RECURSION SOFTWARE, Inc.

Software for Software Developers™

VOYAGER®

Grid Computing with Voyager

By Saikumar Dubugunta

Recursion Software, Inc.
September 28, 2005

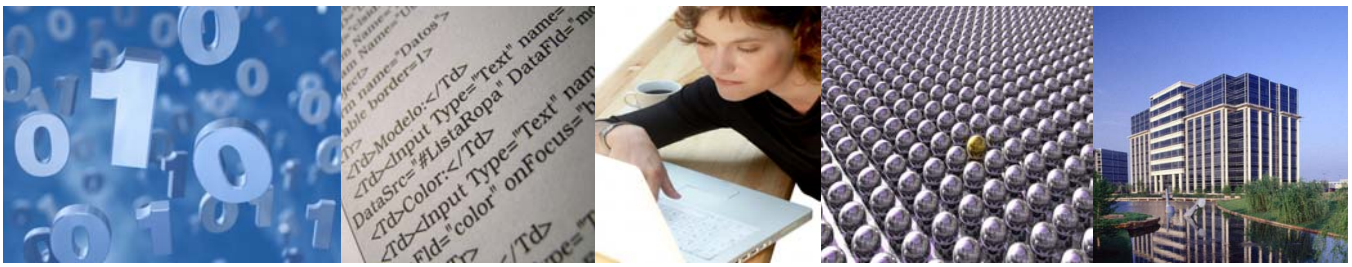


TABLE OF CONTENTS

Grid Computing with Voyager

By Saikumar Dubugunta

Introduction

Grid computing will be the norm of the future. Grid systems and applications aim to integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic "virtual organizations". The realization of this goal requires overcoming the numerous barriers that normally separate different computing systems within and across organizations. An effective and efficient grid system requires that computers, application services, data, and other resources can be accessed as and when required, regardless of physical location.

The fundamental requirements of a grid platform are:

- ***Global Name Space:*** Ability to locate and access services without regard to location or replication.
- ***Metadata Services***: Ability to find, invoke, and track entities.
- ***Data Services***: Efficient access and efficient movement of large quantities of data. Ability to serve data at a rate that achieves the desired service levels is an important aspect of a Grid.
- ***Authentication and Authorization***: Authentication mechanisms are required to establish the identity of individuals and services.
- ***Multiple Security Infrastructures***: Distributed operation implies a need to integrate and interoperate with multiple security infrastructures.
- ***High Availability***: High availability is often realized by expensive fault-tolerant hardware or complex cluster systems. Since grid technologies enable transparent access to a wider resource pool, across organizations as well as within organizations, they can be used as a building block to realize stable, highly reliable execution environments. In such a complex environment, policy-based autonomous control and dynamic mobility are keys to realizing systems that are highly flexible and recoverable.

- ***Disaster Recovery***: A grid must support mechanisms that promote quick and efficient recovery in case of a natural or manmade disaster in order to avoid long-term service disruption.

The Grid is not a monolithic system but will often be composed of resources owned and controlled by various organizations. A grid supports resource sharing and utilization across administrative domains, whether different units within an enterprise or even different institutions. A grid requires mechanisms to provide a context that can be used to associate users, requests, resources, policies, and agreements across organizational boundaries. Sharing resources across organizations also implies various security requirements.

Grids are moving from the obscurely academic to the highly anticipated norm of the future. One often hears about various grids such as: Compute Grids, Data Grids, Science Grids, Access Grids, Knowledge Grids, Bio Grids, etc.

Safe administration requires controlling access to services through robust security protocols and according to provided security policy. For example, obtaining application programs and deploying them into a grid system may require authentication and authorization. Also sharing of resources by users requires some kind of isolation mechanism. In addition, standard, secure mechanisms are required which can be deployed to protect grid systems while supporting safe resource sharing across administrative domains.

Ultimately the grid must be evaluated in terms of the applications, business value, and results that it delivers, not its architecture or physical form. Similarly the software platform that makes the realization of grid computing possible must be chosen based on applications, business values and expected results.

Using Voyager for Grid Computing

Voyager provides out-of-the-box support for these most important features necessary for grid applications. Voyager-based grids help system architects achieve the following enterprise objectives:

1) ***Resource Coordination without Centralized Control****:* Voyager integrates and coordinates resources and users that live within different control domains. The applications that interact with the resources and users are free to make dynamic choices about the execution of jobs within the grid.

2) ***Standard Protocols & Interfaces****:* Voyager APIs are simple and easy to use. Voyager supports interface-based-programming. Developers of objects are not required to know the final implementation of other objects with which they are interacting. Voyager supports SOAP/WSDL and a binary protocol. SOAP/WSDL makes Voyager applications open to other non-Voyager applications. Voyager offers interfaces that are used by system architects to address fundamental issues such as authentication, authorization, resource discovery, and resource access.
3) ***High Quality of Service****:* Voyager allows its constituent resources to be used in a coordinated fashion. It delivers various qualities of service relating, for example, to response time, throughput, availability, and security, and/or co-allocation of multiple resource types. Voyager meets complex user demands, so that the utility of the combined system is significantly greater than that of the sum of its parts.

Voyager – Core Components

Figure 1 shows the basic architecture and core components in Voyager. Voyager consists of the following components:

- **Name Servers (Directories)**: One or more name servers acting as a directory for routing requests from clients within the grid.
- **Code Servers**: One or more code servers (also referred to as resource servers) responsible for storing and distributing the application code. Application servers contact the code servers to get the code for specific objects that are being created.
- **Servers**: Servers are the machines in the grid that basically execute the application logic. The servers have access to the resources or are associated with the resources that are core to the services provided by the grid. The servers when deployed are composed of the Java Virtual Machine and Voyager. They are capable of downloading and executing any code as instructed by either a start up configuration or requests from clients. Servers contact the code servers to get the application code.
- **Clients**: Clients are user-facing or integrating applications that utilize the logic and resources present in the servers.

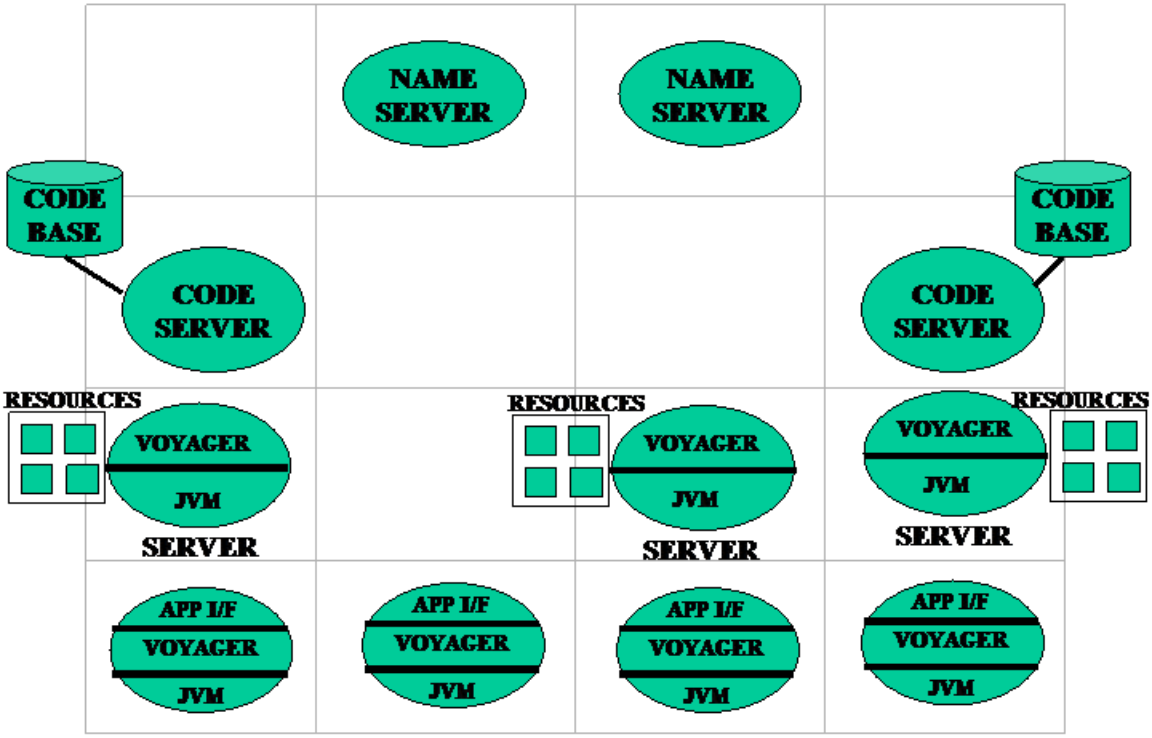


Figure 1: Voyager Core Components for Grid Computing

The different components can exist in the same network, different networks, and different geographically distributed networks. Voyager seamlessly connects the different components and allows application code to be location insensitive.

Code Distribution

Figure 2 shows the code distribution mechanism in Voyager. Development organizations submit code to the Voyager Code Servers for deployment purposes.

Application servers download the code from the Code Server. The code download is triggered in one of two ways:

- The location of a particular application is fixed by the deployment architecture. The application server is instructed during startup sequence (e.g., using a startup configuration file) to create a set of objects. The server downloads the necessary code at startup and instantiates the objects.
- A client determines that a particular job or sequences of jobs need to be executed on a particular server. The server is determined based on the logic in the client application about the necessary resources and response times. A policy server can also determine the location. Upon

determining the location, the client sends a request to the application server at that location to create the object.

The code server to be used in case of a replicated scenario is an application decision, usually controlled by a startup configuration file.

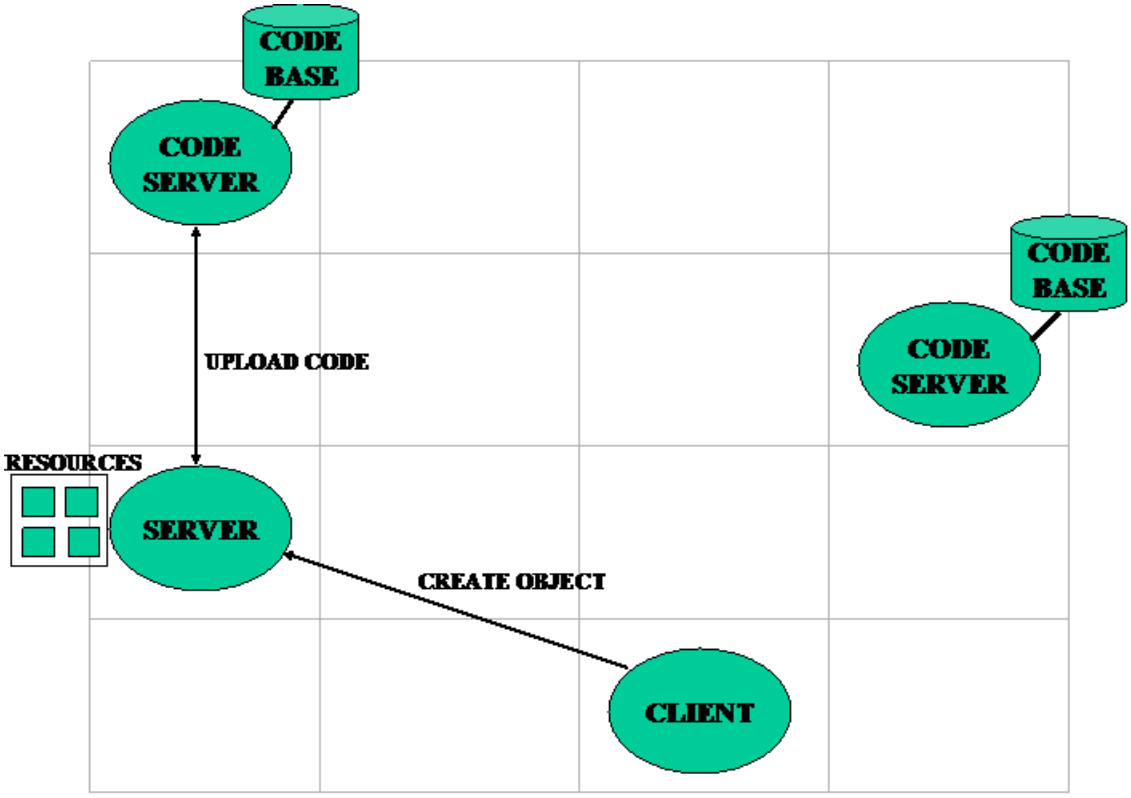


Figure 2: Voyager Code Distribution

The flexibility in the creation of service objects and execution of jobs offers tremendous advantages to the architects. The system architects can design services as:

- **Location Static**: Always exist and execute at the same location or server.
- **Pseudo Static**: Services come up on a given location. However, based on the service policies within the environment the service objects can remove themselves from one location at get created at a different location.
- **Location Fluid**: Service Objects get created in a more dynamic fashion. Their location is totally determined by the component requesting the service. (Please see discussion on mobility for a detailed description).

Fault Tolerant Directory

Voyager Name Server manages the global namespace. All service objects created in Voyager are registered with Voyager Name Server. This allows Voyager objects to access other Voyager objects transparently, subject to security constraints, without regard to location or replication.

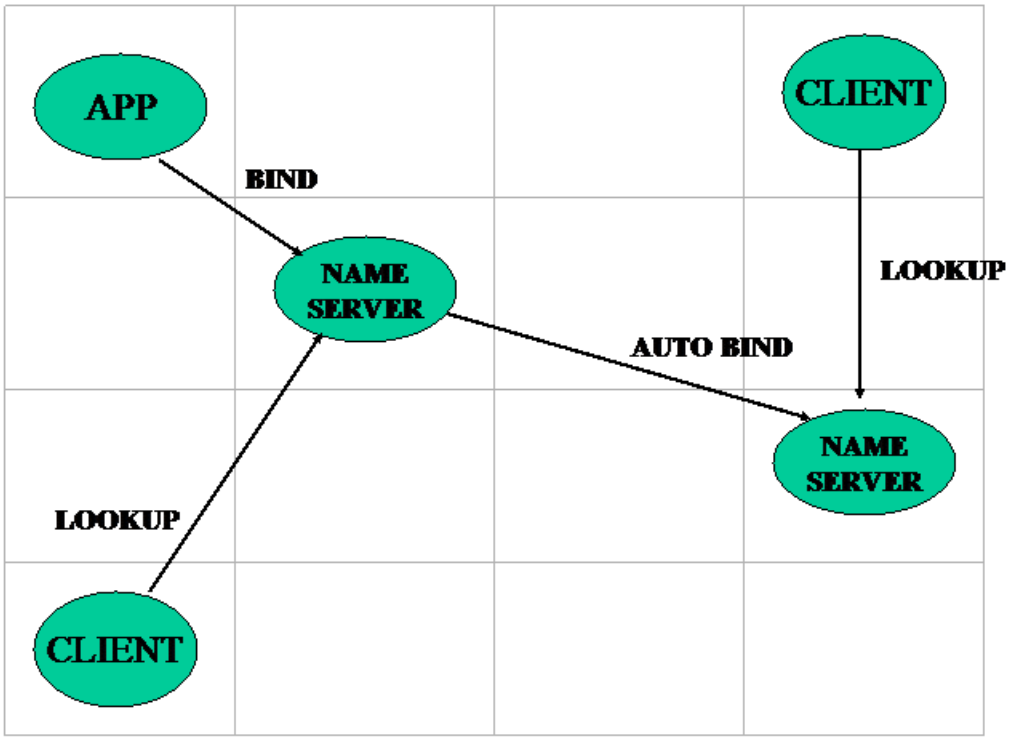


Figure 3: Fault Tolerant Directory

A Voyager client looking for a pre-created service object contacts the name server for location. The name server looks into the database and returns a response with the location of the object.

If the request object does not exist the client has the flexibility to use the mechanism described in **Figure 2** to decide a location and create the object at that location.

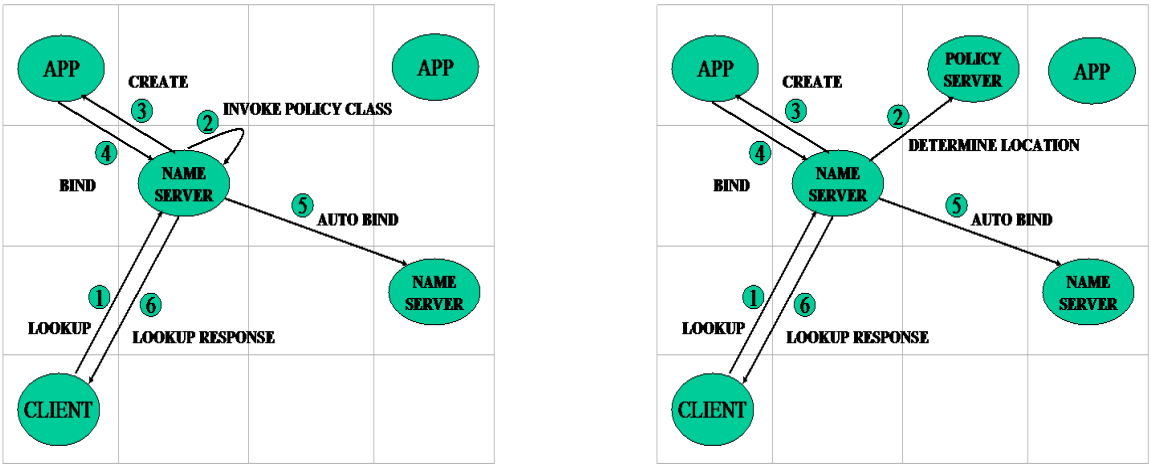
Voyager directory service can be run in fault tolerant redundant mode. In this mode two name servers are started at physically different locations. A creation of a service object on a server results in:

1) The service object is registered (BIND) with the primary name server for the location where service object is created.
2) The primary name server automatically registers the object in the fault tolerant name server.

The primary and secondary name servers are provided as part of the configuration for the application servers. For load balancing purposes, the primary and secondary can be swapped for different application servers. Registration of an object on name server causes the object to be automatically registered with the redundant name server.

Similarly the primary and secondary name servers for different clients can be set different to balance the load.

A key aspect of Voyager is the ability of policies to be embedded in the name server.



(A) (B)

Figure 4: Policy Based Location

Figure 4 (A) shows the dynamic creation of an object in an application server based on policies. The following is a description of the various steps:

1) Client sends a lookup request to the name server for a service object.
2) The name server cannot find service object in its database. Name server invokes the policy class to get the location of the service object.
3) The policy class determines the location at which the service objects needs to be instantiated. The policy class creates the object in the application server.

4) The application server registers the object in the name server.
5) The name server registers the object in the redundant name server.
6) The original name server where the lookup request was received sends a response to the client.

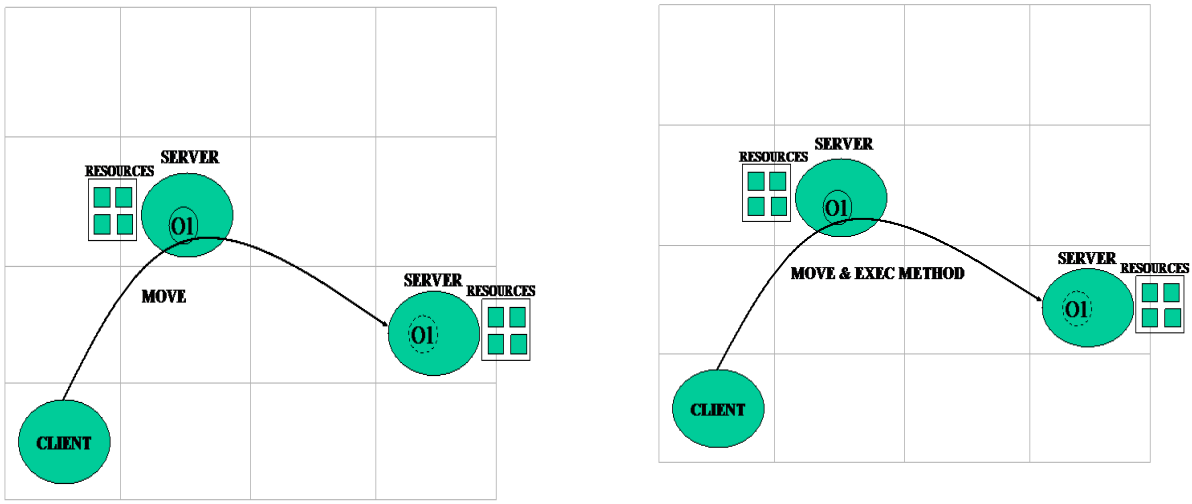
Figure 4 (B) shows a similar sequence of operations. However, in this realization of policy-based location the policy class actually contacts a policy server to determine the location. The policy server determines the location based on enterprise specific guidelines based on network bandwidth, CPU utilization, proximity to the storage network, service level assurance requirements for the service, etc.

It is also possible to create multiple instances of the same object for load balancing purposes. The policy classes embedded in the name server make the decision on the object to be returned when a lookup is performed.

Exploiting Grid Resources with Voyager Mobility

The premise of a grid is the ability of the services in the grid to effectively utilize the available resources. Resources include network bandwidth, processing power and storage throughput. Static location of services at deployment time cannot guarantee effective utilization of resources. Static location also cannot overcome unplanned downtime or failure of physical resources. It is important to have an underlying service in the platform that can relocate applications dynamically.

Voyager mobility makes it possible to dynamically relocate applications across the grid without disrupting other services.



(A) (B)

Figure 5: Voyager Mobility. (A) Client requests server to move object**. (B)** Client requests server to move object and immediately execute a method.

Mobility Scenarios

Figure 5 shows two mobility scenarios using Voyager. The following is a brief description.

In the first scenario, the client is requesting the server to move one of its objects to a different location. The object is identified by its reference that is already in the client because of an earlier lookup. The criterion for moving the object is application dependent. For example, in high volume transactions the client application can determine that the response time from the server is not within limits after a few requests. The client can contact a policy server to identify a new location. The client can then move the object to the new location. Moving the object will retain the state of the object as opposed to creating a new object on a different server.

In the second scenario, the client is instructing a move followed by an immediate execution of a method. This causes the object to move to the new server and execute the method in its own thread of control. The object terminates after the completion of the operation.

Abstraction of Implementation

In a grid scenario, there are potentially large numbers of application processes that are running in a synchronized manner to achieve the enterprise goals. It is not advisable to bring all applications down for upgrading or versioning. Another important requirement in grid computing is the ability to create/develop client applications (or peer applications) without depending on the implementation of the server applications (or service classes). This is possible only when the client applications work with interfaces and not with the final server classes.

Typical implementations require the generation of proxy classes which are used on the client side to perform marshalling of data, exception processing etc. The patented dynamic proxy generation feature of Voyager makes this straightforward.

Figure 6 shows the dynamic proxy generation process in Voyager.

1) The client application is developed to an interface. The client requests a reference to a service object in the application server that implements a given interface. The client supplies the name of the object, but not the class, as part of the request and the interface. (The client interacts with the name server before contacting the application server, not shown in the figure for simplicity).
2) The server application returns an object reference to the object. The object reference contains the implementation details of the class including the class, the methods in the class, exceptions thrown etc.
3) The voyager on the client side generates a proxy class dynamically using the information received from the server. The proxy class also implements the same interface. The interface is returned to the client application.

The client application uses the interface to invoke services on the object.

Figure 6 shows two objects O1 and O2 implementing the same interface I but of different classes C1 and C2. The client communicates with both objects using the same interface. The client is totally abstracted from the implementation of the interface in the two servers.

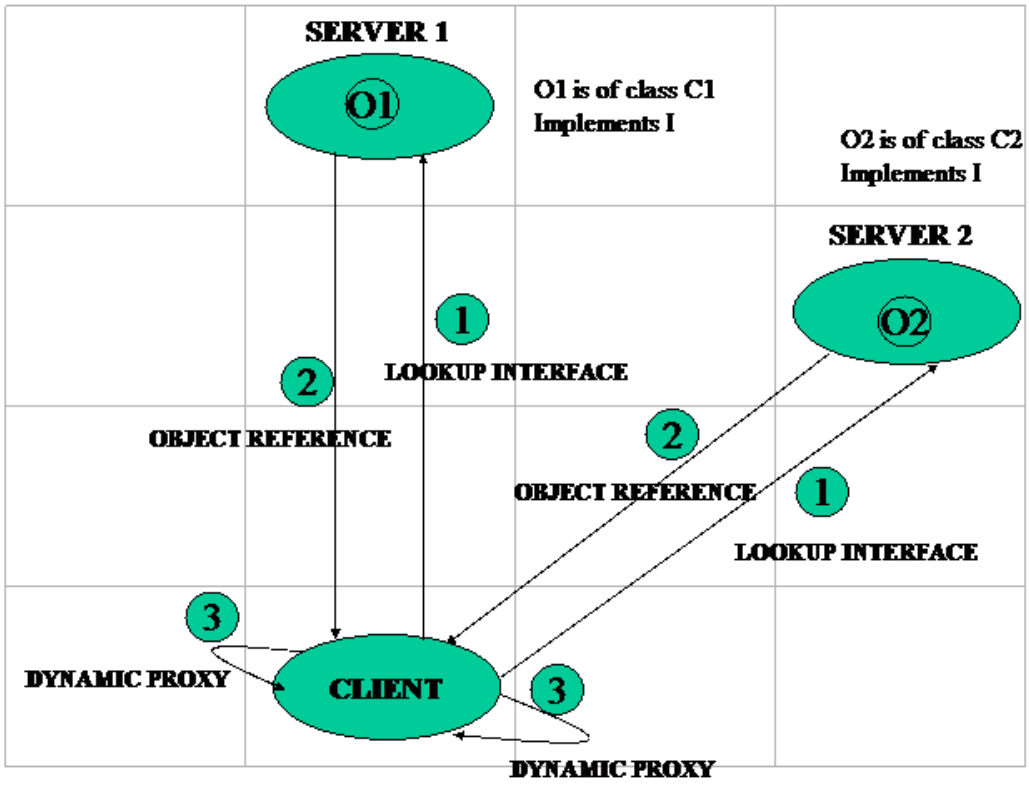


Figure 6: Dynamic Proxy Generation

Security

Security is pre-defined in enterprises today. The Voyager platform has security classes that can be implemented and plugged in, providing an excellent security framework. Security ACLs are resources that are already in the grid. The ability to utilize existing security infrastructure is an important aspect on any grid platform.

Figure 7 shows two methods of plugging in security into Voyager.

- In the first method a Voyager server contacts a security server to validate a request before it is executed.
- In the second method the server executes the security class locally to arrive at a decision.

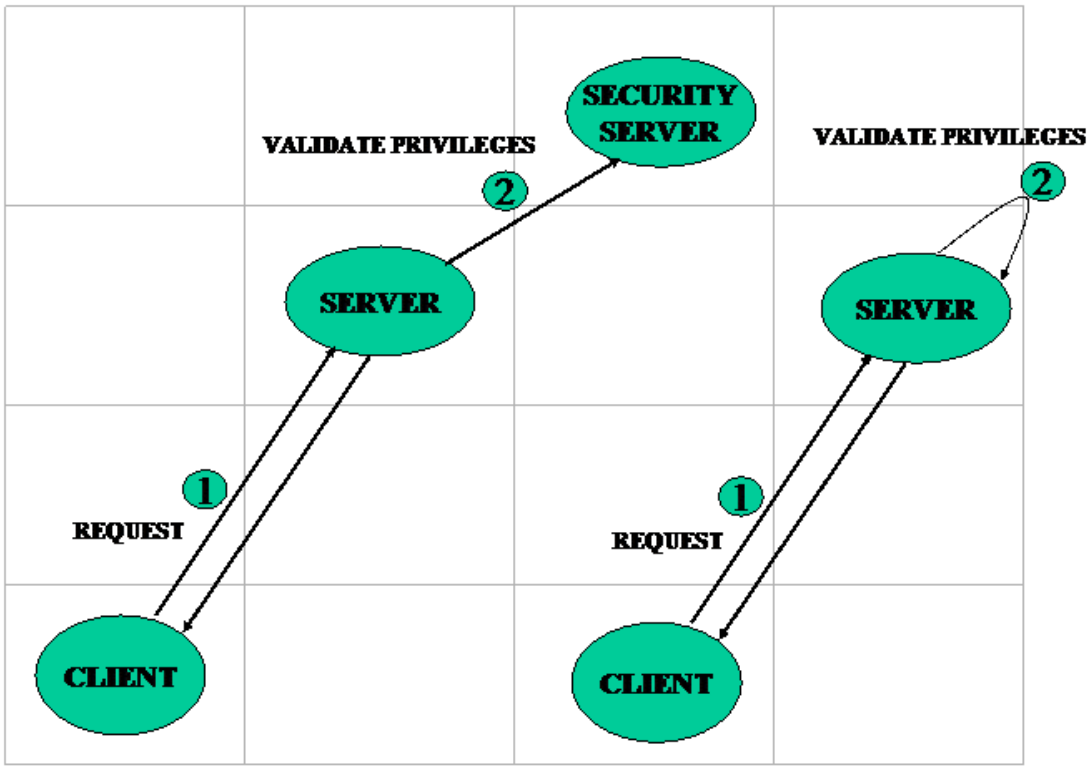


Figure 7: Voyager Security. Architect options: (A) Voyager server contacts a security server to validate a request or **(B)** Server executes a security class locally**.**

The two methods of security offer great flexibility to system architect in designing security into Voyager applications. Architects can choose local validation for requests with low security or static security definitions whereas requests that require higher degree of security or have dynamic security definitions can be validated by contacting a security policy server.

Voyager security provides a solution to the main grid problem of integrating multiple security infrastructures. Different domains of the grid are under different policies and so are the applications that are part of those domains. The ability to connect to different security servers from different applications helps bridge and merge different security infrastructures.

Voyager Disaster Recovery

Ability to recover quickly from a downtime is the most important requirement of a grid environment. In a disaster recovery scenario, the platform has to deal with the unavailability of certain key resources. This requires a movement of the services between locations within a domain or across domains to achieve the quality of service objectives. Disaster recovery is seamless in Voyager. The following features in Voyager provide system architects the necessary tools to implement disaster recovery:

- **Code Distribution**: Application code can be easily and quickly downloaded to a new location.
- **Remote Instantiation**: A disaster recovery application can determine the new location for static services and trigger instantiation of failed static objects.
- **Policy Based Location**: Policies in the name server can decide the new location for failed objects.
- **Mobility**: Objects that are running without failure can be moved to new locations to guarantee required quality of service of the overall Grid.

Management Console

The Voyager management console allows the applications on Voyager to be monitored and managed. The management console features allow users to:

- Discover service objects that are running in Voyager
- Change the location of objects
- View the event/trace logs created by service objects
- Add code to the code servers remotely
- Configure applications and application security

Summary

Grid computing will be the norm of the future. Grid systems and applications integrate, virtualize, and manage resources and services within distributed, heterogeneous, dynamic “virtual organizations”.

High quality, high performance grid applications require a platform than can support: code distribution, dynamic object instantiation, mobility, fault tolerant directory and code servers, dynamic proxy generation, flexible security and management console.

Ultimately the grid must be evaluated in terms of the applications, business value, and results that it delivers, not its architecture or physical form. Similarly the software platform that makes the realization of grid computing possible must be chosen based on applications, business values and expected results.

Voyager supports all the above features via user-friendly APIs and console administration. Voyager grids are advanced and make grid application development, deployment and maintenance an easy task.

Keywords: Grid computing, Distributed development, Dynamic Proxy Generation, Voyager, Grid Security

About the author

Saikumar Dubagunta has extensive experience in the development and deployment of software tools and applications developed in various languages and architectures. Mr. Dubagunta has a long history of holding senior level positions and has founded two successful software companies. He is currently the founder and president of Appera Software Inc. a subsidiary of Recursion Software Inc. Previously he held the position of Vice President of Engineering at Advanced Storage Array Products (ASAP), a storage management company. Preceding this, Mr. Dubagunta was co-founder, Chief Technologist, and Vice President of Engineering of Trigon Technology Group Inc. and was instrumental in the successful growth and merger of Trigon with Vertel Corporation. He then became Vice President of World Wide Professional services for Vertel Corporation, where he was responsible for the development of custom applications for worldwide customers in the integration of Telecommunications Applications. Mr. Dubagunta has an MS in Computer Science from Texas A&M University and a BS in Computer Science from Birla Institute of Technology and Science, India.

VOYAGER®

Recursion Software, Inc.
2591 North Dallas Parkway
Suite 200
Frisco, Texas 75034
1.800.727.8674 or 972.731.8800
www.recursionsw.com

RECURSION SOFTWARE, Inc.