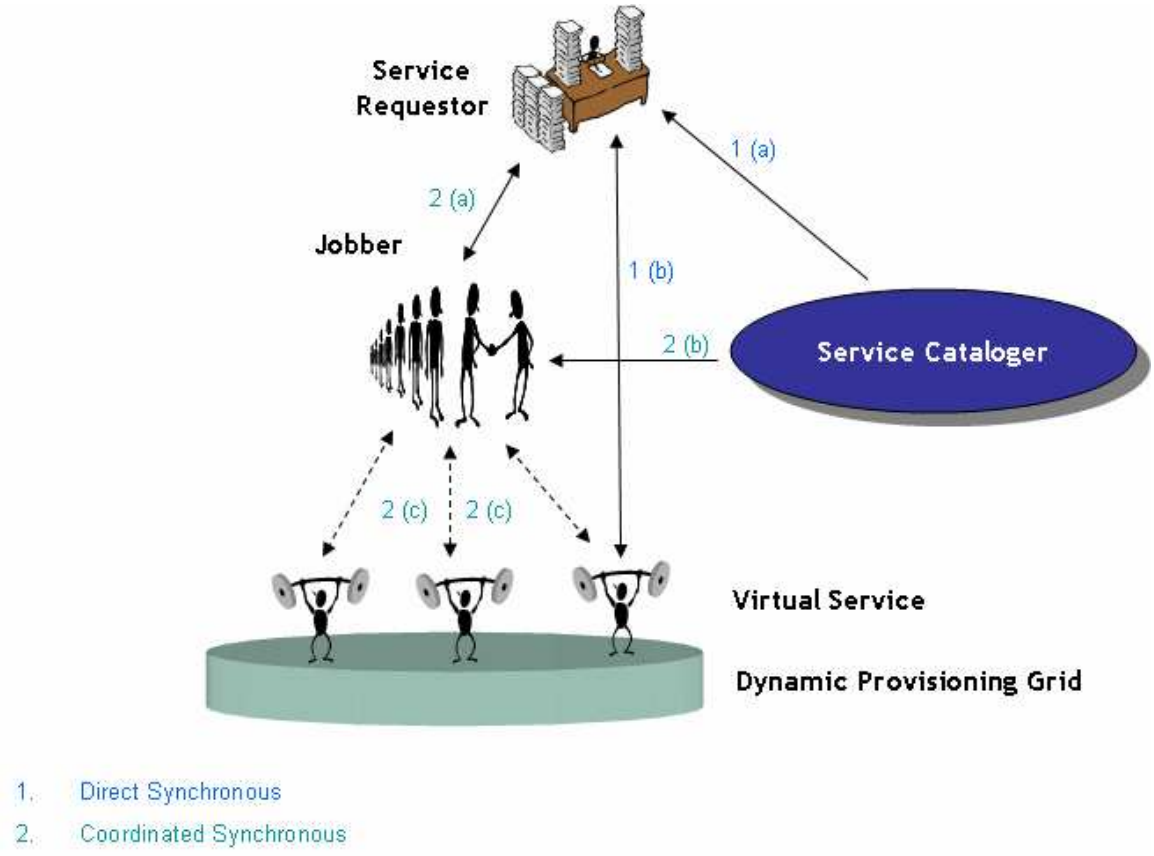

Figure 5.1 Synchronous Exertion Execution in SORCER

5.1.3 Exertion Space (Asynchronous broker)

Exertion space is the asynchronous broker. The asynchronous behavior is obtained by applying the pull technology. Either jobber (coordination) or the service provider can place exertions in the space. The Callers are listening to the space and when an exertion is put in the space, it is automatically taken by of the callers. Since any of the callers can take the job from the space, the execution is asynchronous. A caller with more amount of processing power (or network connectivity) is the one who is more probable of taking the exertion from the space. Hence this does bring some load balancing in the system. The asynchronous approach has been depicted in the Figure 5.2.

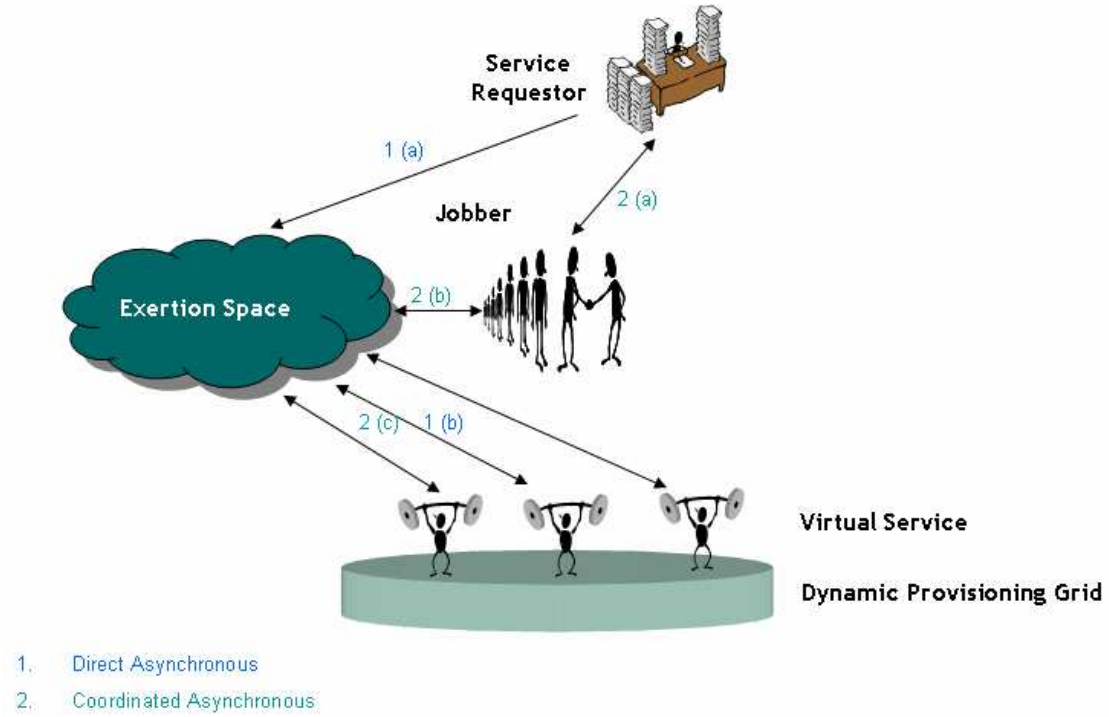

Figure 5.2 Synchronous Exertion Execution in SORCER

Presently, SORCER exertion space is same as JavaSpaces used in the Jini Framework. However, as mentioned in the Future work section, using and developing secure exertion spaces will surely be an important task in future. Since the prime objective of a JavaSpace is to be publicly available, this task becomes substantially difficult.

If the secure framework cannot control who is putting the jobs in the exertion space and who is taking the job form it, the whole framework can easily be compromised.

Since securing exertion space is a big problem in itself and it was outside the scope of my thesis problem, hence, I have used "synchronous" channel of job execution which uses the Cataloger (Synchronous Broker) for the job execution.

5.2 S-Grid Components

5.2.1 SGrid Dispatcher

SGrid Dispatcher is the service provider which publishes a Service UI which lets the requestor enter the inputs, outputs, arguments and the application name etc for the job. Dispatcher then encapsulates it into a caller context and sends it to the Jobber.

The Service UI of the dispatcher looks like the one give below:

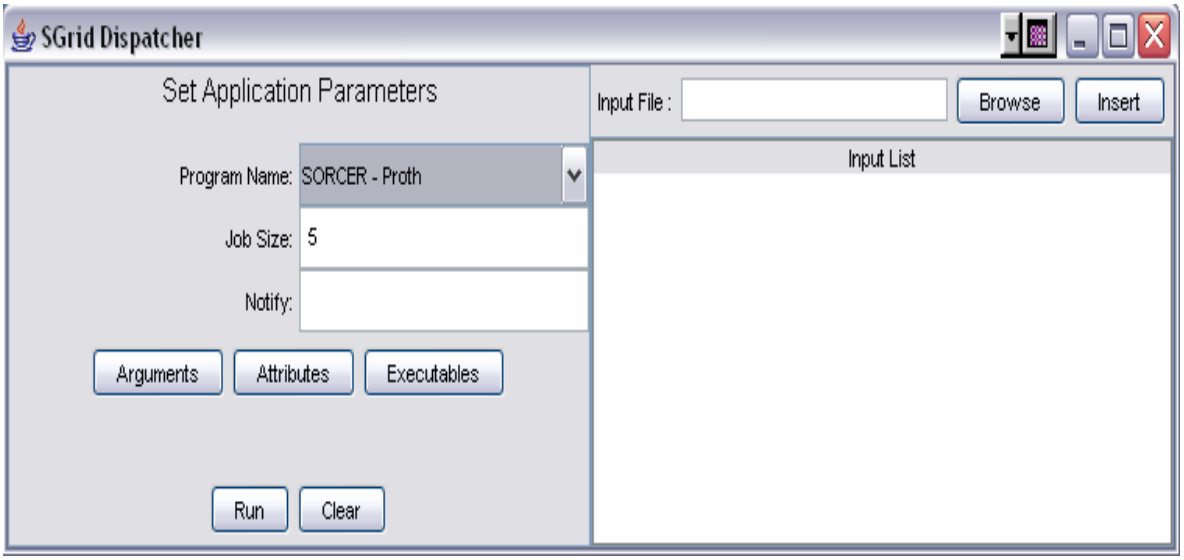

Figure 5.3 SGrid Dispatcher Service UI

The Arguments tab lets the requestor specify the command line arguments to be specified to the executable when the Caller executes it.

The Attributes tab gives the user the flexibility to choose a platform, host or domain where he wants his inputs to be dispatched and executed. By default, the user's request is sent to any compute resource in the grid which is ready to process the job.

From the Executables tab the user can specify the executables (for different platforms) and their locations (from where Caller can download them if needed).

5.2.2 Caller

Callers are generic SORCER service providers which make system calls. These callers are platform independent service providers and have the capability of downloading (using Filestore [34]) required binaries (based on the OS) or source code required to make the system call. Callers can also run a java code and send the results either to the requestor or store them somewhere as specified in Caller context. The Figure 5.4 provides a pictorial idea of caller context.

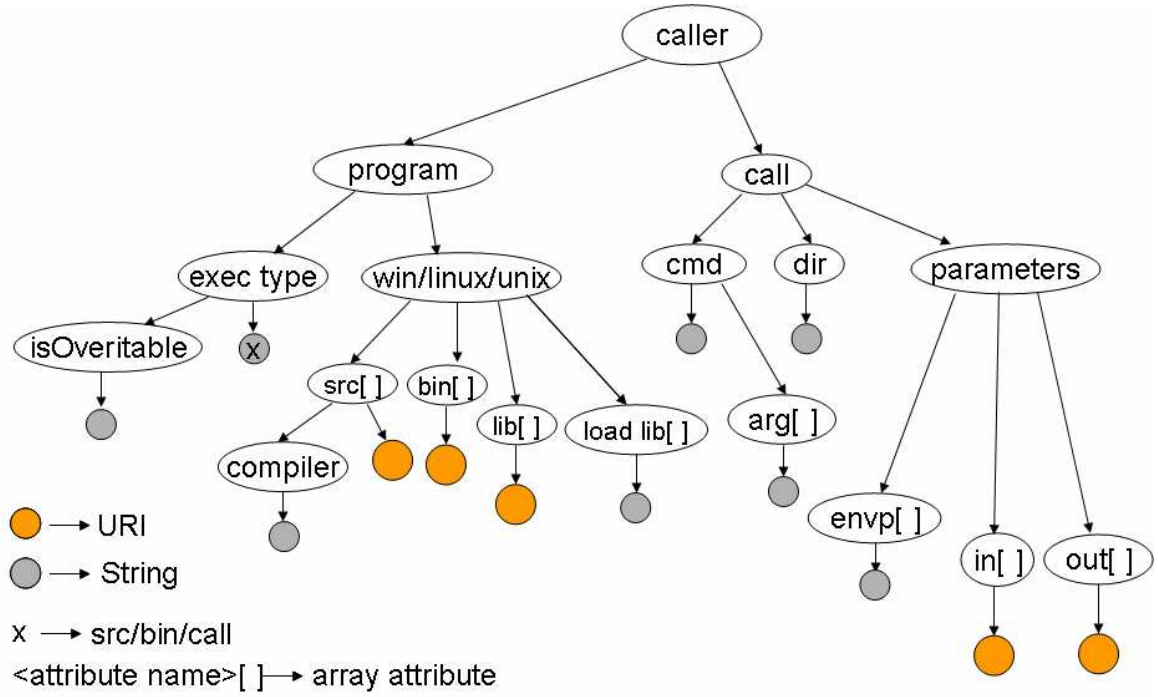

Figure 5.4 Caller Context

5.2.3 File Store

Caller has the capability of downloading binaries or source files with the help of FileStore. FileStore is a SORCER service provider which allows users and other providers to upload and download required files from the database.

5.3 Security Framework and Implementation

5.3.1 Proposed Framework

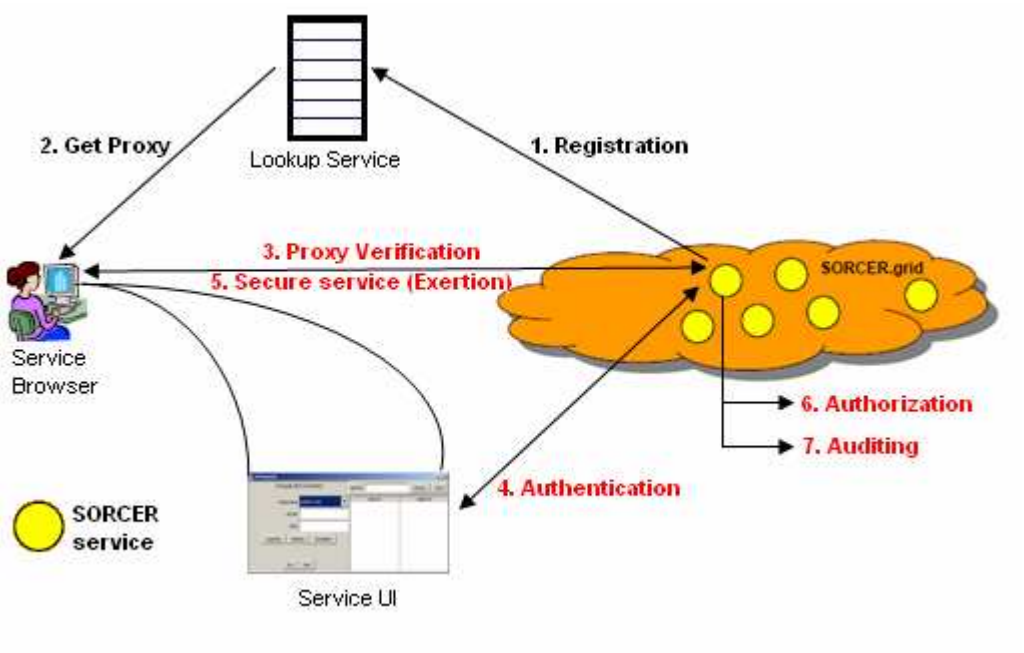

Figure 5.5 Security Framework

5.3.2 Use of Cataloger

Since the security has been built on the synchronous mode of job execution in SORCER, Cataloger has been used. Cataloger is used by Jobber to get a proxy for the Caller before sending the jobs to the caller.

5.3.3 Ensuring Security

The proposed framework ensures a trust-environment between the requestor and SORCER providers. All the SORCER providers are to be started using secure endpoints such as SSL, Kerberos etc. Following Security requirements have been built in SORCER:

- Authentication
- Authorization (and Delegation)
- Proxy Verification (Trusting the downloaded code)
- Integrity
- Auditing
- Security wrapper for executing legacy code on Caller

5.3.4 Security Wrapper for Service UI

A security wrapper for Service UI is needed so that just by subcalssing this wrapper, any service UI inherits all the embedded security automatically. This will ensure that each service provider will not have to think about writing security for its own Service UI. This will help in less security patch-ups since the security will be provided by default.

This security wrapper should be able to perform the following:

- Authenticate user
- Verify proxy
- Set Integrity constraint (avoid man in the middle attack)
- Set Mutual Authentication constraint (to trust the service)

- Set Confidentiality constraint (to ensure privacy) etc.

The secure wrapper should be able to abort the communication in case the service provider doesn't satisfy these constraints. Also it shall be able to send this information (failed attempt) to the Auditor so that it can log the information accordingly.

5.3.5 Security wrapper for Caller

The security wrapper for Caller is needed. Since the legacy code (or application) may be in any language and Caller is supposed to execute system calls, if the executables consist of virus code (changed by a third party), it may result in disaster for the grid resources. So the caller shall make a system call only for those files (executables) whose integrity it can trust. In other words, caller shall have a mechanism to check whether the downloaded executable from the database has been changed by a third party or not.

5.3.6 Auditor

An Auditor is required to audit critical communications between requestors and Service Providers. This is a passive means of ensuring security. In case, the system is compromised, the audits can be checked for where and when the security breach actually occurred. Regular checks of the logs will help in improving security and tracking the fault as and when an attack on system security is performed.

5.3.7 Package Diagram

The package hierarchies and dependencies are shown as a UML diagram in the Figure 5.6. Since Intrinsic Security Framework is intrinsic to the SORCER existing

framework, very few packages were actually added to the default SORCER packages. The (top level) packages that are critical the SGrid framework are:

- jgapp.jaas
- sorcer.base
- sorcer.core
- sorcer.core.provider.jobber
- sorcer.core.provider.caller
- sorcer.core.provider.catalog
- sorcer.core.providor.auditor
- sorcer.core.security.ui
- sorcer.provider.grid.dispatcher

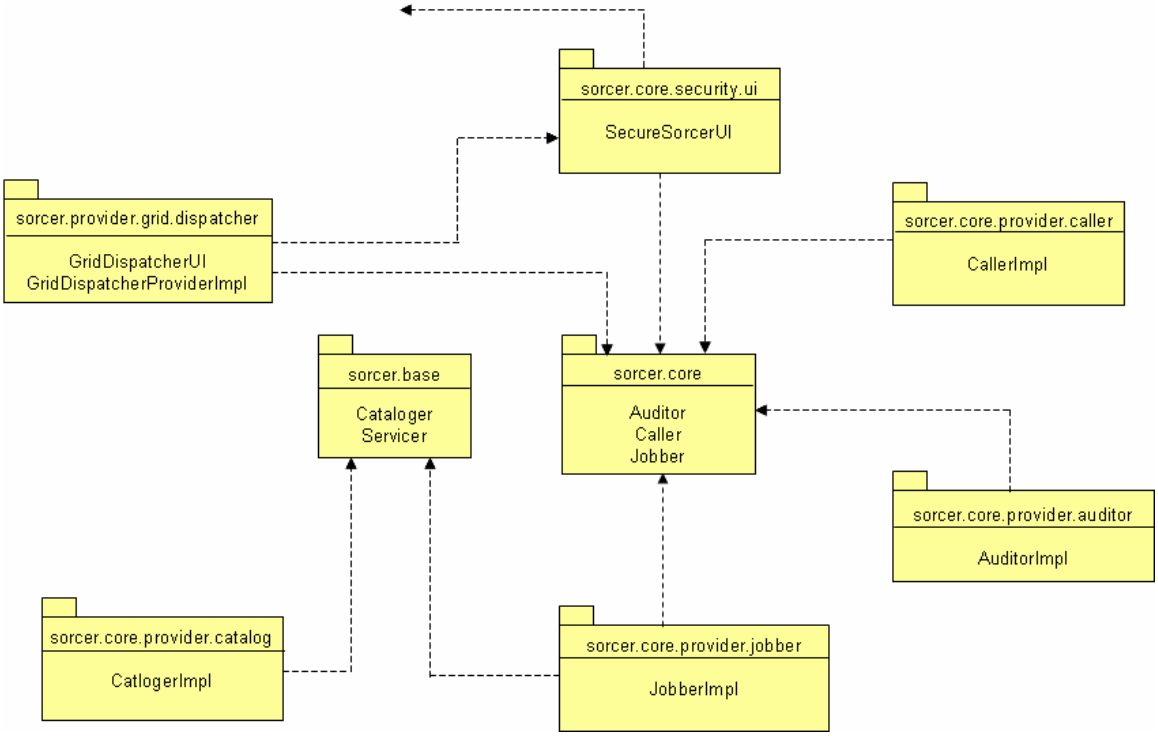

Figure 5.6 Package Diagram

5.3.8 Use Cases

The framework's objectives have been classified as use cases. These use cases are used to demonstrate the working of the security framework built upon current version of SORCER [35].

Accessing Services

This use case (Figure 5.7) shows how the access controlled services are being used in the framework. The numbered steps show the actions being performed in the respective order.

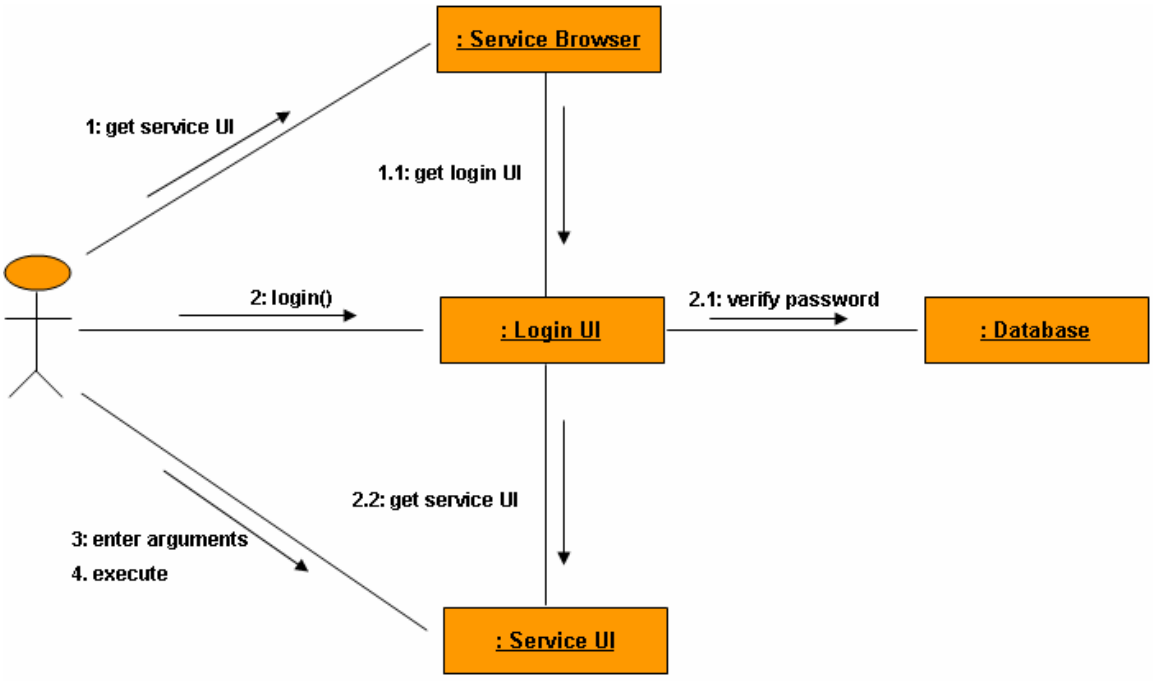

Figure 5.7 Use Case: Accessing Services

Secure SGrid Providers

The Secure SGrid Providers (Figure 5.8) use case scenario shows the two way security on the network. If the client requires security, but the service provider does not support security constraints (for example TCPEndpoints) then the transaction is not completed (action 4.2 in Figure 5.8).

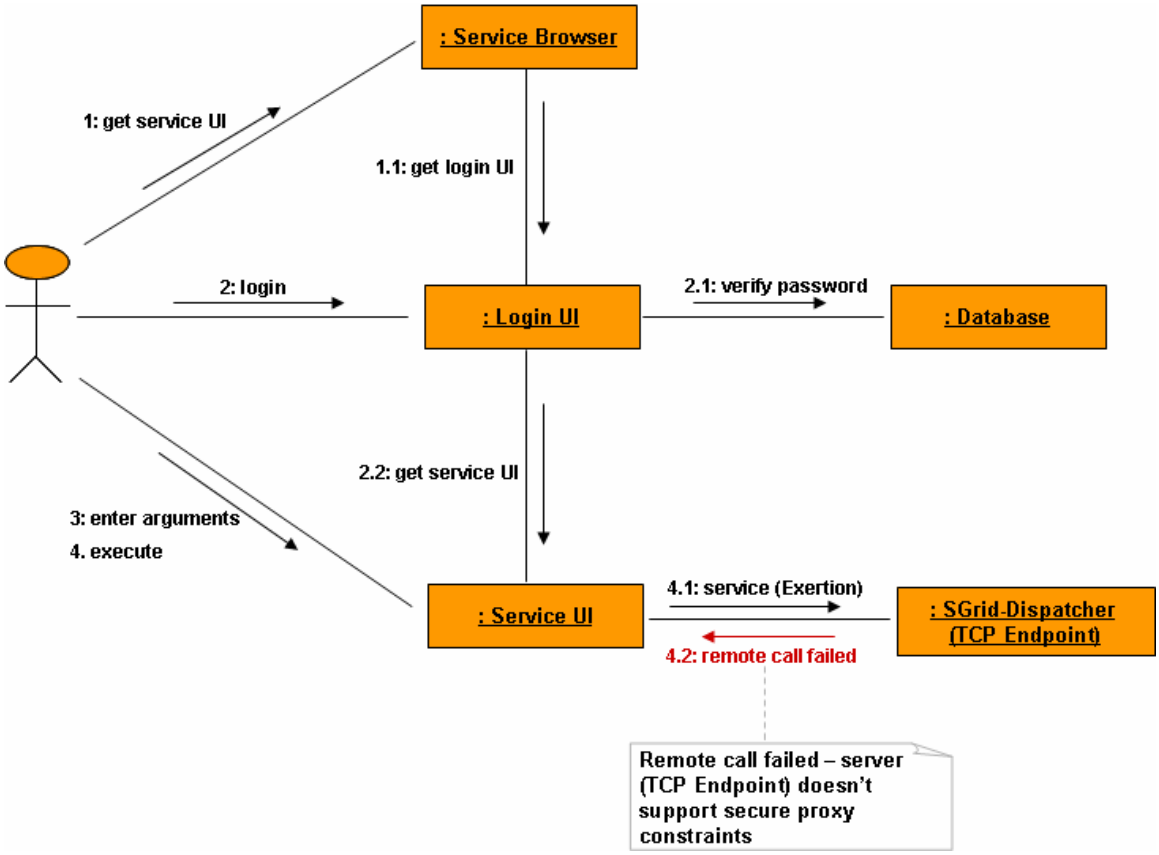

Figure 5.8 Use Case: Secure S-Grid Providers

SGrid Authorization

Authorization is one of the objectives of the secure framework. In Figure 5.9 an anonymous user logs in. This user doesn't have the proper authorization to access the remote method called on the service provider. Using the policy files we can define method level authorization based on the type of user.

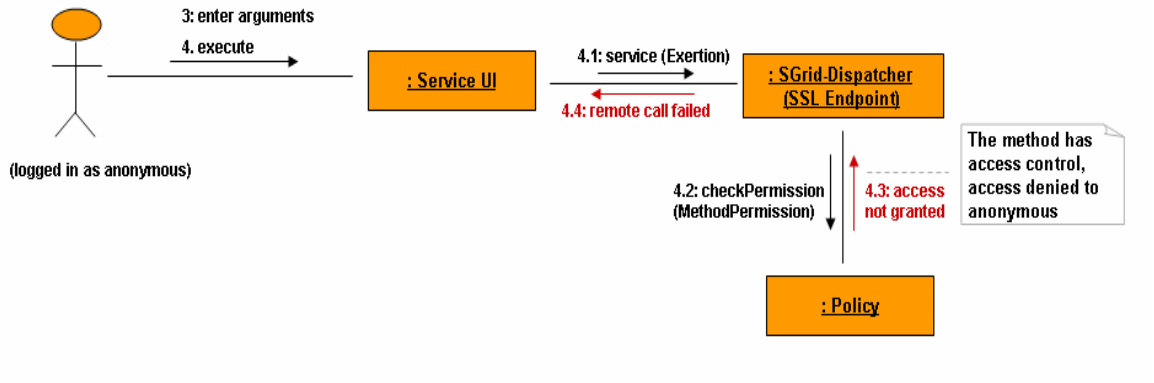

Figure 5.9 Use Case: SGrid Authorization – Access Denied

In the Figure 5.10 the user is Abhijit. This user is authorized to access the remote method on the provider and hence the remote call is allowed to go forward (to Jobber).

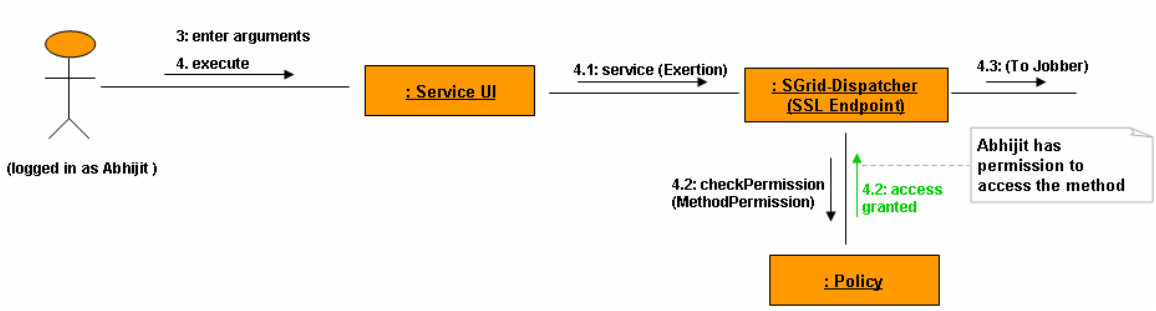

Figure 5.10 Use Case: SGrid Authorization – Access Granted

Executing Legacy code

Executing legacy code is also important to provide a security wrapper for the Caller since it executes system calls. The Caller first checks whether the obtained executable file has the same hash value as is given in its records, if the file has not been modified as per the information available to the caller, the execution is allowed (Figure 5.12) otherwise execution is failed (Figure 5.11).

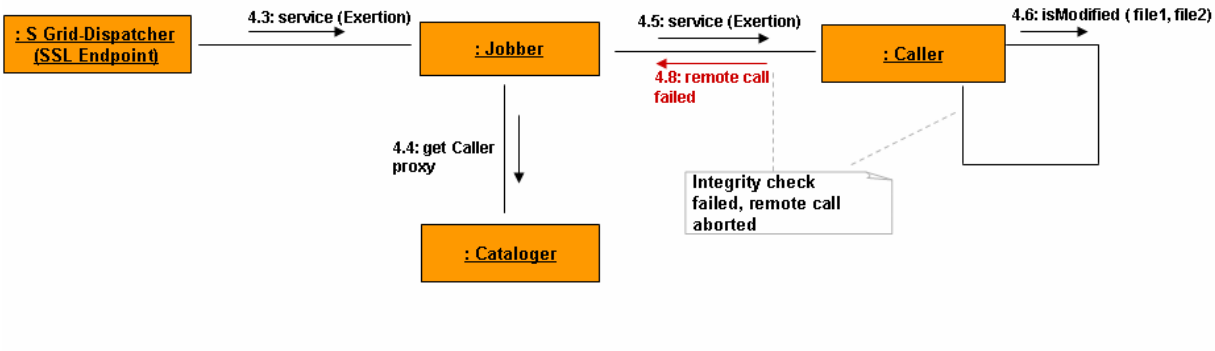

Figure 5.11 Use Case: Executing Legacy Code – File Modified

Figure 5.12 shows the flow when the remote call is successful at the Caller. The green arrow indicates that the result is finally being sent to the Requestor

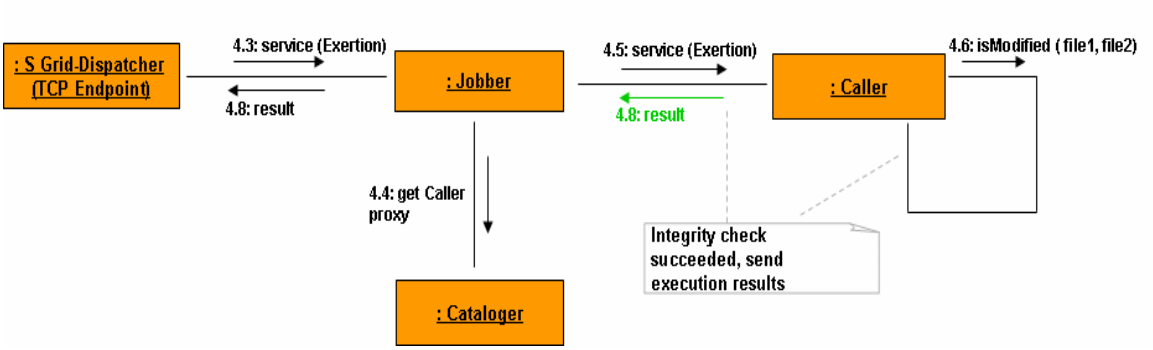

Figure 5.12 Use Case: Executing Legacy Code – File not modified

All the above usecases when combined form the total functioning of SGrid. The basic flow being:

- Login Service UI prompts the requestor to "log in"
- Only when the user is logged in, user is shown the Service UI pertaining to the provider (GridDispatcherProvider in this case).
- After inputting the values in the Service UI, when user clicks on "run", the values are passed to the provider.
- The provider checks for the user authorization, creates the CallerContext and sends the Job to the Jobber
- Jobber create jobs or tasks according to the strategy provided by the user and dispatches the jobs to the Caller
- The Caller checks the integrity of the "executable" (legacy code) and then performs the required execution and returns the result.

All the above use-cases can be represented by one class diagram as provided below (Figure 5.13):

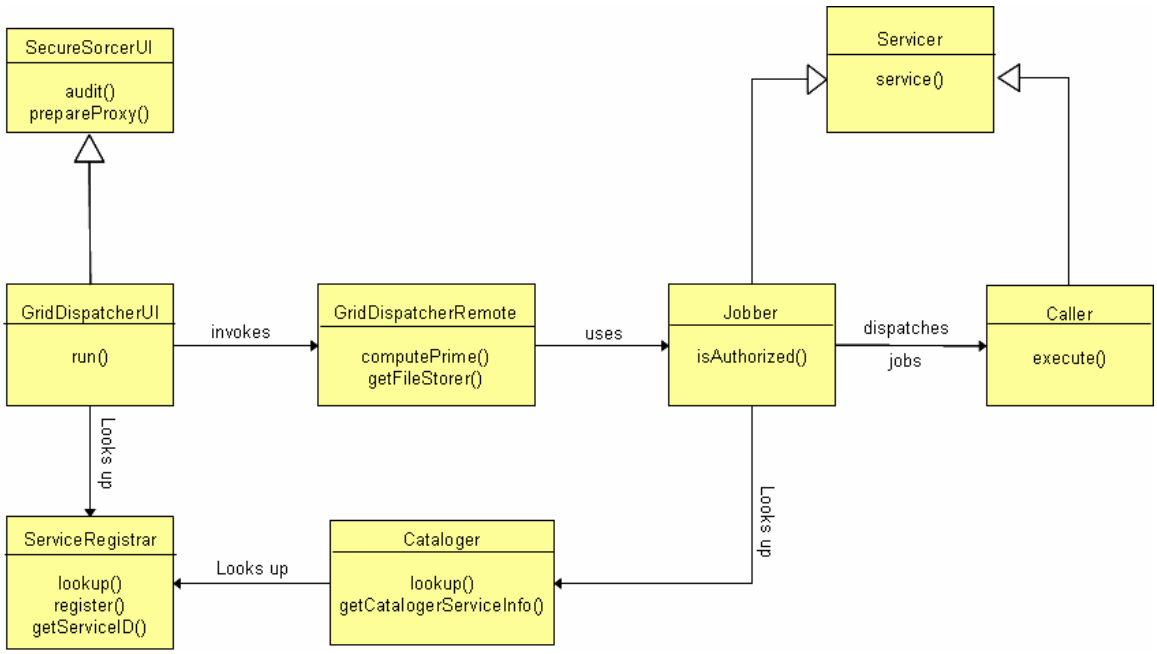

Figure 5.13 Class Diagram

Auditing

Auditing is enabled with the help of a provider called Auditor (Figure 5.14). The service providers send the critical messages to the Auditor for it to log the messages. The auditor can either log them into files or submit the messages to the database in predefined format.

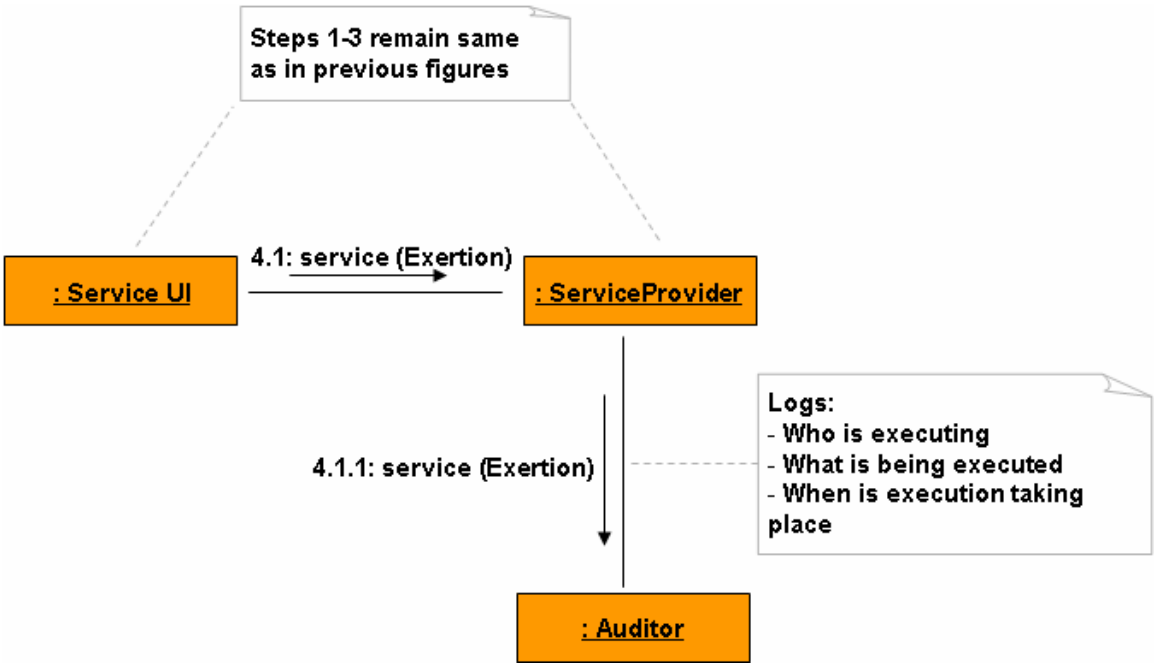

Figure 5.14 Use Case: Auditor

Proxy Verification

The Figure 5.15 shows the flow in which the proxy verification takes place in the framework.

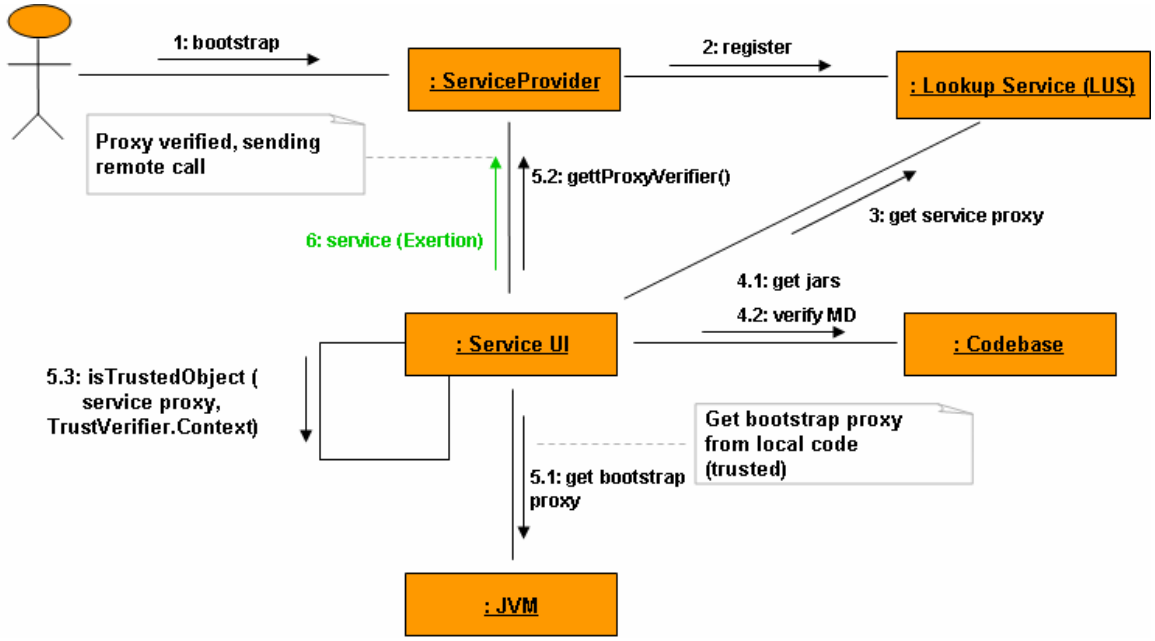

Figure 5.15 Proxy Verification

Figure 5.16 shows the physical configuration of how the resources were deployed to show the working of the suggested framework.

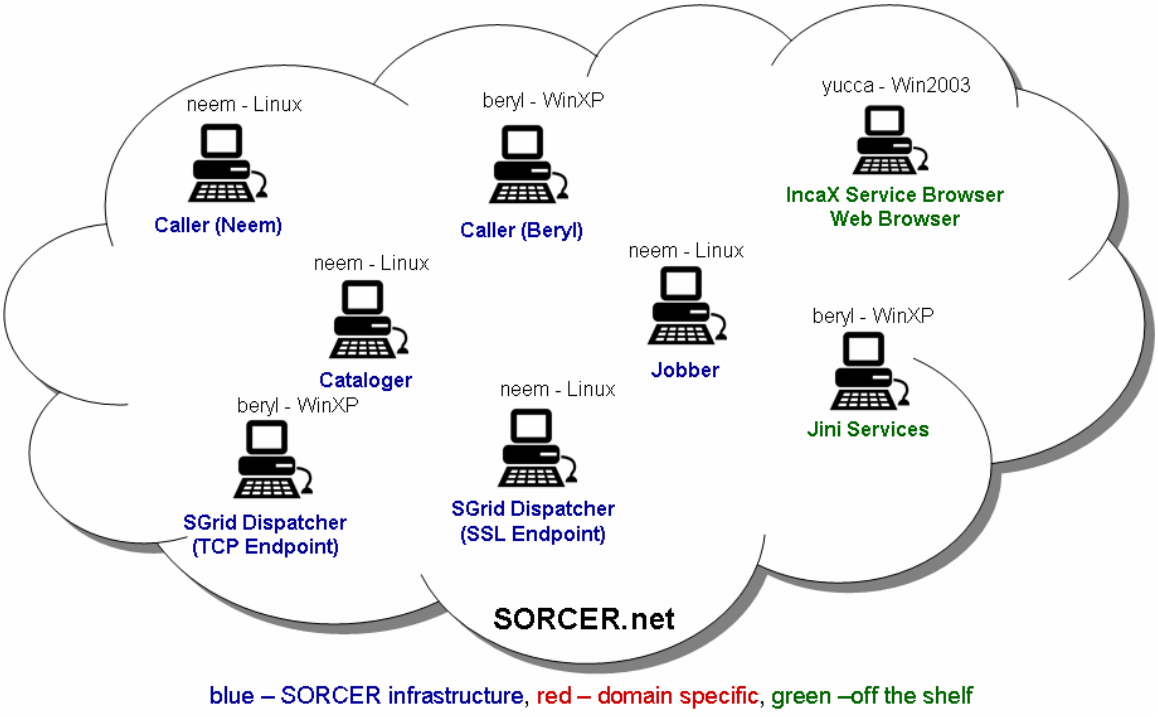

Figure 5.16 Deployment Diagram

It is worth noticing that the various services have been started on different platforms (OS). This shows the heterogeneous nature of the SGrid framework. Following servers have been used to deploy the services:

- Neem (Linux)
- Emerald (Windows XP)
- Beryl (Windows XP)
- Yucca (Windows 2003)
- The requestor uses the IncaX [33] service Browser, to access the list of available services on the network. Here, requestor computer (Yucca) has no prior setup of SORCER framework which signifies the 0-Install feature of the SORCER framework. A view of the IncaX browser has been given in Figure 5.17:

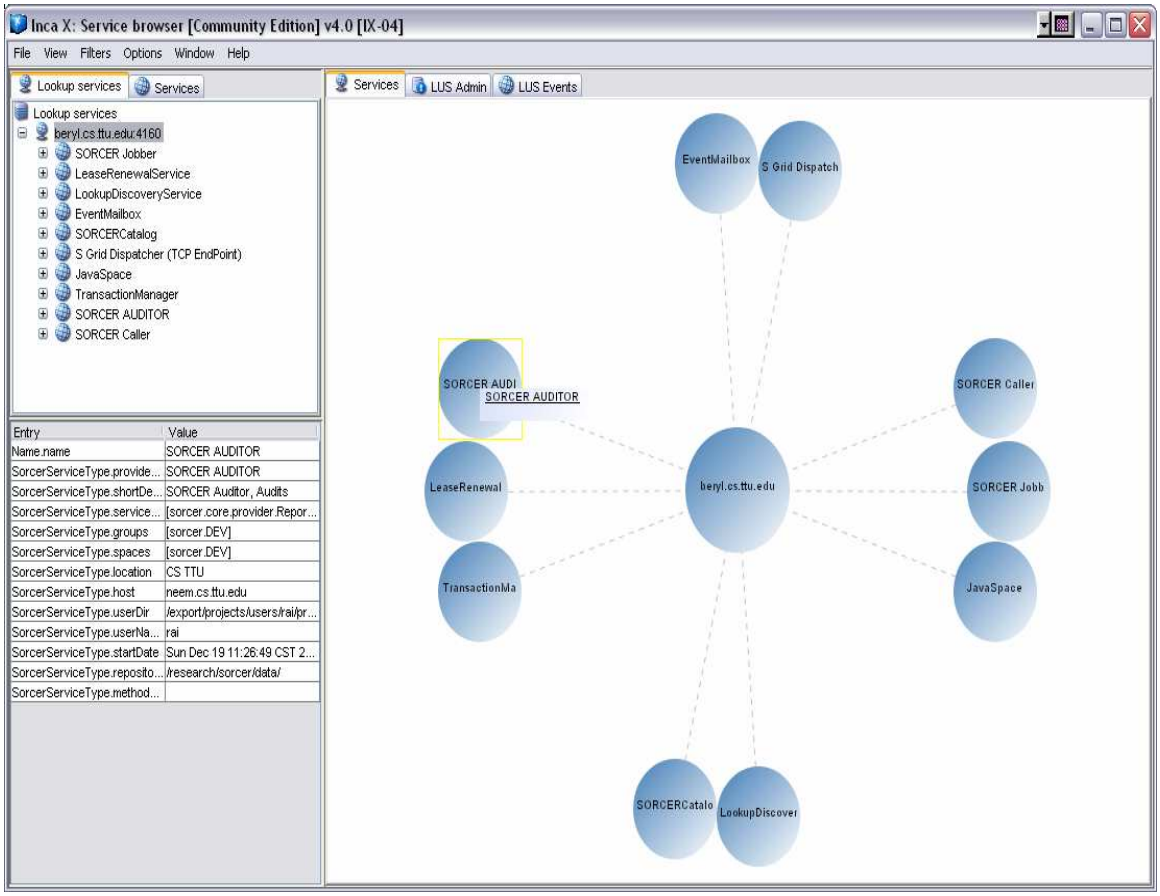

Figure 5.17 Incax Browser

5.4 Validation

The validation of the framework was done on the SGrid Framework. Following snapshots show the working of the framework as required in the use cases:

5.4.1 User Authentication:

The Figure 5.18 shows the snapshot of a user logging in the network.

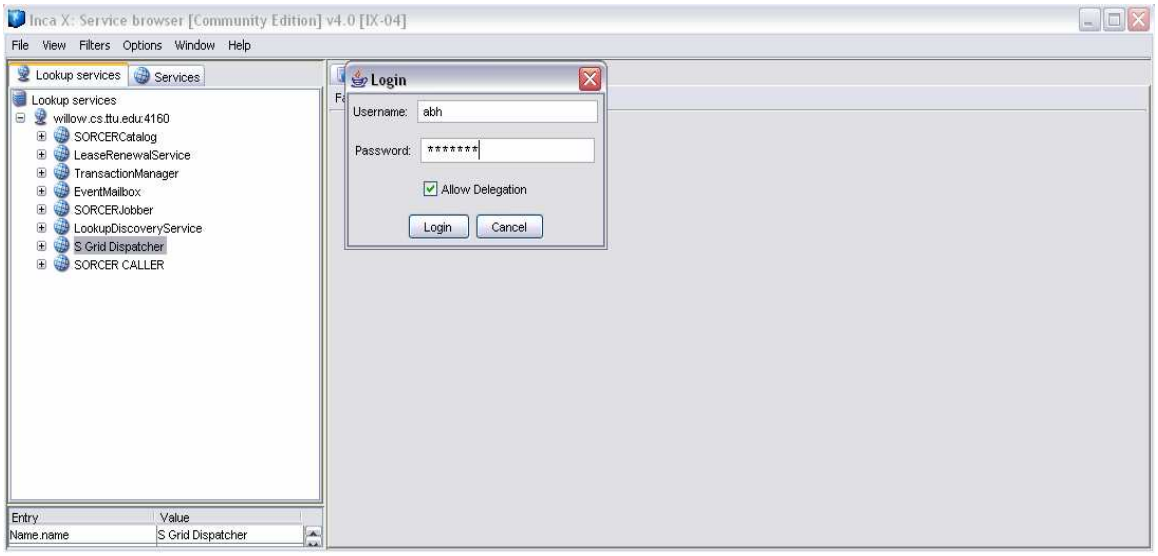

Figure 5.18 User Authentication

The log-in window show up even before the main Service UI window shows up. This is because the Service UI extends `SecureSorcerUI` class which locks the window unless the password is input.

5.4.2 Authorization:

In the Figure 5.19 snapshot of how remote call fails when a user who is not allowed access to invoke a particular method on the provider, tries to invoke the method. In this case "anonymous" who is not allowed to invoke "`computePrime()`" method on the SGrid Dispatcher get a "remote called failed exception".

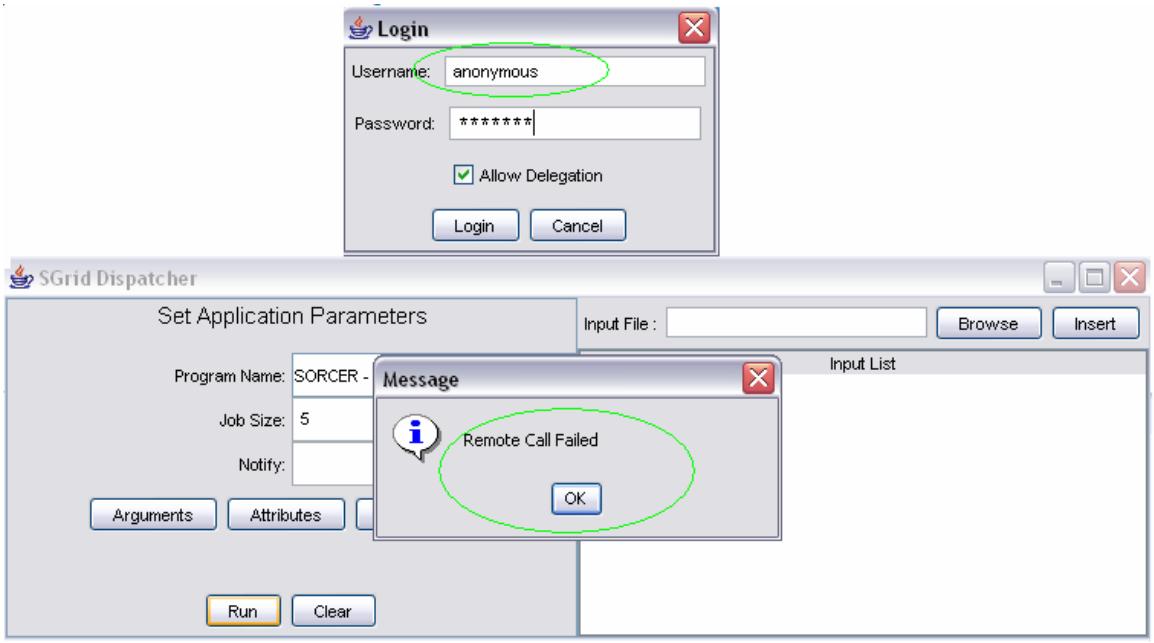

Figure 5.19 Anonymous Log in – Authorization failed

The green circles show that the user "anonymous" has logged-in and he has received a "Remote Call Failed" exception.

5.4.3 Caller – Integrity Check

The Figure 5.20 shows the snap shot of the result when the executable fails the integrity check at the callers end. Caller responds with "File has been Modified" Exception in the output window. Caller responds in such as way because in case the file has been modified recently the caller doesn't have the information of the file being changed. According to its information the file has been changed and it will not execute the file unless its information has been updated (by a trusted user) with the hash of the new file.

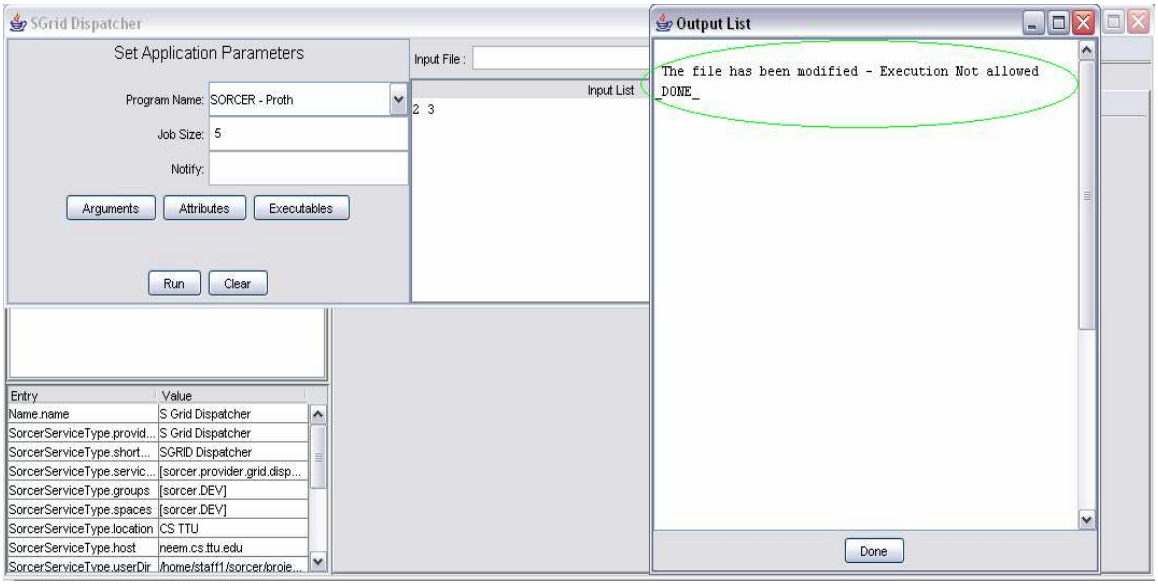

Figure 5.20 Caller Integrity Check

Once the user receives this error message, he shall update the Caller information about the file either by himself (if he has proper authority) or by communicating the problem to an administrator.

5.4.4 Successful Communication

The Figure 5.21 show the snapshot of a successful communication in which case an output window is popped-up which contains the results as obtained by the Caller.

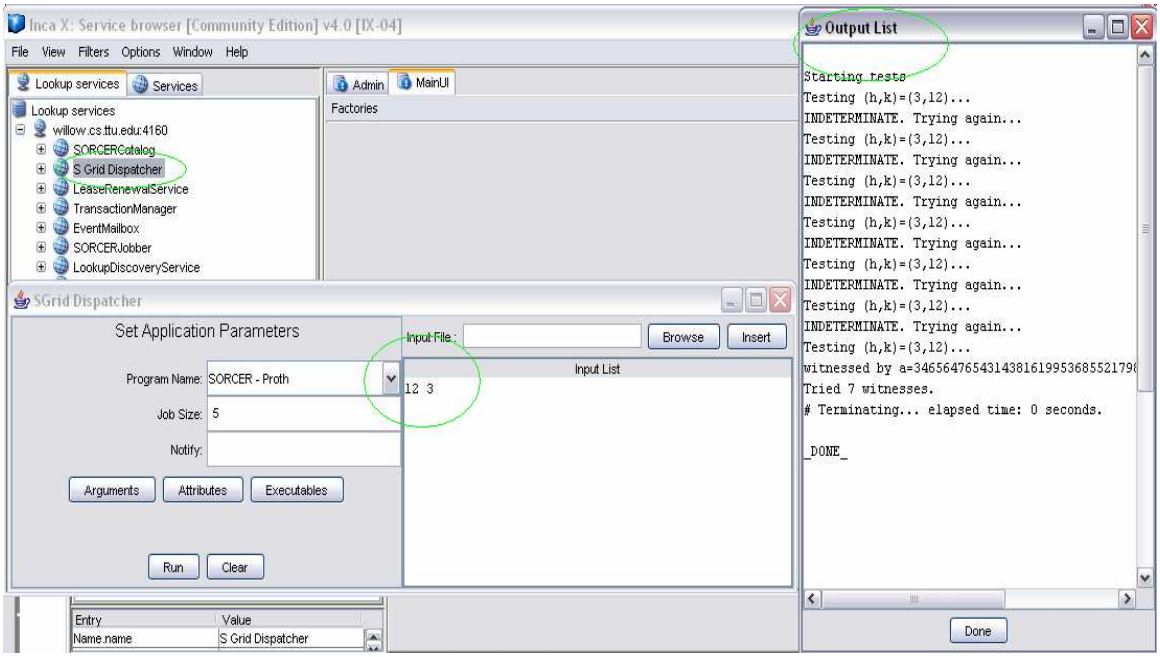

Figure 5.21 Successful communication

The output window is marked with the green circle. The results are the standard results from Proth.exe when the input is given as (12 ,3).

5.5 Future Work

5.5.1 Secure JavaSpaces

Since the prime objective of a JavaSpace is to be publicly available, the task of making Java Space secure becomes substantially difficult. Also, the JavaSpace works with “Pull” technology, meaning that it is just a store of Entries. The providers pull the entries from the space and put into the space accordingly. JavaSpace doesn’t distribute the jobs, and this makes it very difficult to control who puts and takes the jobs form the space.

If the secure framework cannot control who is putting the jobs in the exertion space and who is taking the job form it, the whole framework can easily be compromised.

Since securing exertion space is a big problem in itself it will be a very good topic in extension to the proposed framework.

5.5.2 Cataloger Security

The cataloger also shall be made secure. There shall be provision where it registers/provides proxies to only trusted providers. Otherwise there is always threat of rogue services getting registered as trusted services on the network

The framework shall provide mutual trust environment so that services can trust that they are not registering to a rogue cataloger. Also, the cataloger shall register only trusted services and provide proxies to trusted services only.

5.5.3 Custom Server End Points for SORCER

The implementation consists of servers using SSL endpoints (Jini Default `SslServerEndpoint`). The other choices were `KerberosServerEndpoint`, `HttpsServerEndpoint` etc. However, these are default implementations provided by Jini, are not very flexible. There may be a lot of other requirements to be able to work in secure framework, these requirements call for creation of custom Server endpoints which can enable secure communication in more flexible ways.

For example, `SslServerEndpoint` doesn't support `Delegation.YES` constraint because the Private credentials of the Subject cannot pass from the Subject to the service provider, for security reasons. However, `KerberosServerEndpoint` does support `Delegation.YES` constraint. Kerberos Endpoints however cannot provide as good confidentiality as SSL Endpoints can (RSA Encryption).

Requiring both scenarios for a secure framework generally calls for additional measures such as use of SPKI keys etc. A customized endpoint which inherits all the capabilities provided by both of these endpoints will be a great addition to the security in the framework.

5.5.4 Providing extra authentication capabilities

A useful extension to the proposed framework would be to introduce more secure methods of user Authentication. Few of these methods care outlined below:

- Java Card
- Finger Prints
- Voice Recognition
- Retinal Scan etc.

This will also help in Authorization based on which method of authentication user used to get authenticated. For example the user who got authenticated using “retinal scan” (hence more trusted) will be given more privileges to the Grid as compared to the user who got authenticated using Java Card (because the card has more probability of getting compromised).

REFERENCES

[1] IBM Autonomic Computing Initiative, Retrieved from http://www-306.ibm.com/autonomic/index.shtml

[2] I. Foster, and C.Kesselman, (eds.). The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.

[3] Rajkumar B. High Performance Cluster Computing Volume 1. Prentice Hall, 1999.

[4] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the i-way high performance distributed computing experiment,

[5] In Proc. 5th IEEE Symposium on High Performance Distributed Computing, pages 562_571, 1997

[6] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration, 2002.

[7] Grimshaw, A.S. "Enterprise-Wide Computing." Science, 256: 892-894, Aug. 12, 1994.

[8] Sun cluster grid architecture whitepaper, 2002, Retrieved from http://wwws.sun.com/software/grid/SunClusterGridArchitecture.pdf

[9] Larry J. Mitchell, Dennis Reddy. Fault Tolerant Dynamic Distributed Computations & Grid Systems

[10] Rajkumar B. High Performance Cluster Computing Volume 1. Prentice Hall, 1999.

[11] Jan Newmarch's Guide to Jini Technologies, Retrieved from http://pandonia.canberra.edu.au/java/jini/tutorial/Jini.xml

[12] Soorianarayanan, S (2004-2005). Autonomic Provisioning in SORCER Environment, Masters Thesis

[13] Zhao, S. & Sobolewski, M. (2001). "Context Model Sharing in the SORCER Environment"

[14] Garms, J. and Somerfield, D. (2001). *Professional Java Security*, Wrox Press Ltd., ISBN 1861004257

[15] Gorissen, D. H2O Metacomputing - Jini Lookup and Discovery (2003-2004), Master's Thesis

[16] Jini 2.0 New API Jini Network Technology Specifications. Retrieved Febuarary 5, 2003 from www.sun.com/software/jini/specs

[17] Jini Starter Kit 2.0, Retrieved from http://www.javaworld.com/javaworld/jw-05-2003/jw-0509-jiniology-p3.html

[18] Jini Technology Starter Kit Overview v2.01, Retrtieved from http://java.sun.com/developer/products/jini/arch2_0.html

[19] Nathalie, F. William, L. Mayer, A. Newhouse, S. Darlington, J. (2002), ICENI: An Open Grid Service Architecture Implemented with Jini, Retrieved from, http://citeseer.nj.nec.com/furmento02iceni.html

[20] JavaTM Remote Method Invocation (RMI) [online] [cited 2002 March 22]. Available at URL: http://java.sun.com/products/jdk/rmi/

[21] Basic Services (Reggie, Mahalo, JavaSpaces), Retrieved from http://www.cdegroot.com/articles/jini-newsletter/2000-13-basic-services-2/

[22] Reggie Reference, Retrieved from http://www.kedwards.com/jini/reggie.html

[23] Basic Services (Mercury, Norm, Fiddler), Retrieved from http://www.cdegroot.com/articles/jini-newsletter/2000-14-basic-services-3/

[24] Edwards, W.K. (2000). *Core Jini*: 2nd ed., Prentice Hall, ISBN 0-13-089408-7

[25] Java 2 API, Retrieved from http://java.sun.com/j2se/1.4.2/docs/api/

[26] Jini by example- white paper. Retrieved Febuarary 5, 2003 from http://www.cswl.com/whiteppr/tutorials/jini.html

[27] Röhl P. J., Kolonay, R.M., Irani, R.K., Sobolewski, M.,Kao, K. 2000. "A Federated Intelligent Product Environment,AIAA-2000-4902, 8th AIAA /USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, CA, September 6-8.

[28] FIPER (Federated Intelligent Product Environment) [online] [cited 2002 March 22]. Retrieved Febuarary 5, 2003 from http://www.oai.org/pages/FIPER.html.

[29] Kolonay, R.M., Sobolewski, M. "Grid interactive service-oriented programming environment", CE2004: The 11th ISPE International Conference on Concurrent Engineering: Research and Applications Beijing Friendship Hotel - Beijing, P. R. China 26 - 30 July, 2004

[30] Sobolewski, M. 2002. FIPER: The Federated S2S Environment, JavaOne, Sun's 2002 Worldwide Java Developer Conference, San Francisco, 2002.

[31] Soorianarayanan, S and Sobolewski M. "Monitoring Federated Services in CE Grids", CE2004: The 11th ISPE International Conference on Concurrent Engineering: Research and Applications Beijing Friendship Hotel - Beijing, P. R. China 26 - 30 July, 2004

[32] Jini Extensible Remote Invocation. Retrieved Febuarary 5, 2003 from http://www.artima.com/intv/jeri.html

[33] Incax. Retrieved from http://www.incax.com/

[34] Michael Sobolewski, Sekhar Soorianarayanan and Ravi-Kiran Malladi-Venkata, Service-Oriented File Sharing, Proceedings of the IASTED Intl., Conference on Communications, Internet, and Information technology Nov 17-19, 2003, Scottsdale, AZ.

[35] Arrington C. T, Rayhan S.H. Enterprise Java and UML, 2nd Edition, Wiley & Sons, ISBN 0-47-126778-3

[36] Rich B. A. Java Authentication and Authorization services, Retrieved from http://se2c.uni.lu/tiki/se2c-bib_download.php?id=470

[37] Java Glossary: Policy, Retrieved from http://www.mindprod.com/jgloss/policyfile.html

[38] The Java Developers Almanac 1.4: Managing Policy Files, Retrieved from http://javaalmanac.com/egs/java.security/UsePolicy.html

[39] Jurjens J. Secure Java Development with UML, Retrieved from http://www4.in.tum.de/~juerjens/papers/inetsec01talk.pdf

APPENDIX A:

SGRID INTERFACES

Following is a List of Interfaces that are used in the Framework:

sorcer.base.Cataloger

```
public interface Cataloger extends Remote {
   public Provider lookup(Entry[] attributes) throws RemoteException;
   public Provider lookup(ServiceID serviceID) throws RemoteException;
   public HashMap getProviderMethods() throws RemoteException;
   public ServiceContext getContexts(String provider, String method) throws
RemoteException;
   public String getCatalogerServiceInfo() throws RemoteException;
}
```

sorcer.core.Jobber

```
public interface Jobber extends AdministratableProvider, Remote {
   public boolean isAuthorized(Subject subject, String serviceType, String
providerName)
         throws RemoteException;
}
```

sorcer.core.Caller

```
public interface Caller extends Remote {
   public ServiceContext execute(ProviderContext context) throws
RemoteException;
```

```
    }
```

sorcer.core.Auditor

```
    public interface Auditor extends Remote {
        public ServiceContext audit(ProviderContext context) throws
RemoteException;
    }
```

sorcer.provider.grid.dispatcher.GridDispatcherRemote

```
    public interface GridDispatcherRemote extends Remote {
        public ServiceContext computePrime(ServiceContext ctx) throws
RemoteException;
        public FileStorer getFileStorer() throws RemoteException;
    }
```

APPENDIX B

CONFIGURATION FILES FOR SECURE SERVER

The configuration file is required to start a server with a Subject so that it can start with proper SslEndpoints. The configuration file looks like the following:

```
sorcer.core.provider.ServiceProvider{
    private static users=
        KeyStores.getKeyStore("file:../config/truststore.server", null);
    private static clientUser =
        KeyStores.getX500Principal("client", users);

    serverExporter =
        new BasicJeriExporter(
            SslServerEndpoint.getInstance(0),
            new BasicILFactory(
                new BasicMethodConstraints(
                    new InvocationConstraints(
                        new InvocationConstraint[]{Integrity.YES},
                        (InvocationConstraint []) null)),
                null
            )
        );
    loginContext = new LoginContext("GAppLogin");
```

```
}
```

Enabling SSL endpoints

The SSL endpoints are enabled with the help of `BasicJeriExporter` class:

```
new BasicJeriExporter(
        SslServerEndpoint.getInstance(0),
        new BasicILFactory(
            new BasicMethodConstraints(
                new InvocationConstraints(
                    new InvocationConstraint[]{Integrity.YES},
                    (InvocationConstraint []) null)),
            null
        )
    );
```

`SslServerEndpoint.getInstance(0)` enables server to start the SSL server at any free port possible. With the help of `BasicILFactory class`, various constraints (`Integrity.YES`) can be specified, so that the server looks for these constraints before allowing any transaction.

Assigning a Subject

Subject is assigned with the help of JAAS framework again. The “loginContext” is specified and this loginContext is then provided.

```
loginContext = new LoginContext("GAppLogin");
```

The server then looks for "`GappLogin`" in the configuration file which is given as the system property "`java.security.auth.login.config`" at the run time. This file's contents look like this:

```
GAppLogin {
  jgapp.jaas.PsswdLoginModule required;
};
GAppLogin {
  com.sun.security.auth.module.KeyStoreLoginModule required
  keyStoreAlias="mykey"
  keyStoreURL="file:../config/keystore.server"
  keyStorePasswordURL="file:../config/password.server";
};
```

APPENDIX C:

POLICY FILES

The policy file is very important part of starting a server with SSL and controlling access based on Authorization [38] [37].

```
grant codebase "file:../lib/dispatcher.jar"{
        permission java.net.SocketPermission "*", "connect,accept,listen";
        permission javax.security.auth.PrivateCredentialPermission
"javax.security.auth.x500.X500PrivateCredential javax.security.auth.x500.X500Principal
\"*\"", "read";
        permission net.jini.security.AuthenticationPermission
"javax.security.auth.x500.X500Principal \"*\"", "accept";
        permission javax.security.auth.AuthPermission
"createLoginContext.GAppLogin";
        permission java.io.FilePermission "<<ALL FILES>>" ,
"read,write,execute";
        permission javax.security.auth.AuthPermission "doAsPrivileged";
        permission net.jini.discovery.DiscoveryPermission "sorcer.DEV";
        //permission sorcer.provider.dispatcher.MethodPermission "*";
        permission java.lang.RuntimePermission "getClassLoader";
        permission net.jini.io.context.ContextPermission
"net.jini.io.context.ClientSubject.getClientSubject";
};
```

```
        grant principal javax.security.auth.x500.X500Principal "CN=abhijit, OU=a, O=a,
L=a, ST=a, C=a"{
                permission java.security.AllPermission;
        };

        grant principal javax.security.auth.x500.X500Principal "CN=Server, OU=IT,
O=Sorcer, L=Lubbock, ST=Tx, C=US"{
                permission java.net.SocketPermission "*", "connect,accept,listen";
                permission javax.security.auth.PrivateCredentialPermission
"javax.security.auth.x500.X500PrivateCredential javax.security.auth.x500.X500Principal
\"*\"", "read";
                permission javax.security.auth.AuthPermission "doAsPrivileged";
                permission javax.security.auth.AuthPermission "doAs";
                permission javax.security.auth.AuthPermission "getSubject";
                permission net.jini.discovery.DiscoveryPermission "sorcer.DEV";
                permission java.lang.RuntimePermission "getClassLoader";
                permission java.io.FilePermission "provider.properties", "read";
                permission java.io.FilePermission "provider.log", "read, write";
                permission java.io.FilePermission "<<ALL FILES>>" , "read";
                permission java.util.PropertyPermission "java.home", "read";
                permission java.util.PropertyPermission "sorcer.env.file", "read";
                permission java.util.PropertyPermission "sorcer.debug", "read";
```

```
        permission java.util.PropertyPermission "sorcer.home", "read";
        permission java.util.PropertyPermission "sorcer.rmi.host", "read";
        permission java.util.PropertyPermission "sorcer.rmi.port", "read";
        permission java.util.PropertyPermission "sorcer.http.host", "read";
        permission java.util.PropertyPermission "sorcer.http.port", "read";
        permission java.util.PropertyPermission "sorcer.portal.host", "read";
        permission java.util.PropertyPermission "sorcer.portal.port", "read";
        permission java.util.PropertyPermission "sorcer.lib.codebase", "read";

        permission java.util.PropertyPermission "user.dir", "read";
        permission java.util.PropertyPermission "user.name", "read";

        permission java.lang.RuntimePermission "getClassLoader";
        permission java.lang.RuntimePermission "setIO";
        permission sorcer.security.permission.MethodPermission "*";

};
```

Assigning policy to different Subjects

As given above, different set of policies have to be given for different codebase and Subjects. The reason is if All permissions are given to the code base “dispatcher.jar” (in this case, then the code will no longer look for separate permissions for different Subjects. Hence very restrictive amount of permissions shall be given to the codebase. A list of important (required) permissions to be given is provided below:

- java.net.SocketPermission javax.security.auth.PrivateCredentialPermission
- permission net.jini.security.AuthenticationPermission
- javax.security.auth.AuthPermission
- permission java.io.FilePermission
- permission net.jini.discovery.DiscoveryPermission
- permission java.lang.RuntimePermission
- permission net.jini.io.context.ContextPermission

Now, after allowing the above permissions to the codebase, each Subject shall be provided with its own set of permission. In the case of above policy file (for example), the “Client” subject has been allowed all permissions, however, the Server itself has been restricted to certain permissions.

APPENDIX D

API SPECIFICATION

Package sorcer.provider.grid.dispatcher

Interface Summary	
GridDispatcherRemote	The interface for S Grid Dispatcher

Class Summary	
GridDispatcherArgUI	This is the helper class which renders the Argument UI when the Arguments button in the SGrid UI is clicked
GridDispatcherAttribUI	This is the helper class which renders the Argument UI when the Arguments button in the SGrid UI is clicked
GridDispatcherContextUtil	Helper class for the SGrid Dispatcher (UI and Provider) to set and get Caller

	Context
GridDispatcherCtxUI	This is the helper class which renders the Operating System UI when the Arguments button in the SGrid UI is clicked
GridDispatcherExecUI	Helper class to create the UI for specyfying the executables to be run
GridDispatcherProviderImpl	The impl class for s Grid Dispatcher
GridDispatcherProviderImpl.Disco	The class which looks up for jobber proxy from the lookup service
GridDispatcherProviderImpl.DispatcherResult	Class which waits for the return of the results from Caller
GridDispatcherProviderImpl.JobDispatcher	
GridDispatcherProviderImpl.JobsDispatcher	Class for dispatching the job to the Jobber, internally used by

	JobDispatcher class
GridDispatcherUI	The Class which renders the S Grid service UI
GridDispatcherUI.DispatcherListener	DispatcherListener Class listens for the results

sorcer.provider.grid.dispatcher

Interface GridDispatcherRemote

All Superinterfaces:

java.rmi.Remote

All Known Implementing Classes:

GridDispatcherProviderImpl

---

public interface GridDispatcherRemote

extends java.rmi.Remote

The interface for S Grid Dispatcher

Author:

Abhijit Rai

Method Summary	
sorcer.base.ServiceContext	**computePrime**(sorcer.base.ServiceContext ctx) This method dispatched the compute call to Caller.
sorcer.core.FileStorer	**getFileStorer**() Gets the proxy for filestore provider

Method Detail

computePrime

```
public sorcer.base.ServiceContext
computePrime(sorcer.base.ServiceContext ctx)
                                        throws
java.rmi.RemoteException
```

This method dispatched the compute call to Caller. It is named computePrime because it was first developed for Sorcer Proth

Returns:

ServiceContext with the compute results included

Throws:

java.rmi.RemoteException

---

getFileStorer

```
public sorcer.core.FileStorer getFileStorer()
                                      throws
java.rmi.RemoteException
```

Gets the proxy for filestore provider

Returns:

the filestore provider

Throws:

java.rmi.RemoteException

sorcer.provider.grid.dispatcher

Class GridDispatcherArgUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └
sorcer.provider.grid.dispatcher.GridDispatcherArgUI
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class GridDispatcherArgUI

extends javax.swing.JFrame

implements java.awt.event.ActionListener

This is the helper class which renders the Argument UI when the Arguments button in the SGrid UI is clicked

Author:

Abhijit Rai

See Also:

Serialized Form

Nested Class Summary

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

javax.swing.JTextField	**addArgsTfld**
javax.swing.JTextField	**addInTfld**
javax.swing.JTextField	**addOutTfld**
javax.swing.JCheckBox	**argsChkBx**
javax.swing.JTextArea	**argsTarea**
javax.swing.JComboBox	**callPrvCbx**
javax.swing.JComboBox	**hostnameCbx**
javax.swing.JCheckBox	**inChkBx**
javax.swing.JTextArea	**inTarea**
javax.swing.JComboBox	**locationCbx**
javax.swing.JComboBox	**opSysCbx**

javax.swing.JCheckBox	**outChkBx**
javax.swing.JTextArea	**outTarea**

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants

DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
GridDispatcherArgUI() Prepares the UI by combining various components

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent ae) Called when an action occurs on the Argument UI

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets,

getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent,

processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object

clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer

getFont, postEvent

Field Detail

addArgsTfld

```
public javax.swing.JTextField addArgsTfld
```

addInTfld

```
public javax.swing.JTextField addInTfld
```

---

addOutTfld

```
public javax.swing.JTextField addOutTfld
```

---

locationCbx

```
public javax.swing.JComboBox locationCbx
```

---

hostnameCbx

```
public javax.swing.JComboBox hostnameCbx
```

---

opSysCbx

```
public javax.swing.JComboBox opSysCbx
```

---

callPrvCbx

```
public javax.swing.JComboBox callPrvCbx
```

---

argsTarea

```
public javax.swing.JTextArea argsTarea
```

---

inTarea

```
public javax.swing.JTextArea inTarea
```

---

outTarea

```
public javax.swing.JTextArea outTarea
```

---

argsChkBx

```
public javax.swing.JCheckBox argsChkBx
```

---

inChkBx

```
public javax.swing.JCheckBox inChkBx
```

---

outChkBx

```
public javax.swing.JCheckBox outChkBx
```

Constructor Detail

GridDispatcherArgUI

```
public GridDispatcherArgUI()
```

Prepares the UI by combining various components

Method Detail

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent ae)
```

Called when an action occurs on the Argument UI

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Parameters:

ae - The event which was caused by the action

sorcer.provider.grid.dispatcher

Class GridDispatcherAttribUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └
```

sorcer.provider.grid.dispatcher.GridDispatcherAttribUI

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class GridDispatcherAttribUI

extends javax.swing.JFrame

implements java.awt.event.ActionListener

This is the helper class which renders the Argument UI when the Arguments button in the SGrid UI is clicked

Author:

Abhijit Rai

See Also:

Serialized Form

Nested Class Summary

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
GridDispatcherAttribUI() Prepares the UI by combining various components

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent ae)

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale,

setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Constructor Detail

GridDispatcherAttribUI

```
public GridDispatcherAttribUI()
```

Prepares the UI by combining various components

Method Detail

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent ae)
```

Specified by:

actionPerformed in interface java.awt.event.ActionListener

sorcer.provider.grid.dispatcher

Class GridDispatcherContextUtil

```
java.lang.Object
  └ sorcer.provider.grid.dispatcher.GridDispatcherContextUtil
```

---

public class GridDispatcherContextUtil

extends java.lang.Object

Helper class for the SGrid Dispatcher (UI and Provider) to set and get Caller Context

---

Constructor Summary
GridDispatcherContextUtil()

Method Summary	
static net.jini.core.event.RemoteEventListener	**getCallback**(sorcer.base.ServiceContext ctx)
static java.lang.String	**getException**(sorcer.base.ServiceContext ctx)
static java.lang.String	**getInputFile**(sorcer.base.ServiceContext ctx)
static java.lang.String[]	**getInputValues**(sorcer.base.ServiceContext ctx)

static int	**getJobSize**(sorcer.base.ServiceContext ctx)
static java.lang.String	**getNotify**(sorcer.base.ServiceContext ctx)
static java.lang.String	**getOutputFile**(sorcer.base.ServiceContext ctx)
static void	**setCallback**(sorcer.base.ServiceContext ctx, net.jini.core.event.RemoteEventListener rel) sets callback path for the Caller Context
static void	**setException**(sorcer.base.ServiceContext ctx, java.lang.String values) sets jobsize for the Caller Context
static void	**setInputFile**(sorcer.base.ServiceContext ctx, java.lang.String file) sets input file path path for the Caller Context
static void	**setInputValues**(sorcer.base.ServiceContext ctx, java.lang.String[] values) sets input values for the Caller Context
static void	**setJobSize**(sorcer.base.ServiceContext ctx, java.lang.String size)

	sets jobsize for the Caller Context
static void	**setNotify**(sorcer.base.ServiceContext ctx, java.lang.String notify) sets notify path for the Caller Context
static void	**setOutputFile**(sorcer.base.ServiceContext ctx, java.lang.String file) sets the output file path for the Caller Context

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GridDispatcherContextUtil

```
public GridDispatcherContextUtil()
```

Method Detail

getOutputFile

```
public static java.lang.String
getOutputFile(sorcer.base.ServiceContext ctx)
```

Parameters:

ctx - Caller Context

Returns:

returns the output file path as a String (extracts from the Caller Context)

---

setOutputFile

```
public static void setOutputFile(sorcer.base.ServiceContext ctx,
                                 java.lang.String file)
```

sets the output file path for the Caller Context

Parameters:

ctx - Caller Context

---

getCallback

```
public static net.jini.core.event.RemoteEventListener
getCallback(sorcer.base.ServiceContext ctx)
```

Parameters:

ctx - Caller Context

Returns:

returns callback path

---

setCallback

```
public static void setCallback(sorcer.base.ServiceContext ctx,
net.jini.core.event.RemoteEventListener rel)
```

sets callback path for the Caller Context

Parameters:

ctx - Caller Context

rel - Remote evvent listener for the call back handler

---

getNotify

```
public static java.lang.String
getNotify(sorcer.base.ServiceContext ctx)
```

Parameters:

ctx - Caller Context

Returns:

returns notify path as a String (extracts from the Caller Context)

---

setNotify

```
public static void setNotify(sorcer.base.ServiceContext ctx,
                             java.lang.String notify)
```

sets notify path for the Caller Context

Parameters:

ctx - Caller Context

notify -

---

getInputFile

```
public static java.lang.String
getInputFile(sorcer.base.ServiceContext ctx)
```

Parameters:

ctx - Caller Context

Returns:

returns input file path as a String (extracts from the Caller Context)

---

setInputFile

```
public static void setInputFile(sorcer.base.ServiceContext ctx,
                                 java.lang.String file)
```

sets input file path path for the Caller Context

Parameters:

ctx - Caller Context

file - filename

---

getInputValues

```
public static java.lang.String[]
getInputValues(sorcer.base.ServiceContext ctx)
```

Parameters:

ctx - Caller Context

Returns:

returns input values as an array String (extracts from the Caller Context)

---

setInputValues

```
public static void setInputValues(sorcer.base.ServiceContext ctx,
                                  java.lang.String[] values)
```

sets input values for the Caller Context

Parameters:

ctx - Caller Context

values - array of input values

---

getJobSize

```
public static int getJobSize(sorcer.base.ServiceContext ctx)
```
Parameters:

ctx - Caller Context

Returns:

returns the jobsize set in the Caller Context

---

setJobSize

```
public static void setJobSize(sorcer.base.ServiceContext ctx,
                              java.lang.String size)
```
sets jobsize for the Caller Context

Parameters:

ctx - Caller Context

size - size of the job

---

getException

```
public static java.lang.String
getException(sorcer.base.ServiceContext ctx)
```
Parameters:

ctx - Caller Context

Returns:

returns exception received in the Caller Context

---

setException

```
public static void setException(sorcer.base.ServiceContext ctx,
                                java.lang.String values)
```

sets jobsize for the Caller Context

Parameters:

ctx - Caller Context

values -

sorcer.provider.grid.dispatcher

Class GridDispatcherCtxUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └
sorcer.provider.grid.dispatcher.GridDispatcherCtxUI
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class GridDispatcherCtxUI

extends javax.swing.JFrame

implements java.awt.event.ActionListener

This is the helper class which renders the Operating System UI when the Arguments button in the SGrid UI is clicked

Author:

Abhijit Rai

See Also:

Serialized Form

Nested Class Summary

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
GridDispatcherCtxUI() Prepares the UI by combining various components

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent ae) Called when an action occurs on the Argument UI

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale,

setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Constructor Detail

GridDispatcherCtxUI

```
public GridDispatcherCtxUI()
```

Prepares the UI by combining various components

Method Detail

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent ae)
```

Called when an action occurs on the Argument UI

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Parameters:

ae - The event which was caused by the action

sorcer.provider.grid.dispatcher

Class GridDispatcherExecUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └
sorcer.provider.grid.dispatcher.GridDispatcherExecUI
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class GridDispatcherExecUI

extends javax.swing.JFrame

implements java.awt.event.ActionListener

Helper class to create the UI for specyfying the executables to be run

Author:

Abhijit Rai

See Also:

Serialized Form

Nested Class Summary

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
GridDispatcherExecUI() Prepares the UI by combining various components

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent ae) Called when an action occurs on the Argument UI

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale,

setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Constructor Detail

GridDispatcherExecUI

```
public GridDispatcherExecUI()
```

Prepares the UI by combining various components

Method Detail

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent ae)
```

Called when an action occurs on the Argument UI

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Parameters:

ae - The event which was caused by the action

sorcer.provider.grid.dispatcher

Class GridDispatcherProviderImpl

```
java.lang.Object
  └sorcer.core.provider.ServiceProvider
      └sorcer.core.provider.SorcerProvider
          └
sorcer.provider.grid.dispatcher.GridDispatcherProviderImpl
```

All Implemented Interfaces:

net.jini.admin.Administrable, sorcer.core.AdministratableProvider, com.sun.jini.admin.DestroyAdmin, java.util.EventListener, GridDispatcherRemote, net.jini.admin.JoinAdmin, sorcer.base.MonitorableServicer, sorcer.base.Provider, net.jini.export.ProxyAccessor, java.rmi.Remote, net.jini.core.constraint.RemoteMethodControl, java.io.Serializable, net.jini.security.proxytrust.ServerProxyTrust, net.jini.lookup.ServiceIDListener, sorcer.base.Servicer, sorcer.util.SORCER

---

public class GridDispatcherProviderImpl

extends sorcer.core.provider.SorcerProvider

implements GridDispatcherRemote, sorcer.util.SORCER

The impl class for s Grid Dispatcher

See Also:

Serialized Form

Nested Class Summary

static class	**GridDispatcherProviderImpl.Disco** The class which looks up for jobber proxy from the lookup service
static class	**GridDispatcherProviderImpl.DispatcherResult** Class which waits for the return of the results from Caller
static class	**GridDispatcherProviderImpl.JobDispatcher**
static class	**GridDispatcherProviderImpl.JobsDispatcher** Class for dispatching the job to the Jobber, internally used by JobDispatcher class

Nested classes inherited from class sorcer.core.provider.SorcerProvider
sorcer.core.provider.SorcerProvider.KeepAwake

Field Summary

Fields inherited from class sorcer.core.provider.ServiceProvider
delegate

Fields inherited from interface sorcer.util.SORCER
ADD_DATANODE, ADD_DOMAIN, ADD_JOB_TO_SESSION, ADD_LEAFNODE, ADD_SUBDOMAIN, ADD_TASK, ADD_TASK_TO_JOB_SAVEAS, ADD_TASK_TO_JOB_SAVEAS_RUNTIME, APPEND, AS_PROPS, AS_SESSION, ATTRIBUTE_MODIFIED, BGCOLOR, BROKEN_LINK, CATALOG_CONTENT, CATALOGER_EVENT, CLEANUP_SESSION, CMPS, Command, CONTEXT_ATTRIBUTE_VALUES, CONTEXT_ATTRIBUTES, CONTEXT_RESULT, CPS, CREATION_TIME, DATANODE_FLAG, DELETE_CONTEXT_EVT, DELETE_JOB_EVT, DELETE_NOTIFICATIONS, DELETE_SESSION, DELETE_TASK, DELETE_TASK_EVT, DROP_EXERTION, EXCEPTION_IND, EXCEPTIONS, EXERTION_PROVIDER, FALSE, GET, GET_CONTEXT, GET_CONTEXT_NAMES, GET_FT, GET_JOB, GET_JOB_NAME_BY_JOB_ID, GET_JOBDOMAIN, GET_JOBNAMES, GET_NEW_SERVLET_MESSAGES, GET_NOTIFICATIONS_FOR_SESSION, GET_RUNTIME_JOB, GET_RUNTIME_JOBNAMES, GET_SESSIONS_FOR_USER, GET_TASK, GET_TASK_NAME_BY_TASK_ID, GET_TASK_NAMES, GETALL_DOMAIN_SUB, IN_FILE, IN_PATH, IN_SCRIPT, IN_VALUE, IND, IS_NEW, JOB_ID, JOB_NAME, JOB_STATE, JOB_TASK, MAIL_SEP, MAX_LOOKUP_WAIT, MAX_PRIORITY, META_MODIFIED, MIN_PRIORITY, MODIFY_LEAFNODE, MSG_CONTENT, MSG_ID, MSG_SOURCE, MSG_TYPE, NEW_CONTEXT_EVT, NEW_JOB_EVT, NEW_TASK_EVT, NONE, NORMAL_PRIORITY, NOTIFY_EXCEPTION, NOTIFY_FAILURE, NOTIFY_INFORMATION, NOTIFY_WARNING, NOTRUNTIME, NULL, OBJECT_DOMAIN, OBJECT_NAME, OBJECT_OWNER, OBJECT_SCOPE, OBJECT_SUBDOMAIN, Order, OUT_COMMENT, OUT_FILE, OUT_PATH, OUT_SCRIPT, OUT_VALUE, PERSIST_CONTEXT, PERSIST_JOB, PERSIST_SORCER_NAME, PERSIST_SORCER_TYPES, PERSISTENCE_EVENT, POSTPROCESS, PREPROCESS, PRIVATE, PRIVATE_SCOPE, PROCESS, PROVIDER, PROVIDER_CONTEXT, PUBLIC_SCOPE, REGISTER_FOR_NOTIFICATIONS, REMOVE_CONTEXT, REMOVE_DATANODE, REMOVE_JOB, REMOVE_TASK,

RENAME_CONTEXT, RENAME_SORCER_NAME, RESUME_JOB, RUNTIME, SAPPEND, SAVE_TASK_AS, SAVEJOB_AS, SAVEJOB_AS_RUNTIME, SCRATCH_CONTEXTIDS, SCRATCH_JOBEXERTIONIDS, SCRATCH_METHODIDS, SCRATCH_TASKEXERTIONIDS, Script, SCRIPT, SELECT, SELF, SERVICE_EXERTION, SOC_BOOLEAN, SOC_CONTEXT_LINK, SOC_DATANODE, SOC_DB_OBJECT, SOC_DOUBLE, SOC_FLOAT, SOC_INTEGER, SOC_LONG, SOC_PRIMITIVE, SOC_SERIALIZABLE, SOC_STRING, SORCER_FOOTER, SORCER_HEADER, SORCER_HOME, SORCER_INTRO, SORCER_TMP_DIR, SPOSTPROCESS, SPREPROCESS, SPROCESS, STEP_JOB, STOP_JOB, STOP_TASK, SUBCONTEXT_CONTROL_CONTEXT_STR, SUSPEND_JOB, SYSTEM_SCOPE, TABLE_NAME, TASK_COMMAND, TASK_ID, TASK_JOB, TASK_NAME, TASK_PROVIDER, TASK_SCRIPT, TRUE, UPDATE_CONTEXT, UPDATE_CONTEXT_EVT, UPDATE_DATANODE, UPDATE_EXERTION, UPDATE_JOB, UPDATE_JOB_EVT, UPDATE_TASK, UPDATE_TASK_EVT

Constructor Summary
GridDispatcherProviderImpl() Constructor for the dispatcher
GridDispatcherProviderImpl(java.lang.String[] args, com.sun.jini.start.LifeCycle lifeCycle) Constructor for the dispatcher

Method Summary	
sorcer.base.ServiceContext	**computePrime**(sorcer.core.ProviderContext dispatcherCtx) method to be invoked by SGrid Dispatcher UI

sorcer.base.ServiceContext	**computePrime**(sorcer.base.ServiceContext dispatcherCtx) this method is called internally
sorcer.core.FileStorer	**getFileStorer**() Method finds the filestore provider
net.jini.lookup.entry.UIDescriptor	**getMainUIDescriptor**() Create s UI descriptor to publish the UI
void	**init**() Calls the init method of the superclass
boolean	**isValidTask**()
void	**setPrincipal**(sorcer.base.Exertion ex, java.security.Principal p) sets the principal for the exertion
void	**setPrincipal**(sorcer.base.Exertion ex, javax.security.auth.Subject subj) sets the principal for the exertion
void	**setPrincipal**(sorcer.base.ServiceContext ctx, java.security.Principal p) sets the principal for the context

Methods inherited from class sorcer.core.provider.SorcerProvider
addLookupAttributes, addLookupGroups, addLookupLocators, destroy, getAdmin, getConstraints,

getGrants, getLookupAttributes, getLookupGroups, getLookupLocators, getProxy, getProxyVerifier, getServiceProxy, grant, grantSupported, init, modifyLookupAttributes, removeLookupGroups, removeLookupLocators, setConstraints, setLookupGroups, setLookupLocators, toString

Methods inherited from class sorcer.core.provider.ServiceProvider

doJob, doTask, dropJob, dropTask, fireEvent, getAttributes, getDelegate, getDescription, getGroups, getInfo, getLeastSignificantBits, getMethodContexts, getMostSignificantBits, getProperties, getProperty, getProviderID, getProviderName, getScratchDirectory, getScratchURL, hangup, init, invokeMethod, invokeMethod, isActive, isValidMethod, isValidTask, loadConfiguration, notifyException, notifyException, notifyExceptionWithStackTrace, notifyFailure, notifyFailure, notifyInformation, notifyWarning, processJob, quit, removeScratchDirectory, restore, resume, service, service0, serviceIDNotify, setProperties, startTiming, step, stop, stopTiming, suspend, update

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface sorcer.base.Provider

fireEvent, getAttributes, getDescription, getGroups, getInfo, getMethodContexts, getProperties, getProperty, getProviderID, getProviderName, hangup, init, invokeMethod, invokeMethod, isValidMethod, isValidTask, notifyException, notifyException, notifyExceptionWithStackTrace, notifyFailure, notifyFailure, notifyInformation, notifyWarning, restore, setProperties, startTiming, stopTiming, update

Methods inherited from interface sorcer.base.MonitorableServicer
resume, step, stop, suspend

Methods inherited from interface sorcer.base.Servicer
service

Constructor Detail

GridDispatcherProviderImpl

```
public GridDispatcherProviderImpl()
                            throws java.rmi.RemoteException
```

Constructor for the dispatcher

GridDispatcherProviderImpl

```
public GridDispatcherProviderImpl(java.lang.String[] args,
com.sun.jini.start.LifeCycle lifeCycle)
                            throws java.lang.Exception
```

Constructor for the dispatcher

Parameters:

args - Arguments to be supplied

Method Detail

setPrincipal

```
public void setPrincipal(sorcer.base.Exertion ex,
```

```
                              javax.security.auth.Subject subj)
```
sets the principal for the exertion

Parameters:

ex - exertion to be dispatched

subj - subject from which principal is to be set

---

setPrincipal

```
public void setPrincipal(sorcer.base.Exertion ex,
                         java.security.Principal p)
```
sets the principal for the exertion

Parameters:

ex - exertion to be dispatched

---

setPrincipal

```
public void setPrincipal(sorcer.base.ServiceContext ctx,
                         java.security.Principal p)
```
sets the principal for the context

Parameters:

ctx - context to be dispatched

---

getFileStorer

```
public sorcer.core.FileStorer getFileStorer()
                                        throws
java.rmi.RemoteException
```
Method finds the filestore provider

Specified by:

getFileStorer in interface GridDispatcherRemote

Returns:

returns the proxy for filestore provider

Throws:

java.rmi.RemoteException

---

computePrime

```
public sorcer.base.ServiceContext
computePrime(sorcer.core.ProviderContext dispatcherCtx)
                                                throws
java.rmi.RemoteException
```

method to be invoked by SGrid Dispatcher UI

Parameters:

dispatcherCtx - Context sent by the UI

Throws:

java.rmi.RemoteException

---

computePrime

```
public sorcer.base.ServiceContext
computePrime(sorcer.base.ServiceContext dispatcherCtx)
                                                throws
java.rmi.RemoteException
```

this method is called internally

Specified by:

computePrime in interface GridDispatcherRemote

Parameters:

dispatcherCtx - Context sent by the UI

Returns:

ServiceContext with the compute results included

Throws:

java.rmi.RemoteException

---

init

```
public void init()
          throws java.rmi.RemoteException
```
Calls the init method of the superclass

Specified by:

init in interface sorcer.base.Provider

Throws:

java.rmi.RemoteException

---

getMainUIDescriptor

```
public net.jini.lookup.entry.UIDescriptor getMainUIDescriptor()
```
Create s UI descriptor to publish the UI

Specified by:

getMainUIDescriptor in interface sorcer.base.Provider

Returns:

returns the UI descriptor to be published

---

isValidTask

```
public boolean isValidTask()
```
Returns:

true if the task is valid

sorcer.provider.grid.dispatcher

Class GridDispatcherProviderImpl.Disco

```
java.lang.Object
  └
sorcer.provider.grid.dispatcher.GridDispatcherProviderImpl.Disco
```

Enclosing class:

GridDispatcherProviderImpl

---

public static class GridDispatcherProviderImpl.Disco

extends java.lang.Object

The class which looks up for jobber proxy from the lookup service

---

Constructor Summary
GridDispatcherProviderImpl.Disco() The constructor for Disco class

Method Summary	
sorcer.core.FileStorer	**getFileStorer**() Method finds the filestore provider
sorcer.core.Jobber	**getJobber**() Tries to lookup for

	the jobber proxy
sorcer.core.provider.autonomicprovisioner.AutonomicProvisioner	**getProvisioner**() Tries to lookup for the
void	**print**(int i, java.lang.Object obj)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

GridDispatcherProviderImpl.Disco

```
public GridDispatcherProviderImpl.Disco()
```
The constructor for Disco class

Method Detail

getJobber

```
public sorcer.core.Jobber getJobber()
```
Tries to lookup for the jobber proxy

Returns:

Returns the Jobber proxy if jobber is found

---

getProvisioner

```
public
sorcer.core.provider.autonomicprovisioner.AutonomicProvisioner
getProvisioner()
```

Tries to lookup for the

Returns:

Returns the AutonomicProvisioner proxy if AutonomicProvisioner is found

---

getFileStorer

```
public sorcer.core.FileStorer getFileStorer()
```

Method finds the filestore provider

Returns:

returns the proxy for filestore provider

---

print

```
public void print(int i,
                  java.lang.Object obj)
```

sorcer.provider.grid.dispatcher

Class GridDispatcherProviderImpl.DispatcherResult

```
java.lang.Object
  └
sorcer.provider.grid.dispatcher.GridDispatcherProviderImpl.DispatcherResult
```

Enclosing class:

GridDispatcherProviderImpl

---

public static final class GridDispatcherProviderImpl.DispatcherResult

extends java.lang.Object

Class which waits for the return of the results from Caller

Field Summary	
java.net.URL	**outputURL**

Constructor Summary
GridDispatcherProviderImpl.DispatcherResult(net.jini.core.event.RemoteEventListener callback)
GridDispatcherProviderImpl.DispatcherResult(java.lang.String resultFile)

Method Summary	
void	**doFinally**() Writes the final outputs to the file and notifies the callback
void	**done**(sorcer.base.ServiceContext ctx)

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

outputURL

```
public java.net.URL outputURL
```

Constructor Detail

GridDispatcherProviderImpl.DispatcherResult

```
public
GridDispatcherProviderImpl.DispatcherResult(java.lang.String resultFile
)
```

GridDispatcherProviderImpl.DispatcherResult

```
public
GridDispatcherProviderImpl.DispatcherResult(net.jini.core.event.RemoteE
ventListener callback)
```

Method Detail

done

```
public void done(sorcer.base.ServiceContext ctx)
```

---

doFinally

```
public void doFinally()
```
Writes the final outputs to the file and notifies the callback

sorcer.provider.grid.dispatcher

Class GridDispatcherProviderImpl.JobDispatcher

```
java.lang.Object
  └java.lang.Thread
      └
sorcer.provider.grid.dispatcher.GridDispatcherProviderImpl.JobDispatche
r
```

All Implemented Interfaces:

java.lang.Runnable

Enclosing class:

GridDispatcherProviderImpl

---

public static final class GridDispatcherProviderImpl.JobDispatcher

extends java.lang.Thread

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary
GridDispatcherProviderImpl.JobDispatcher(java.lang.String[] inputValues, int fromIndex, int toIndex, java.lang.String notify, GridDispatcherProviderImpl.JobsDispatcher disp)

GridDispatcherProviderImpl.JobDispatcher(java.lang.String[] inputValues, int fromIndex, int toIndex, java.lang.String notify, GridDispatcherProviderImpl.JobsDispatcher disp, javax.security.auth.Subject client)

Method Summary	
sorcer.core.ServiceTask	**getTask**(java.lang.String inputValue, int index) Creates a task from the caller context
void	**run**() Run method for the JobDispatcher thread

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

GridDispatcherProviderImpl.JobDispatcher

```
public
GridDispatcherProviderImpl.JobDispatcher(java.lang.String[] inputValues
,
                                         int fromIndex,
                                         int toIndex,

java.lang.String notify,

GridDispatcherProviderImpl.JobsDispatcher disp)
```

GridDispatcherProviderImpl.JobDispatcher

```
public
GridDispatcherProviderImpl.JobDispatcher(java.lang.String[] inputValues
,
                                         int fromIndex,
                                         int toIndex,

java.lang.String notify,

GridDispatcherProviderImpl.JobsDispatcher disp,

javax.security.auth.Subject client)
```

Method Detail

run

```
public void run()
```

Run method for the JobDispatcher thread

---

getTask

```
public sorcer.core.ServiceTask
getTask(java.lang.String inputValue,
                                    int index)
```

Creates a task from the caller context

Parameters:

inputValue - inputValues for the executable

index - index values to keep track of number of tasks created

sorcer.provider.grid.dispatcher

Class GridDispatcherProviderImpl.JobsDispatcher

```
java.lang.Object
  └java.lang.Thread
      └
sorcer.provider.grid.dispatcher.GridDispatcherProviderImpl.JobsDispatcher
```

All Implemented Interfaces:

java.lang.Runnable

Enclosing class:

GridDispatcherProviderImpl

---

public static final class GridDispatcherProviderImpl.JobsDispatcher

extends java.lang.Thread

Class for dispatching the job to the Jobber, internally used by JobDispatcher class

---

Field Summary

Fields inherited from class java.lang.Thread
MAX_PRIORITY, MIN_PRIORITY, NORM_PRIORITY

Constructor Summary

GridDispatcherProviderImpl.JobsDispatcher(java.lang.String[] inputValues, int jobSize, java.lang.String notify, GridDispatcherProviderImpl.DispatcherResult result)
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String[] inputValues, int jobSize, java.lang.String notify, GridDispatcherProviderImpl.DispatcherResult result, javax.security.auth.Subject client)
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String inputFile, int jobSize, java.lang.String notify, GridDispatcherProviderImpl.DispatcherResult result)
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String inputFile, int jobSize, java.lang.String notify, GridDispatcherProviderImpl.DispatcherResult result, javax.security.auth.Subject client)

Method Summary	
void	**done**(sorcer.core.ServiceJob resultJob) Keeps track of the number of jobs returned with the result, till all the results are obtained
java.net.URL	**getOutputURL**()
java.lang.String[]	**parseInputFile**()

	The method to parse input file to get all the input valuse contained in the file
void	**run**() The run method of the thread

Methods inherited from class java.lang.Thread
activeCount, checkAccess, countStackFrames, currentThread, destroy, dumpStack, enumerate, getContextClassLoader, getName, getPriority, getThreadGroup, holdsLock, interrupt, interrupted, isAlive, isDaemon, isInterrupted, join, join, join, resume, setContextClassLoader, setDaemon, setName, setPriority, sleep, sleep, start, stop, stop, suspend, toString, yield

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

GridDispatcherProviderImpl.JobsDispatcher

```
public
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String inputFile,
                                          int jobSize,
java.lang.String notify,
GridDispatcherProviderImpl.DispatcherResult result)
```

Parameters:

inputFile -

jobSize -

notify -

result -

GridDispatcherProviderImpl.JobsDispatcher

```
public
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String[] inputValue
s,
                                                    int jobSize,

java.lang.String notify,

GridDispatcherProviderImpl.DispatcherResult result)
```

Parameters:

inputValues -

jobSize -

notify -

result -

GridDispatcherProviderImpl.JobsDispatcher

```
public
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String inputFile,
                                                    int jobSize,

java.lang.String notify,

GridDispatcherProviderImpl.DispatcherResult result,

javax.security.auth.Subject client)
```

Parameters:

inputFile -

jobSize -

notify -

result -

client -

GridDispatcherProviderImpl.JobsDispatcher

```
public
GridDispatcherProviderImpl.JobsDispatcher(java.lang.String[] inputValue
s,
                                                    int jobSize,
java.lang.String notify,
GridDispatcherProviderImpl.DispatcherResult result,
javax.security.auth.Subject client)
```

Parameters:

inputValues -

jobSize -

notify -

result -

client -

Method Detail

run

```
public void run()
```

The run method of the thread

---

parseInputFile

```
public java.lang.String[] parseInputFile()
```
The method to parse input file to get all the input valuse contained in the file

---

done

```
public void done(sorcer.core.ServiceJob resultJob)
```
Keeps track of the number of jobs returned with the result, till all the results are

obtained

---

getOutputURL

```
public java.net.URL getOutputURL()
```

sorcer.provider.grid.dispatcher

Class GridDispatcherUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └sorcer.core.security.ui.SecureSorcerUI
                          └
sorcer.provider.grid.dispatcher.GridDispatcherUI
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class GridDispatcherUI

extends sorcer.core.security.ui.SecureSorcerUI

The Class which renders the S Grid service UI

See Also:

Serialized Form

Nested Class Summary	
static class	**GridDispatcherUI.DispatcherListener** DispatcherListener Class listens for the results

Nested classes inherited from class sorcer.core.security.ui.SecureSorcerUI
sorcer.core.security.ui.SecureSorcerUI.AuditThread

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container
java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary	
int	**setSubjectTries**

Fields inherited from class sorcer.core.security.ui.SecureSorcerUI
allowDelegation, auditor, cancelBtn, chbx, config, debug, getPasswdfld, getUsrnamefld, gotSubject, loggedPrincipal, loggedSubject, loginBtn, preparedProxy

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR, W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
GridDispatcherUI(java.lang.Object obj) Constructr for S Grid service UI

Method Summary	
void	**actionPerformed**(java.awt.event.ActionEvent ae) Invoked when an action takes place on the UI
void	**done**() Resets the GUI after the results are obtained
void	**recievedOutput**(java.lang.String output)

	Appends the results to the output
void	**setSubject**(javax.security.auth.Subject subj, sorcer.base.ServiceContext ctx) sets the subject of the Context

Methods inherited from class sorcer.core.security.ui.SecureSorcerUI
debug, getAuthUI, getPreparedProxy, instantiate, prepareProxy

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane, setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame
addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window
addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container
add, add, add, add, add, addContainerListener, applyComponentOrientation, areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component
action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale,

setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Field Detail

setSubjectTries

```
public int setSubjectTries
```

Constructor Detail

GridDispatcherUI

```
public GridDispatcherUI(java.lang.Object obj)
```

Constructr for S Grid service UI

Parameters:

obj - the proxy object obtained from the Lookup service

Method Detail

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent ae)
```
Invoked when an action takes place on the UI

Parameters:

ae - the action event caused by the action

---

setSubject

```
public void setSubject(javax.security.auth.Subject subj,
                       sorcer.base.ServiceContext ctx)
                throws sorcer.base.ContextException
```
sets the subject of the Context

Parameters:

subj - Subject which has logged in

ctx - The prepared context provided by the subject

Throws:

sorcer.base.ContextException

---

done

```
public void done()
```
Resets the GUI after the results are obtained

---

recievedOutput

```
public void recievedOutput(java.lang.String output)
```
Appends the results to the output

sorcer.provider.grid.dispatcher

Class GridDispatcherUI.DispatcherListener

```
java.lang.Object
  └
sorcer.provider.grid.dispatcher.GridDispatcherUI.DispatcherListener
```

All Implemented Interfaces:

java.util.EventListener, java.rmi.Remote,

net.jini.core.event.RemoteEventListener, java.io.Serializable

Enclosing class:

GridDispatcherUI

---

public static final class GridDispatcherUI.DispatcherListener

extends java.lang.Object

implements net.jini.core.event.RemoteEventListener, java.io.Serializable,

java.rmi.Remote

DispatcherListener Class listens for the results

See Also:

Serialized Form

Field Summary	
GridDispatcherUI	**ui**

Constructor Summary
GridDispatcherUI.DispatcherListener()
GridDispatcherUI.DispatcherListener(GridDispatcherUI pui)

Method Summary	
net.jini.core.event.RemoteEventListener	**getListener**() gets a listener for the results
void	**notify**(net.jini.core.event.RemoteEvent event) notifies when outputs are obtained

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

ui

```
public transient GridDispatcherUI ui
```

Constructor Detail

GridDispatcherUI.DispatcherListener

```
public GridDispatcherUI.DispatcherListener()
```

GridDispatcherUI.DispatcherListener

```
public GridDispatcherUI.DispatcherListener(GridDispatcherUI pui)
```

Method Detail

getListener

```
public net.jini.core.event.RemoteEventListener getListener()
                                                    throws
java.rmi.RemoteException
```

gets a listener for the results

Returns:

Returns a remote listener

Throws:

java.rmi.RemoteException

---

notify

```
public void notify(net.jini.core.event.RemoteEvent event)
            throws java.rmi.RemoteException
```

notifies when outputs are obtained

Specified by:

notify in interface net.jini.core.event.RemoteEventListener

Throws:

java.rmi.RemoteException

Package sorcer.security.permission

Class Summary	
MethodPermission	This class is used to provide permission to the Subject to invoke methods on the provides

sorcer.security.permission

Class MethodPermission

```
java.lang.Object
  └java.security.Permission
      └net.jini.security.AccessPermission
          └sorcer.security.permission.MethodPermission
```

All Implemented Interfaces:

java.security.Guard, java.io.Serializable

---

public class MethodPermission

extends net.jini.security.AccessPermission

implements java.io.Serializable

This class is used to provide permission to the Subject to invoke methods on the provides

Author:

Abhijit Rai

See Also:

Serialized Form

Constructor Summary
MethodPermission(java.lang.String method) Creates a method permission object for the passed method

Methods inherited from class net.jini.security.AccessPermission
equals, getActions, hashCode, implies

Methods inherited from class java.security.Permission
checkGuard, getName, newPermissionCollection, toString

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

MethodPermission

```
public MethodPermission(java.lang.String method)
```

Creates a method permission object for the passed method

Parameters:

method - Mehtod name for which the permission needs to be granted

Package sorcer.core.security.ui

Class Summary	
SecureSorcerUI	Provides the security for the ServiceUI.

sorcer.core.security.ui

Class SecureSorcerUI

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Window
              └java.awt.Frame
                  └javax.swing.JFrame
                      └sorcer.core.security.ui.SecureSorcerUI
```

All Implemented Interfaces:

javax.accessibility.Accessible, java.awt.event.ActionListener,

java.util.EventListener, java.awt.image.ImageObserver, java.awt.MenuContainer,

javax.swing.RootPaneContainer, java.io.Serializable, javax.swing.WindowConstants

---

public class **SecureSorcerUI**

extends javax.swing.JFrame

implements java.awt.event.ActionListener

Provides the security for the ServiceUI. Any UI which subclasses this class will inherit its security capabilities. The class does the following: - Prepares the proxy obtained from the lookup service to enable constraints such as Mutual Authentication, Integrity etc. - Provides a UI (AuthUI) for the User to Login and blocks (doesnt allow the service UI unless user is authenticated)

Author:

Abhijit Rai

See Also:

Serialized Form

Nested Class Summary	
protected class	**SecureSorcerUI.AuditThread** (Depricated) Inner class which helps create a Audit thread to start sending messages to Auditor ina different thread

Nested classes inherited from class javax.swing.JFrame
javax.swing.JFrame.AccessibleJFrame

Nested classes inherited from class java.awt.Frame
java.awt.Frame.AccessibleAWTFrame

Nested classes inherited from class java.awt.Window
java.awt.Window.AccessibleAWTWindow

Nested classes inherited from class java.awt.Container

java.awt.Container.AccessibleAWTContainer

Nested classes inherited from class java.awt.Component
java.awt.Component.AccessibleAWTComponent, java.awt.Component.BltBufferStrategy, java.awt.Component.FlipBufferStrategy

Field Summary	
protected boolean	**allowDelegation**
protected sorcer.core.Auditor	**auditor**
protected javax.swing.JButton	**cancelBtn**
protected javax.swing.JCheckBox	**chbx**
protected net.jini.config.Configuration	**config**
protected boolean	**debug**
protected javax.swing.JPasswordField	**getPasswdfld**

protected javax.swing.JTextField	**getUsrnamefld**
protected boolean	**gotSubject**
protected java.security.Principal	**loggedPrincipal**
protected javax.security.auth.Subject	**loggedSubject**
protected javax.swing.JButton	**loginBtn**
protected java.lang.Object	**preparedProxy**

Fields inherited from class javax.swing.JFrame
accessibleContext, EXIT_ON_CLOSE, rootPane, rootPaneCheckingEnabled

Fields inherited from class java.awt.Frame
CROSSHAIR_CURSOR, DEFAULT_CURSOR, E_RESIZE_CURSOR, HAND_CURSOR, ICONIFIED, MAXIMIZED_BOTH, MAXIMIZED_HORIZ, MAXIMIZED_VERT, MOVE_CURSOR, N_RESIZE_CURSOR, NE_RESIZE_CURSOR, NORMAL, NW_RESIZE_CURSOR, S_RESIZE_CURSOR, SE_RESIZE_CURSOR, SW_RESIZE_CURSOR, TEXT_CURSOR,

W_RESIZE_CURSOR, WAIT_CURSOR

Fields inherited from class java.awt.Component
BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

Fields inherited from interface javax.swing.WindowConstants
DISPOSE_ON_CLOSE, DO_NOTHING_ON_CLOSE, HIDE_ON_CLOSE

Fields inherited from interface java.awt.image.ImageObserver
ABORT, ALLBITS, ERROR, FRAMEBITS, HEIGHT, PROPERTIES, SOMEBITS, WIDTH

Constructor Summary
SecureSorcerUI()
SecureSorcerUI(java.lang.Object obj) The Constructor where Object is the obtained proxy.

Method Summary

void	**actionPerformed**(java.awt.event.ActionEvent e) Invoked when an action ocurs on the UI
protected void	**debug**(java.lang.String msg) debug method automatically looks for an Auditor (if not already found one) and sends the message to the Auditor
protected void	**getAuthUI**() Renders the Authentiaction UI
protected java.lang.Object	**getPreparedProxy**() Returns the prepared proxy as object
void	**instantiate**() Instantiates The following system properties and fields for the UI java.security.auth.login.config java.rmi.server.codebase javax.net.ssl.trustStore (set to the user.home)
protected void	**prepareProxy**(java.lang.Object obj) prepare proxy method is called by the constructor to prepare the proxy and sets the permissions to prepare the required proxy.

Methods inherited from class javax.swing.JFrame
addImpl, createRootPane, frameInit, getAccessibleContext, getContentPane, getDefaultCloseOperation, getGlassPane, getJMenuBar, getLayeredPane, getRootPane, isDefaultLookAndFeelDecorated, isRootPaneCheckingEnabled, paramString, processWindowEvent, remove, setContentPane, setDefaultCloseOperation, setDefaultLookAndFeelDecorated, setGlassPane,

setJMenuBar, setLayeredPane, setLayout, setRootPane, setRootPaneCheckingEnabled, update

Methods inherited from class java.awt.Frame

addNotify, finalize, getCursorType, getExtendedState, getFrames, getIconImage, getMaximizedBounds, getMenuBar, getState, getTitle, isResizable, isUndecorated, remove, removeNotify, setCursor, setExtendedState, setIconImage, setMaximizedBounds, setMenuBar, setResizable, setState, setTitle, setUndecorated

Methods inherited from class java.awt.Window

addPropertyChangeListener, addPropertyChangeListener, addWindowFocusListener, addWindowListener, addWindowStateListener, applyResourceBundle, applyResourceBundle, createBufferStrategy, createBufferStrategy, dispose, getBufferStrategy, getFocusableWindowState, getFocusCycleRootAncestor, getFocusOwner, getFocusTraversalKeys, getGraphicsConfiguration, getInputContext, getListeners, getLocale, getMostRecentFocusOwner, getOwnedWindows, getOwner, getToolkit, getWarningString, getWindowFocusListeners, getWindowListeners, getWindowStateListeners, hide, isActive, isFocusableWindow, isFocusCycleRoot, isFocused, isShowing, pack, postEvent, processEvent, processWindowFocusEvent, processWindowStateEvent, removeWindowFocusListener, removeWindowListener, removeWindowStateListener, setCursor, setFocusableWindowState, setFocusCycleRoot, setLocationRelativeTo, show, toBack, toFront

Methods inherited from class java.awt.Container

add, add, add, add, add, addContainerListener, applyComponentOrientation,

areFocusTraversalKeysSet, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getContainerListeners, getFocusTraversalPolicy, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, isFocusCycleRoot, isFocusTraversalPolicySet, layout, list, list, locate, minimumSize, paint, paintComponents, preferredSize, print, printComponents, processContainerEvent, remove, removeAll, removeContainerListener, setFocusTraversalKeys, setFocusTraversalPolicy, setFont, transferFocusBackward, transferFocusDownCycle, validate, validateTree

Methods inherited from class java.awt.Component

action, add, addComponentListener, addFocusListener, addHierarchyBoundsListener, addHierarchyListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addMouseWheelListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, createVolatileImage, createVolatileImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, firePropertyChange, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentListeners, getComponentOrientation, getCursor, getDropTarget, getFocusListeners, getFocusTraversalKeysEnabled, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getHierarchyBoundsListeners, getHierarchyListeners, getIgnoreRepaint, getInputMethodListeners, getInputMethodRequests, getKeyListeners, getLocation, getLocation, getLocationOnScreen, getMouseListeners, getMouseMotionListeners, getMouseWheelListeners, getName, getParent, getPeer, getPropertyChangeListeners, getPropertyChangeListeners, getSize, getSize, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, imageUpdate, inside, isBackgroundSet, isCursorSet, isDisplayable, isDoubleBuffered, isEnabled, isFocusable, isFocusOwner, isFocusTraversable, isFontSet, isForegroundSet, isLightweight, isOpaque, isValid, isVisible, keyDown, keyUp, list, list, list, location,

lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processHierarchyBoundsEvent, processHierarchyEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, processMouseWheelEvent, removeComponentListener, removeFocusListener, removeHierarchyBoundsListener, removeHierarchyListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removeMouseWheelListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, requestFocus, requestFocusInWindow, requestFocusInWindow, reshape, resize, resize, setBackground, setBounds, setBounds, setComponentOrientation, setDropTarget, setEnabled, setFocusable, setFocusTraversalKeysEnabled, setForeground, setIgnoreRepaint, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, size, toString, transferFocus, transferFocusUpCycle

Methods inherited from class java.lang.Object
clone, equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.awt.MenuContainer
getFont, postEvent

Field Detail

getUsrnamefld

```
protected javax.swing.JTextField getUsrnamefld
```

getPasswdfld

```
protected javax.swing.JPasswordField getPasswdfld
```

loginBtn

```
protected javax.swing.JButton loginBtn
```

cancelBtn

```
protected javax.swing.JButton cancelBtn
```

chbx

```
protected javax.swing.JCheckBox chbx
```

auditor

```
protected sorcer.core.Auditor auditor
```

loggedSubject

```
protected javax.security.auth.Subject loggedSubject
```

loggedPrincipal

```
protected java.security.Principal loggedPrincipal
```

preparedProxy

```
protected java.lang.Object preparedProxy
```

---

config

```
protected net.jini.config.Configuration config
```

---

debug

```
protected boolean debug
```

---

gotSubject

```
protected boolean gotSubject
```

---

allowDelegation

```
protected boolean allowDelegation
```

Constructor Detail

SecureSorcerUI

```
public SecureSorcerUI(java.lang.Object obj)
```

The Constructor where Object is the obtained proxy. The subclass must call super(obj) to enable this constructor to prepare proxy

Parameters:

obj - the proxy object obtained from the lookup service (obtained directly when ServiceUI is invoked from the browser)

SecureSorcerUI

```
public SecureSorcerUI()
```

Method Detail

debug

```
protected void debug(java.lang.String msg)
```

debug method automatically looks for an Auditor (if not already found one) and sends the message to the Auditor

Parameters:

msg - msg is the message to be sent to the Auditor

---

prepareProxy

```
protected void prepareProxy(java.lang.Object obj)
```

prepare proxy method is called by the constructor to prepare the proxy and sets the permissions to prepare the required proxy. Requires client.ServiceUIProxyPreparer field in the configuration file obtained from the codebase This field specify the constraints to be put in the proxy while preparation presently the class looks by default for "config/prepare-minimal.config" file in the specified codebase.

Parameters:

obj - the proxy object obtained from the lookup service (obtained directly when ServiceUI is invoked from the browser), this parameter is passed by the constructor to this method.

---

getPreparedProxy

```
protected java.lang.Object getPreparedProxy()
```
Returns the prepared proxy as object

Returns:

returns the preparpared proxy as the object

---

instantiate

```
public void instantiate()
```
Instantiates The following system properties and fields for the UI

java.security.auth.login.config java.rmi.server.codebase javax.net.ssl.trustStore (set to the

user.home)

---

getAuthUI

```
protected void getAuthUI()
                  throws java.lang.Exception
```
Renders the Authentiaction UI

Throws:

java.lang.Exception

---

actionPerformed

```
public void actionPerformed(java.awt.event.ActionEvent e)
```
Invoked when an action ocurs on the UI

Specified by:

actionPerformed in interface java.awt.event.ActionListener

Package jgapp.jaas

Class Summary	
PsswdLoginModule	Login module that checks a username and password.

jgapp.jaas

Class PsswdLoginModule

```
java.lang.Object
  └ jgapp.jaas.PsswdLoginModule
```

All Implemented Interfaces:

javax.security.auth.spi.LoginModule

---

public class PsswdLoginModule

extends java.lang.Object

implements javax.security.auth.spi.LoginModule

Login module that checks a username and password.

Constructor Summary
PsswdLoginModule()

Method Summary	
boolean	**abort**() Called if overall login failed to abort authentication process If the login succeded, then this method cleans up any saved state
boolean	**commit**() This is called if all logins succeeded.
void	**initialize**(javax.security.auth.Subject subject, javax.security.auth.callback.CallbackHandler callbackHandler, java.util.Map sharedState, java.util.Map options) Initializes the login module.
boolean	**login**() Attempt to log a user in
boolean	**logout**() Logout the user.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

PsswdLoginModule

```
public PsswdLoginModule()
```

Method Detail

initialize

```
public void initialize(javax.security.auth.Subject subject,
javax.security.auth.callback.CallbackHandler callbackHandler,
                       java.util.Map sharedState,
                       java.util.Map options)
```

Initializes the login module.

Specified by:

initialize in interface javax.security.auth.spi.LoginModule

Parameters:

subject - Subject to be logged in

callbackHandler - Callback handler for handling call backs

sharedState - shared state of the server

options - other options

---

login

```
public boolean login()
              throws javax.security.auth.login.LoginException
```

Attempt to log a user in

Specified by:

login in interface javax.security.auth.spi.LoginModule

Returns:

Returns whether the login was sucessful or not

Throws:

javax.security.auth.login.LoginException

commit

```
public boolean commit()
               throws javax.security.auth.login.LoginException
```

This is called if all logins succeeded.

Specified by:

commit in interface javax.security.auth.spi.LoginModule

Returns:

Returns true if the commit is succeded

Throws:

javax.security.auth.login.LoginException

---

abort

```
public boolean abort()
              throws javax.security.auth.login.LoginException
```

Called if overall login failed to abort authentication process If the login succeded, then this method cleans up any saved state

Specified by:

abort in interface javax.security.auth.spi.LoginModule

Returns:

Returns true if this method succeded, false if the loginModule shall be ignored

Throws:

javax.security.auth.login.LoginException

---

logout

```
public boolean logout()
               throws javax.security.auth.login.LoginException
```

Logout the user.

Specified by:

logout in interface javax.security.auth.spi.LoginModule

Returns:

Returns true if this method succeded, false if the loginModule shall be ignored

Throws:

javax.security.auth.login.LoginException

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree (Permission is granted.)

Abhijit Rai 4/28/05
Student Signature Date

Disagree (Permission is not granted.)

_____ _____
Student Signature Date