

A Distributed Agent-based Architecture for Dynamic Services

Harvey B. Newman, Iosif C. Legrand, Julian J. Bunn

Charles C. Lauritsen Laboratory of High Energy Physics
California Institute of Technology
Pasadena, CA 91125, USA

Abstract

A prototype system for agent-based distributed dynamic services that will be applied to the development of Data Grids for high-energy physics is presented. The agent-based systems we are designing and developing gather, disseminate and coordinate configuration, time-dependent state and other information in the Grid system as a whole. These systems are being developed as an enabling technology for workflow-management and other forms of end-to-end Grid system monitoring and management. This prototype is being developed in Java and is based on the JINI support for distributed applications.

Keywords: Distributed Services, JINI, Mobile Agents, Self-Organizing Neural Networks

1 Introduction

The aim of this prototype system is to provide an information management framework as a flexible and reliable distributed service system, applied for specific data processing tasks in HEP [1], [2]. Currently used programs and tools can be interfaced to become network services in this framework. The code mobility paradigm (agents or proxy-server) extends the client-server (or the remote procedures call) approach to accessing and analyzing information to distributed systems. Several potential advantages of this paradigm are: asynchronous communication and disconnected operation, remote interaction and adaptability, parallel execution and autonomous mobility.

This prototype design is based on a Station Server unit, which is a generic network server that can host Dynamic Services. The Station Servers are dynamically interconnected (peer-to-peer) and provide a distributed framework for hosting services. In our prototype, the Station Server implementation is made available as a network service using JINI [3] technology. This framework should allow cooperating services and applications to access each other seamlessly, to adapt to a dynamic environment, and to share code and configurations transparently. This system design avoids single points of failure, allows service replication and re-activation and aims to offer reliable support for large scale distributed applications in real conditions, where individual (or multiple) components may fail. The prototype services implemented within this design use self-learning algorithms that are able to dynamically optimise the workflow that can be handled by a distributed set of Regional Centers. These services are integrated and interact with others that gather and disseminate information on the available resources, within the JINI-based framework.

2 Why JINI?

JINI extends the Java Platform providing support for Distributed Applications. JINI is a set of Java classes (APIs) and services within a distributed computing framework [3]. The purpose of the JINI architecture is to *federate* groups of software components into a single, dynamic distributed system. The major features which make JINI technology very attractive for this project are:

Lookup Discovery 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.

Leasing Mechanism: 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.

Remote Events: 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.

Transactions Manager: Reliable distributed object models require transaction support to aid in protecting the integrity of the resource layer. The specified transactions are inherited from the Jini programming model and focus on supporting large numbers of heterogeneous resources, rather than a single large resource (database). This service provides a series of operations, either within a single service or spanning multiple services that can be seen as distributed atomic procedures having a two-phase commit.

The JavaSpaces Service: This communication mechanism was heavily influenced by the concept of a Tuple space that was first described in 1982 in a programming language called Linda [3][4]. Distributed programs, unaware of each other's existence, can communicate by releasing data (a tuple) into a persistent space. Programs read, write, and take such entries from the Tuple space that are of interest to them.

3 The Station Server

A Station Server is a network service that can host Dynamic Services. Station Servers are dynamically interconnected (peer-to-peer) providing a distributed framework for Services. The Station Server implementation is made available as a network service using the JINI technology.

Each Station Server registers itself with a set of lookup-servers, and at the same time sets itself up as a remote listener for any state-modifications (transition events) in other Station Servers. This allows each Station Server to keep an updated dynamic list of active Station Servers. Each Station Server gets a proxy for all the other Station Servers. The JINI lease mechanism allows one Server to automatically inform the other Servers (using remote event notification) of changes that may occur in other services, or of any network problems that occur.

Dynamic Services are hosted by the network of Station Servers and made available to "interested" clients. This framework allows services to access the information they require from the entire system and also to interact with other services. Searching for services that match a set of attributes, and registering for remote notification if the system's state changes is part of the JINI lookup discovery service. This web of Station Servers is used to host a set of Dynamic Services. In Figure 1 we represent schematically the way this network of Station Servers interacts, and the way updates are done. In addition, remote event notification is used to support communication between the dynamic services hosted by Station

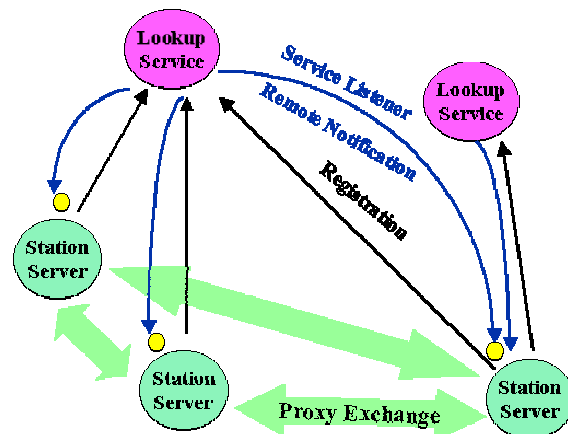


Figure 1. Interconnecting Station Servers and the mechanism to keep such a network updated.

Servers, as well as their management. Subscription for remote events follows same leasing mechanism, and requires periodic updates. Remote event subscription, which allows a service to register for transition events which are not known by the system at the time of registration, are used to notify the interested Station Servers when a new attribute for a service becomes available.

The Station Servers framework provides support for three distributed computing conceptual entities:

Dynamic Services are hosted by the network of Station Servers and make them available to interested clients. This framework allows services to access information from the entire system and to interact with other services. The Station Server does the service management and facilitates the inter-service communication.

Mobile Agents are dynamic services, which can move between Station Servers to perform a certain task (a good repository for agents development can be found at the UMBC AgentWeb page [5]). The interconnected Station Server network allows one to use autonomous mobile agent applications that migrate between the servers. Agents may interact synchronously or asynchronously using the Station Servers support for “roaming” and for a message mailbox. In our approach we intent to keep the agent structure small and simple, but still able to dynamically use information and computational services provided by the Station Servers framework.

“Smart” Proxies are “flexible” services which are deployed to interested clients and services, and act differently at different sites according to the Proxy’s rule base. This rule base includes a set of local and remote parameters, so it can adapt to the surrounding environment and the client’s characteristics, to provide the required functionality for the client in the most effective way (from a global perspective). Possible applications of these proxies include providing efficient access to data by choosing one of several replicas, or balancing the computational load between the client and server sites, depending on the available resources.

The types of components summarized above work together and interact in a distributed environment by using remote event subscription / notification and synchronous and asynchronous message based communication. Code mobility is also required in providing such functionality.

Any additional “mobile code” that may be needed by these components is either loaded from an http server or from a JavaSpace service.

4 Job Scheduling

As an example application of the JINI-based framework, we considered the problem of optimally distributing offline data processing jobs among the worldwide-distributed Regional Centers in the distributed computing model to be used for the LHC [1]. Data processing jobs in this model need efficient (local) access to large amounts of data in addition to substantial computing resources. The job scheduling distributed service provides the mechanism to identify the “best” regional center for the execution of a given job, based on: the computing resources available, the effective cost of shipping data there over wide and local area networks of limited bandwidth, policy information on the relative priority of different tasks, and other characteristics of the center and its state. Once a data processing task (or subtask) is assigned to a Regional Center site, the way it is split in pieces, and distributed to the available processing nodes is done by the local batch queuing system, which is also seen as a network service.

Such a scheduling system may also help manage the way data are distributed or replicated among regional centers as a function of time, making it capable of providing useful information for the establishment and execution of data replication policies.

Each Center starts a Station Server and the Server network is created through the mechanism described earlier. The Station Server needs to host a set of local services such as: system and application monitoring services, the local batch queuing system and the local databases index catalog. When a Regional Center considers exporting a job to another center, it sends out a request to all available centers for an estimate of the time to complete the job. Based on the replies it receives

within a certain time window, and other state and/or policy information, it assigns a “costs” to all connected centers (Figure 2), and this is then used to choose the destination to which a job (and/or the associated data) is to be exported.

We intend to test a Self Organizing Neural Network Scheduling system (SONN) [3] in this framework. The SONN is a network service able to learn from previous experience, to cluster correlated information using a competitive learning algorithm and to improve on the scheduling decisions. This scheme does not have a centralized decision point, and the decisions are done at each center based on “negotiations”

with the other partner services.

A “thin” proxy for this service is exported to remote sites, but the SONN evaluation is done locally, using direct access to other local services. If a site offers an expected “cost” for the remote execution of the job, its description object is transferred to the remote site using a transaction manager and a proxy to the job progress agent is returned. This provides the mechanism for the originating site to control the job execution at a remote

site and to monitor its progress. Once the job is executed the information may be used to improve the knowledge of the data access patterns, as well as the learning procedure used to evaluate further jobs.

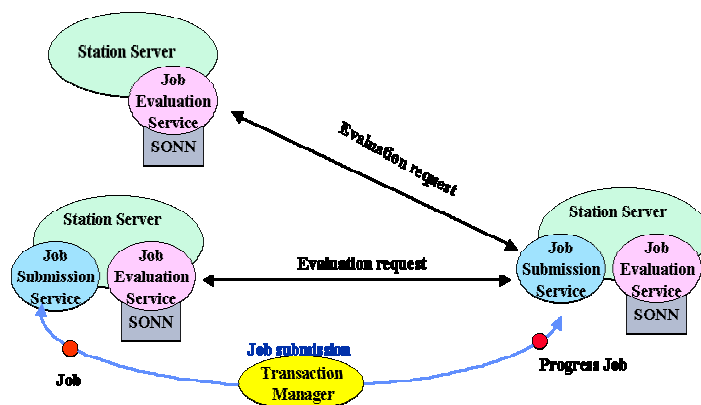


Figure 2 A schematic view of the distributed job scheduling system.

5 Current Status and Outlook

This prototype system is under continuing development. The results so far are very promising. Integration with simple monitoring services on the network of distributed Station Servers has been successfully tested. The lookup discovery mechanism, the transaction manager service, the Javaspaces and remote event notifications system were all successfully tested in this framework over a wide area network. Exporting dynamically complex proxies for services including dedicated GUIs works well. The leasing mechanism, which keeps the distributed service network alive and updated, works correctly in case of service(s) or network failure. Our experience so far has shown that JINI, based on open standards, is an effective technology for building complex distributed services such as those required for an LHC Data Grid system, with a moderate investment in software development manpower.

References

- [1] MONARC Project: <http://www.cern.ch/MONARC>
- [2] The Anatomy of the Grid: Enabling Scalable Virtual Organizations, I. Foster et al, to be published in Intl. J. Supercomputer Applications, 2001
- [3] Jini: <http://www.sun.com/jini> ; <http://www.jini.org>
- [4] Tuple Spaces: <http://www.almaden.ibm.com/cs/Tspaces/>
- [5] UMBC AgentWeb: <http://agents.umbc.edu/about.shtml>
- [6] A Self-Organizing Neural Network for Job Scheduling in Distributed Systems, Harvey B. Newman, Iosif C. Legrand, CERN-CMS NOTE 2001/009