ENABLING PEER-TO-PEER INTERACTIONS ON THE GRID

by

VINCENT MATOSSIAN

A Thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Master of Science

Graduate Program in Electrical and Computer Engineering

Written under the direction of

Professor Manish Parashar

and approved by

_____

_____

_____

New Brunswick, New Jersey

May, 2003

ABSTRACT OF THE THESIS

Enabling peer-to-peer interactions on the Grid

by Vincent MATOSSIAN

Thesis Director: Professor Manish Parashar

The emergence of computational Grids and the potential for seamless aggregation and interactions between distributed services and resources, has made it possible for scientists and engineers to conceive a new generation of realistic scientific and engineering simulations of complex physical phenomena. These applications will symbiotically and opportunistically combine computations, experiments, observations, and real-time data, and will provide important insights into complex systems. P2P and Grid communities are actively working on deploying and standardizing the infrastructure, protocols, and mechanisms, to support the decentralized interactions across distributed resources required by such applications. The underlying infrastructure will enable new classes of applications based on continuous, seamless and secure interactions, where the application components, Grid services, resources and data interact as peers. The Grid is rapidly emerging as the means for coordinated resource sharing and problem solving in multi-institutional virtual organizations while providing dependable, consistent, pervasive access to global resources. The Grid community and Global Grid Forum are developing and deploying standard services and middleware infrastructures that enable seamless and secure discovery, access, allocation, and scheduling of resources in these virtual organizations. These services and middleware infrastructures address the need for ubiquity of heterogeneous resource assemblies and come at a time when scientific projects regularly span across countries and continents, giving scientists the ability to conduct truly global research. This rapidly expanding infrastructure currently lacks

application-level services that would enable peer-to-peer interactions across peers and peer domains.

This thesis presents the design, implementation and evaluation of Pawn, a peer-to-peer messaging framework that builds on the JXTA protocols to support the interaction and associated messaging semantics required by these Grid applications.

Acknowledgements

My warmest recognition goes to my brother Leonardo and my mother Olga, who have supported me in many ways in my effort to open up to the world, and have so genuinely understood my paradoxal need to part in order to find myself. This document would have never come to existence without the support, guidance and help from my research advisor, Dr Manish Parashar. I wish to thank all my friends and members of the TASSL Laboratory, and especially Vijay Mann for his helpful advice and stimulating discussions. To my friend Phillip Stanley-Marbell who (maybe unwillingly) helps me understand and reflect on the difference between those who seem to be, and those who really are. Thank you to CAIP and all its staff, who have always promptly helped me resolve both administrative and systems problems.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

The emergence of computational Grids [3] and the potential for seamless aggregation and interaction between services and resources, has made it possible for scientists and engineers to conceive a new generation of realistic scientific and engineering simulations of complex physical phenomena. These applications will symbiotically and opportunistically combine computations, experiments, observations, and real-time data, and will provide important insights into complex systems such as interacting black holes and neutron stars, formations of galaxies, subsurface flows in oil reservoirs and aquifers, and dynamic response of materials to detonations. However, such global scientific investigation require continuous, seamless and secure interactions where the application components, services, resources (systems, CPUs, instruments, storage) and data (archives, sensors) can interact as peers. For example, consider the oil production process; the process involves (1) sophisticated reservoir simulation components that encapsulate complex mathematical models of the physical interaction in the subsurface and execute on distributed computing systems on the Grid, (2) Grid services that provide secure and coordinated access to the resources required by the simulations, (3) distributed data archives that store historical, experimental and observed data, (4) sensors embedded in the instrumented oilfield that provide real-time data about the current state of the oil field, (5) external services that provide current weather information or economic data to optimize production or profit, and (6) the scientists, engineers and other experts in the field, in the laboratory and in the offices. Furthermore, these entities need to dynamically discover and interact with one another as peers to achieve the overall application objectives. Simulation components interact with Grid services to dynamically obtain necessary resources, detect current resource state, and negotiate required quality of service. The components interact with one another to accurately

model the underlying physical phenomenon, and with data archives and real-time sensor data to enable history matching, dynamic data injection, and data driven adaptations (e.g. to detect bypassed oil). They may interact with other services on the Grid, for example, with optimization services to optimize well placement, with weather services to control production, and with economic services to detect current oil prices so as to maximize revenue. Finally the experts (scientists, engineers, and managers) collaboratively access, monitor, interact with, and steer the simulations and data at runtime to drive the discovery process.

In a loosely-coupled computing environment such as the Grid, large-scale collaborations[1] require communication models and network architectures that allow the system to be flexible, scalable, reliable, and manageable while enabling transparent information access. Software systems typically provide these capabilities through a middleware[11] which defines the set of possible interactions that can take place between interconnected tasks/services distributed over a network. A messaging middleware is one kind of middleware, which handles interactions between entities by defining messages that encapsulate the type of interaction and associated set of parameters. Messages in such a system are exchanged between services running on peers. Each service can be uniquely identified and addressed using an advertised interface. The responsibility of the messaging middleware is to provide guarantees on the delivery of messages between a source and a destination. Every message has to carry enough information in order for the receiving end service to (1) authenticate the requester of its execution, (2) obtain a return address for the response, (3) process the message payload. The solutions provided by messaging middleware systems are typically closely tied to a specific problem definition; however, the expansion and need for global simulations, as presented by the Grid[3], has set new requirements that call for interoperable, scalable, and open middleware interactions.

The Grid is rapidly emerging as the means for coordinated resource sharing and problem solving in multi-institutional virtual organizations while providing dependable,

[1]Throughout this thesis, we define collaboration interactions between peers, as persons or compute nodes

consistent, pervasive access to global resources. The Grid community and Global Grid Forum[7] are developing and deploying standard services and middleware infrastructures that enable seamless and secure discovery, access, allocation, and scheduling of resources in these virtual organizations. These services and middleware infrastructures address the need for ubiquity of heterogeneous resource assemblies and come at a time when scientific projects regularly span across countries and continents, giving scientists the ability to conduct truly global research. This rapidly expanding infrastructure currently lacks application-level services that would enable peer-to-peer interactions across peers and peer domains.

In this thesis, we present Pawn, a peer-to-peer messaging substrate that builds on the emerging Grid middleware to enable the peer-to-peer interactions required for distributed computational systems targeting scientific investigation. Combined to the core services offered by the Grid community, the application-level peer-to-peer interactions defined by the Pawn messaging substrate will enable dynamic data exchange and automatic event triggering, setting the direction towards an autonomic Grid computing framework.

1.2 Overview of the Pawn Peer-to-Peer Messaging Framework

Pawn is a messaging framework designed with the requirements of large-scale scientific collaboratories in mind.

Pawn provides key messaging mechanisms to enable monitoring, computational interaction, steering, dynamic data interrogation and injection, and collaboration between peers on the computing Grid. It provides mechanisms for publishing and subscribing-to information by interest (message type) while providing message filtering and aggregation. Pawn builds on Project JXTA [16], DISCOVER [10][9], and Globus [4] middleware services and enables dynamic peer-to-peer discovery of applications, resources and users peers while enabling interactions between them. In JXTA, interactions are initiated by advertisements published by peers. Advertisements are platform independent messages formatted in the eXtensible Markup Language (XML) [24]. These advertisements are

dynamically discovered by relevant peers based on their interests, which are registered as attribute-value pairs. Building on this basis of publisher-subscriber messaging, Pawn provides services for message aggregation, filtering and optimization. Furthermore, it provides higher level peer-to-peer Grid interaction services such as lightweight and secure remote method calls which combine transport and delivery guaranteed messaging semantics. Note that in Pawn, peers interactions and collaborations occur spontaneously and unlike conventional messaging middleware, do not require a centralized server to route or parse information traveling from one peer to another. Instead, its routing infrastructure relies on a distributed set of lightweight rendezvous/router peers that may fail without incurring a global system failure.

Pawn defines messages that contain the required information to maintain state of the transactions/communications. If a rendezvous peer fails, the message can be rerouted without incurring other penalties than the route reevaluation processing delay.

Pawn defines a set of services for large-scale scientific collaborations, each service advertises itself to a dynamic group of users, messages flow between users and services through independent transports (TCP, UDP, HTTP...).

In Pawn, a peer can be of type application, client, or rendezvous/router; peers can concurrently combine one or more roles. For example, a client peer can act as an application provider as well as a rendezvous peer. We identified the fundamental services necessary to build a collaboratory as:

- Application Execution Service: to remotely start, stop or get the status of an application
- Application Monitoring and Steering Service: handles peers outgoing messages. It is responsible for adding semantic information to every message.
- Application Runtime Control: to announce the existence of an application to a peergroup, or return application responses
- Group communication Service: In order to exchange text messages between individual or a group of peers.

1.3 Contributions

This thesis makes four key contributions:

- We argue that building large-scale scientific collaborations benefits from a purely peer-to-peer architecture as opposed to a client/server architecture.
- We define the design requirements of a peer-to-peer messaging middleware.
- We implement and deploy Pawn, a peer-to-peer messaging that allows building interoperable, adaptive and autonomic applications in a scalable and transparent manner for the user.
- We evaluate the performance of Pawn in real-world scenarios involving large-scale scientific collaborations. We show that the decentralized nature of Pawn offers a novel approach to distributed collaborative application for monitoring and steering, enabling loosely-coupled computational models, comparable to tightly-coupled computational models, at the cost of the network delays latencies.

1.4 Organization of the Thesis

Chapter 2 provides background on peer-to-peer networks, communication models and collaboratories and presents some related work that can help the reader position the problem among the exisiting research initiatives. Chapter 3 describes Project JXTA, the peer-to-peer framework on which Pawn builds. Chapter 4 describes the architecture of Pawn, its network components, network services, messages and interaction types. Chapter 5 presents an application of Pawn, for solving the problem of optimization of an oil reservoir. Chapter 6 provides experimental evaluations and analysis of the Pawn framework. Chapter 7 presents a summary of the work presented and outlines future work.

Chapter 2

Background and Related Work

2.1 P2P and the Grid

Peer-to-peer systems rely on standardized protocols to support interactions between peers. Such systems emerged before, and expanded during the growth of the Internet, as can be seen from two fundamental distributed applications, Usenet and the Domain Name System (DNS). Usenet was created in 1982; it is a newsgroup application that is based on the Network News Transfer Protocol, in which hosts exchange and synchronize lists of news messages that clients can retrieve by defining filters based on interest. DNS was created in 1986, and is a name-to-address lookup mechanism based on a hierarchy of hosts answering a client conversion lookup request. Figure 2.1 illustrates a Peer-to-Peer system; it shows every peer acting as a provider and a consumer of services offered to the group of peers.

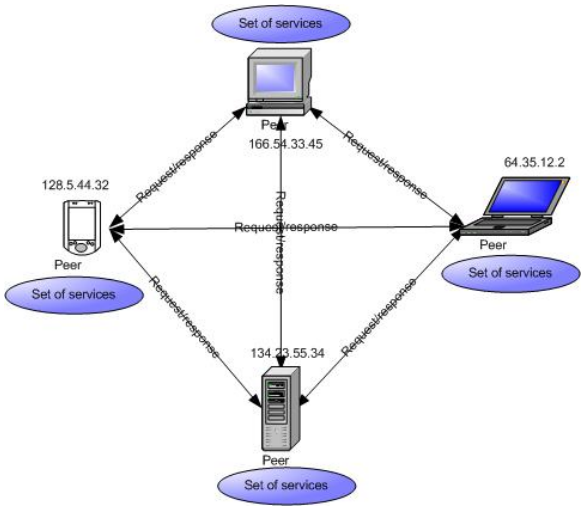

Figure 2.1: A Peer-to-Peer network

More recently, popular P2P applications [14][18], have provided specific functionalities such as file-sharing or distributed computing. However, most of these applications

are based on custom P2P protocols. The absence of interoperability between such applications has motivated the current efforts to standardize protocols for building P2P applications (e.g. JXTA[16] and BOINC[1]). Meanwhile, the Global Grid Forum [7] and Globus[4] are also attempting to standardize the infrastructure of the computing Grid, and its associated protocols and mechanisms. While, these two communities are motivated by different applications, there are significant similarities in the goals of these infrastructures, their architecture and their services.

In both systems, applications are supported by a messaging middleware layer that builds on the underlying protocols and mechanisms to enable the required interaction/communication semantics. Existing P2P and Grid messaging middleware efforts are discussed below.

2.2 Existing Messaging Solutions

A number of messaging solutions for P2P/Grid systems have been proposed in recent years. These systems define generic APIs and protocols for sending and receiving messages using publisher/subscriber or centralized models.

Message Oriented Middleware (MOM) systems such as JMS [12], LeSubscribe [2], and IBM MQSeries[8] provide publisher/subscriber support for content-based routing and event notification across wide area networks. Content-based routing typically defines servers (also called access points) that are responsible for filtering and aggregating incoming messages before distributing them to all the subscribed peers.

While these systems can be leveraged to enable interactions on the Grid, only few recent projects specifically address messaging for Grid applications. These include messaging systems such as GridRPC [19] and XEvents/XMessages[20], , or P2P messaging systems such as the NaradaBrokering [5] system. NaradaBrokering provides wide area event brokering targeting large scale collaborations in education and science. It is based on a publisher/subscriber model and defines a network of broker peers that can be dynamically deployed to provide efficient events distribution among groups of interested peers. The NaradaBrokering system was built on JMS and then integrated with JXTA.

GridRPC [19] provides a remote procedure call API for applications on the Grid. It extends the standard Remote Procedure Call[21] mechanism, and combines it with asynchronous coarse-grained parallel tasking to address the specific requirements of scientific computation in Grid environments. It is designed to hide the dynamicity, insecurity, and instability of the Grid from programmers. The XEvents/XMessages [20] framework is based on the publisher/subscriber model, and builds on SOAP to provide an event system on the Grid. The framework provides an API to publish and listen for events on the Grid. Communication is managed by middleware agents that dispatch messages in response to events. Outgoing messages are stored at the middleware before being sent to their final destination, allowing the framework to provide historical data and be resistent to failure. The ICENI Grid middleware[6] enables component-based application composition for e-Science. It defines a middleware on which services and resources are exposed and accessed through policy and security managers. ICENI follows the Open Grid Services Architecture (OGSA) [22] to define the interfaces of services. It uses Java and Jini for communication between components and for enabling the dynamic composition and deployment of Grid applications at runtime.

Pawn combines properties from messaging and P2P messaging on the Grid to provide publisher/subscriber functionalities (push, pull, request/response, transactions). Unlike other publisher/subscriber systems, Pawn focuses on interaction services to support application monitoring and steering, collaboration, and application execution on the Grid. It extends JXTA pipe and resolver services to provide guaranteed application-level message delivery. Messages contain state information, that allow the system to recover from failure.

Chapter 3

An Overview of Project JXTA

Project JXTA[16] (from juxtapose) is a general-purpose-peer-to-peer framework introduced by SUN Microsystems in April 2001. The main drive behind JXTA is to provide developers with an open, platform and language agnostic framework for deploying interoperable peer-to-peer applications and services. JXTA defines concepts, protocols, and a network architecture.

3.1 JXTA Concepts

JXTA concepts include peers, peergroups, advertisements, modules, pipes, rendezvous and security. A peer is any compute capable device on the network. A peergroup is a group of peers that share a common set of services. An advertisement is a language agnostic document (using XML) that describes a resource or functionality. A module is a piece of code (i.e. functionality) that can be initialized and started at runtime. Modules are published to a peergroup using advertisements that define their general behavior, specification, and implementation. An interested peer can then retrieve the module implementation and run it locally. A pipe is a virtual communication abstraction that provides input and output channels to a peer. Pipes are recognized by unique identifiers that are not based on the physical network address of the peer (e.g. IP address), allowing for a peer to change its network address while keeping its communication address valid. JXTA offers two types of pipes, Unicast pipes that provide a communication channel between two endpoints, and propagate pipes that can propagate a message to a group of peers. Pipes can block the current processing until a message is sent or received (Blocking pipe), or allow the current process to continue while it is waiting for a message to be sent or received (Non-Blocking pipe). Pipes can use any available protocol for message transmission. In the current Java implementation of JXTA, this defaults to TCP for Unicast pipes and UDP for propagate pipes. There are no application-level

guarantees associated with the transmission of messages over pipes. A rendezvous peer acts as a relay as well as a distribution peer for messages and advertisements. Any peer can be a rendezvous peer, and a rendezvous can join or leave the network at any time. A peer may also act as a relay (implementing Network Address Translation) to enable peers behind firewalls to be part of a wide-area peergroup. Security in JXTA is provided by the routing and transport layers at every endpoint, where each message can be encrypted in order to guarantee integrity, authenticity and confidentiality.

3.2 JXTA Protocols

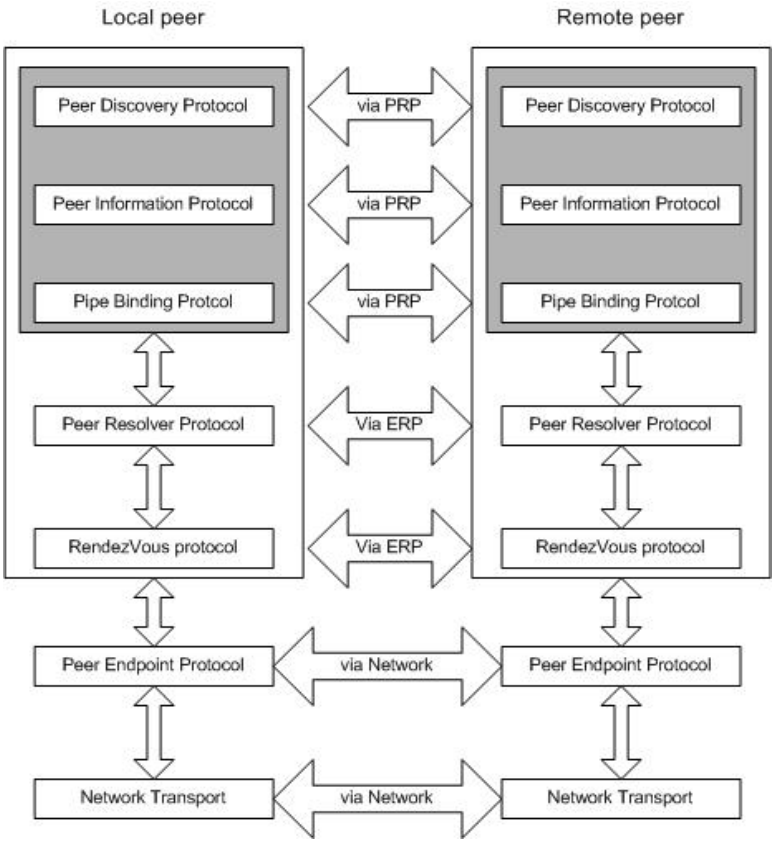

Figure 3.1: JXTA Protocols stack

JXTA defines protocols for : (1) discovering peers (Peer Discovery Protocol, PDP), (2) binding virtual end-to-end communication channels between peers (Pipe Binding Protocol, PBP), (3) resolving queries (Peer Resolver Protocol, PRP), (4) obtaining information on a particular peer, such as its available memory or CPU load (Peer

Information Protocol, PIP) (5) propagating messages in a peergroup (Rendezvous protocol, RVP), (6) determining and routing from a source to a destination using available transmission protocols (Endpoint Routing Protocol, ERP). Protocols in JXTA define the format of the messages exchanged as well as the behavior adopted on receiving a message. For example the Peer Resolver Protocol defines two messages, a Resolver Query Message and a Resolver Response Message. On receiving a resolver query message, a peer will attempt to resolve the message and produce a response, which is sent back to the originator of the query.

3.3 JXTA Architecture

The JXTA architecture builds on three layers: a core layer providing basic services and protocols for every peer to bootstrap a core platform and start communicating with other peers, a service layer enabling additional services to be started and collaboratively shared among members of a peergroup, and an application layer defining the graphical user interface and higher-level behaviors (such as Chat or file-sharing).

Chapter 4

Design and Implementation of Pawn

A conceptual overview of the Pawn P2P substrate is presented in Figure 4.1 and is composed of peers (computing, storage, or user peers), network and interaction services, and mechanisms. These components are layered to represent the requirements stack enabling interactions in a Grid environment. The figure can be read from bottom to top as :"Peers compose messages handled by services through specific interaction modalities".

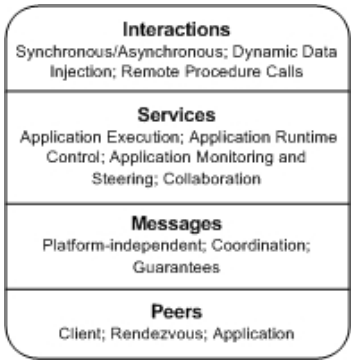

Figure 4.1: Conceptual design overview of Pawn

The overall architecture of Pawn is presented in Figure 4.2. It presents peergroups, composed of peers, that build on the Grid fabric, and implement Pawn services. The Pawn framework is also presented, it is composed of interaction types and services, that are described in this section.

4.1 Pawn Services

A network service is a functionality that can be implemented by a peer and made available to a peergroup. File-sharing or printing are typical examples of network services. In Pawn, network services are application-centric and provide the mechanisms to query, respond, subscribe, or publish information to a peergroup. Pawn offers four key services to enable dynamic collaborations and autonomic interactions in scientific

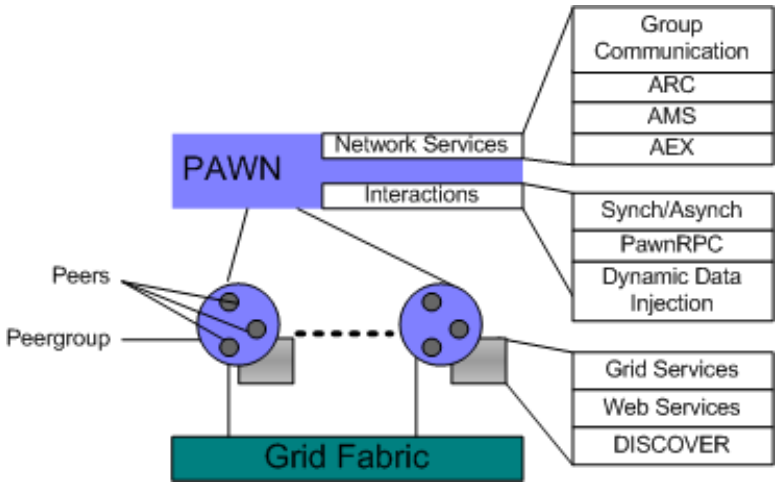

Figure 4.2: Pawn builds on network and interaction services to enable P2P interactions for application on the Grid

computing environments. These services are: *Application Runtime Control*, *Application Monitoring and Steering*,*Application Execution*, and *Group Communication*. These services are briefly described below along with the support they require from the Pawn P2P messaging substrate.

4.1.1 Application Execution Service [AEX]

The Application Execution service enables a peer to remotely start, stop, get the status of, or restart an application. In current systems, it is not uncommon that application users require an account on the domain hosting the application. A new user wishing to access the application will therefore need an account to be set up. This task is still performed manually by the system administrator and is not suitable to a dynamic environment in which users may join or leave spontaneously. The AEX service offers control of an application execution dynamically without the need to obtain an account on the domain where the application resides, by providing access to users through the Pawn framework that is running on every domain as an authenticated and valid user. This service requires a mechanism that supports synchronous and guaranteed remote calls necessary for resource allocation and application deployment (i.e. transaction oriented interactions) in a P2P environment.

4.1.2 Application Monitoring and Steering Service [AMS]

When interacting with an application, users generally monitor the evolution of the application, they may query the application for specific answers, or inject new data into the application to collect results. The application should be listening for user inputs and return responses to the requester. The Application Monitoring and Steering service handles interactive (Request/Response) application querying (i.e. PULL), and dynamic steering (i.e. PUSH) of application parameters. It requires support for synchronous and asynchronous communications with message delivery guarantees, and for dynamic data injection (e.g. to push information to an application at runtime).

4.1.3 Application Runtime Control Service [ARC]

Upon starting an application, users should be notified of the existence, location, and communication modalities of the application in order to start interacting with it. The application runtime control service announces the existence of an application to the peergroup, sends application responses, publishes application update messages, and notifies the peergroup of an application termination. This service requires mechanisms for pushing information to the peergroup and for responding to queries (i.e. PUSH and Request/Response interactions).

4.1.4 Collaboration Service [Group Communication, Presence]

The collaboration service provides collaborative tools and support for group communication and detection of presence. Users interested in an application can monitor and steer applications, and require mechanisms to control the collaborative interaction process. In addition to interacting with applications, users can interact with each other to exchange ideas and share points of view based on display of results obtained from the applications runs. Collaborating peers need to establish direct end-to-end communications through synchronous/asynchronous channels (e.g. for file transfer or text communication), and be able to publish information to the peergroup (Transaction and PULL interactions).

4.2 Types of Peers in Pawn

In Pawn, peers can implement one or more services (behaviors); the combination of services implemented by a peer defines its role. Typical roles for a peer are client, application or rendezvous and are presented in table 4.1. The services are shown horizontally and the types of peers vertically, "No" signifies that the peer does not implement the given service, "Yes" indicates that the peer does implement the given service, finally "Depends" indicates that the peer may or may not implement this service.

Services	AEX	ARC	AMS	Group Communication	Routing
Client	No	No	No	Yes	Depends
Rendezvous	No	No	No	No	Yes
Application	Yes	Yes	Depends	No	Depends

Table 4.1: Peer types and associated services.

Every peer maintains a cache where advertisements are stored; the cache is updated upon receiving new advertisements. If the discovered advertisement was not already present in the cache, it is then added; if it was present in the cache, it is discarded.

4.2.1 Client Peer

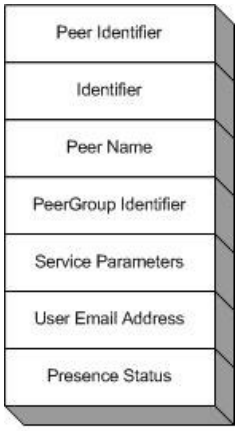

Figure 4.3: MetaData of a peer Advertisement

The client peer can deploy applications on available resources for monitoring and/or steering; the client can also collaborate with other peers in the group using Chat and

Whiteboard tools. Upon joining the group, a client peer advertises its presence through a peer presence advertisement, other peers can then discover the newly joined peer and start communicating with it. This peer advertisement contains the peer identifier that uniquely identifies it across all groups, the peergroups identifiers of which the peer is a member, a list of service parameters that are peer services implemented by this peer, an email address used to validate the authenticity of the peer for Chat communications, and a presence status stating the network status of this peer to the group (i.e. online, offline, away). When leaving the group, a peer publishes a presence advertisement informing other peers that its status has changed to offline. The joining procedure is typically a connection to a rendezvous peer that is part of the group. JXTA provides a mechanism to configure a peer with a list of predefined rendezvous endpoint addresses. We used this feature to direct every peer to a web link listing all rendezvous peers which are part of the Pawn framework. Upon startup, peers download the list of all rendezvous peers and can proceed to connect to the network through a rendezvous peer. A client peer may act as a rendezvous peer either from an initial configuration or switch to this behavior at runtime.

4.2.2 Application Peer

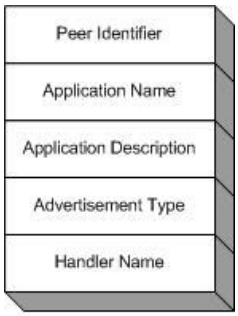

Figure 4.4: MetaData of an Application Advertisement

An application peer exports its application interfaces and controls to the peergroup; these interfaces are used by other peers to interact with the application. An application may already be enabled to communicate remotely with a middleware server as in the DISCOVER computational collaboratory [10]; in such a case, the application peer acts

as a proxy peer, relaying queries and responses to and from clients to applications. Application peers publish an application advertisement that informs the group about the application name, the peer identifier on which it is currently running and gives a brief description of the application, specifies an application type (e.g. simulation, remote startup utility, web service, open Grid service, etc.), and contains a handler name that every subsequent communication with this specific application will need to use as a handler tag. When newly joined peers are discovered, an application advertisement is sent to the group to inform all peers about the existence of the application.

4.2.3 Rendezvous Peer

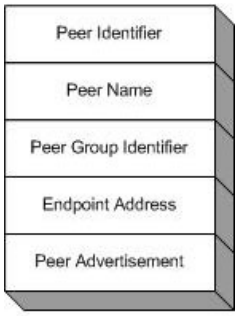

Figure 4.5: MetaData of a Rendezvous Advertisement

The role of the rendezvous peer is to distribute or relay messages. Rendezvous peers filter messages as defined by filtering rules input from the connected clients. Rendezvous peers route messages from a source to a destination, being reliable TCP unicast messages traveling between specific endpoint addresses to an unreliable group multicast. The rendezvous peer is a lightweight component that can fail without incurring global system failure. This is possible as long as there are enough rendezvous peers deployed in a system, allowing peers to reconnect to another rendezvous peer whenever a rendezvous disconnects or fails. Upon joining the network, rendezvous peers advertise their existence by publishing a rendezvous advertisement that other peers can use to initiate a connection to the rendezvous. A rendezvous advertisement contains the unique peer identifier, the name of the peer and the identifier of the group. We modified the original JXTA implementation of the rendezvous advertisement to include the endpoint

address as well as the peer advertisement of the physical node on which the rendezvous is running in order for the peers discovering the advertisement to initiate a rendezvous connection at runtime.

4.3 Implementation of Pawn Services

JXTA defines unicast pipes that provide a communication channel between two endpoints, and propagate pipes that can multicast a message to a peergroup. It also defines the Resolver Service that sends and receives messages in an asynchronous manner. The recipient of the message can be a specific peer or a peergroup. The pipe and resolver services transfer messages using available underlying transport protocols (TCP, HTTP, TLS). To realize the four services identified above, Pawn extends the pipe and resolver services to provide stateful and guaranteed messaging. This messaging is then used to enable the key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls.

4.3.1 Stateful Messages

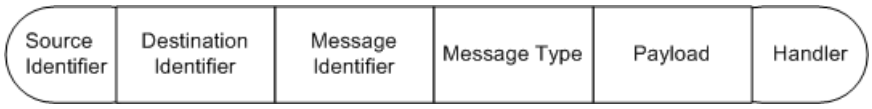

Figure 4.6: Format of a Pawn message

In Pawn, messages are platform-independent, and are composed of source and destination identifiers, a message type, a message identifier, a payload, and a handler tag, that are shown on Figure 4.6. The handler tag uniquely identifies the service that will process the message. State is maintained by making every message a self-sufficient and self-describing entity that carries enough information such that, in case of a link failure, it can be resent to its destination by an intermediary peer without the need to be recomposed by its original sender. Intermediary peers can act as proxies, and handle storing and forwarding of messages to and from a peer. Figure 4.7 shows the sequence of events following a link failure, that is detected if an acknowledgement to a message is not received from the destination peer. Acknowledgment messages are received from

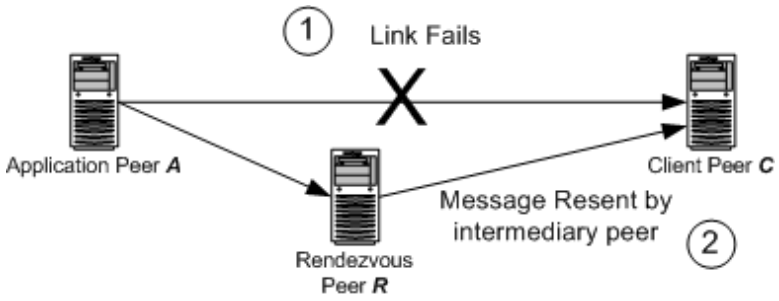

Figure 4.7: Using stateful messages, after detection of a transmission failure, a message can be resent from an intermediary peer without the need to be recomposed by its original sender

the intermediary proxy peer as well as the sender peer, if an acknowledgement is not received, the proxy peer will forward messages on behalf of the sender peer. This capability is combined with the monitoring of the status of the peer connected to the proxy peer. If the peer happens to be down, the proxy will act on behalf of the peer for a determined time set by a timeout value passed on during connection. Figure 4.7 shows such an event in which in response to a peer failure, an intermediary rendezvous node takes care of re-sending a message that was meant to be sent before the sender peer failed. The failure detection is handled by the rendezvous peer monitoring the status of the peers connected to it. Once a failure is detected, the rendezvous will send every message from its queue that are labeled with the failed peer identifier. It will store all responses to the failing peer for the determined timeout value and forward it to them if the peer comes back online.

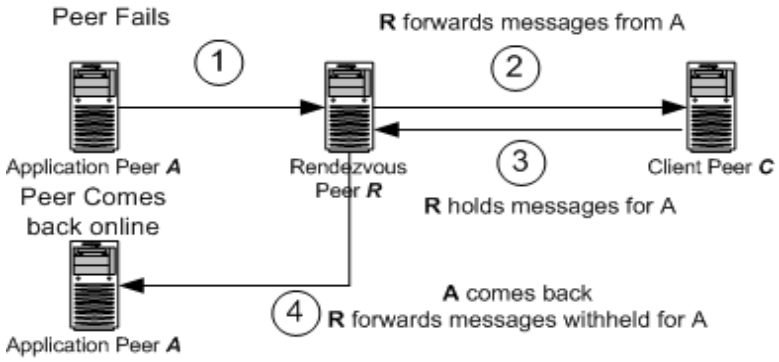

Figure 4.8: After detection of a peer failure, messages can be retrieved from a proxy peer once the peer comes back up, message ordering and payload information are contained within the stateful message and do not require message recomposition from the sender

In addition, messages can include system and application parameters in the payload to maintain application state.

4.3.2 Message Guarantees

Pawn implements application-level communication guarantees by combining stateful messages and a per-message acknowledgment table maintained at every peer. FIFO message queues are used to handle all incoming and outgoing messages. Every outgoing message that expects a response is flagged in the table as awaiting acknowledgment. This flag is removed once the message is acknowledged. Messages contain a default timeout value representing an upper limit on the estimated response time. If an acknowledgment is not received and the timeout value expires, the message is resent by an intermediary node. The message identifier is a composition of the destination and sender's unique peer identifiers. It is incremented for every transaction during a session (interval between a peer joining and leaving a peergroup) to provide application-level message ordering guarantees.

4.3.3 Synchronous/Asynchronous Communication

Communication in JXTA can be synchronous (using blocking pipes) or asynchronous (using non-blocking pipes or the resolver service). In order to provide reliable messaging, Pawn combines these communication modalities with stateful messaging and guarantee mechanism. Figure 4.9 presents a request/response interaction between the AMS and ARC services, and the message queuing for outgoing and incoming messages during an end-to-end communication. The figure shows that a message (query) is formed and added to the outgoing queue by the peer implementing AMS. This message is then sent out to the peer implementing ARC. The receiving peer queues the message before processing it to maintain ordering of application-level messages. Once the message is processed, a similar mechanism in the reverse direction is used to send back a response to the requesting peer. When using synchronous communication, the sender blocks its processing after adding the message to the queue, until reception of the corresponding response.

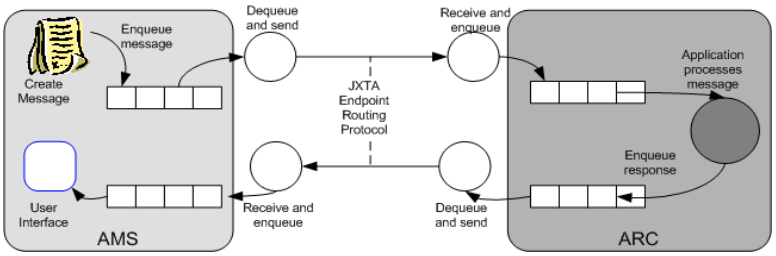

Figure 4.9: An interaction between a peer implementing the AMS service and a peer implementing the ARC service. Messages are added to the queue to be sent in order, and are received and enqueued to be processed in order.

4.3.4 Dynamic Data Injection

Pawn leverages JXTA pipes mechanisms with and combines it with its guaranteed message delievery mechanism to provide Dynamic Data Injection. The pipe advertisement is obtained from the peer advertisement published by every peer in Pawn; the pipe allows to uniquely identify an input and output communication channel to the peer. This pipe is used by other peers to create an end-to-end channel to dynamically send and receive messages.

Every interacting peer implements a message handler that listens for incoming messages on the peer's input pipe channel. The message payload is passed to the application/service identified by the handler tag field, allowing for data to be dynamically injected into the application at runtime. This process enables applications to build autonomic interactions that require very limited human intervention; for example, services can dynamically inject data into a running application process until a certain threshold value is reached.

4.3.5 Remote Procedure Calls (PawnRPC)

The PawnRPC mechanism provides the low-level constructs for building applications interactions across distributed peers. Using PawnRPC, a peer can dynamically invoke a method on a remote peer by passing its request as an XML message through a pipe. The interfaces for the methods are exported by a peer and are published as part of the peer advertisement during peer discovery.

Figure 6.6 presents the sequence of operations for a PawnRPC call. The figure

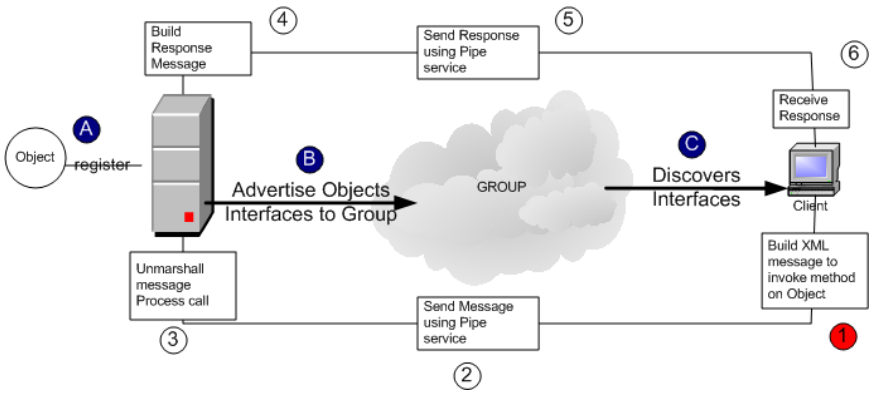

Figure 4.10: Sequence of Operations for a PawnRPC call on a dynamically registered object

shows two distinct sequences, a lettered, A, B and C sequence, and a numbered 1 to 6 sequence. The A-B-C sequence represents the operations prior to the remote invocation call, these involve registering the object that will receive method calls (A), advertising the interfaces of this object to the group (B), and a remote peer discovering this advertisement (C). The 1 to 6 sequence presents the successive operations required to invoke a remote procedure call and obtain a response. (1) The PawnRPC XML message is a composition of the destination address, the remote method name, the arguments of the method, and the arguments associated types. (2)(3) Upon receiving a PawnRPC message, a peer locally checks the credentials of the sender, and if the sender is authorized, (4) the peer invokes the appropriate method and (5)(6) returns a response to the requesting peer. The process may be done in a synchronous or asynchronous manner. PawnRPC uses the messaging guarantees to assure delivery ordering, and stateful messages to tolerate failure.

Chapter 5

A Prototype Application: The Oil Reservoir Optimization Process

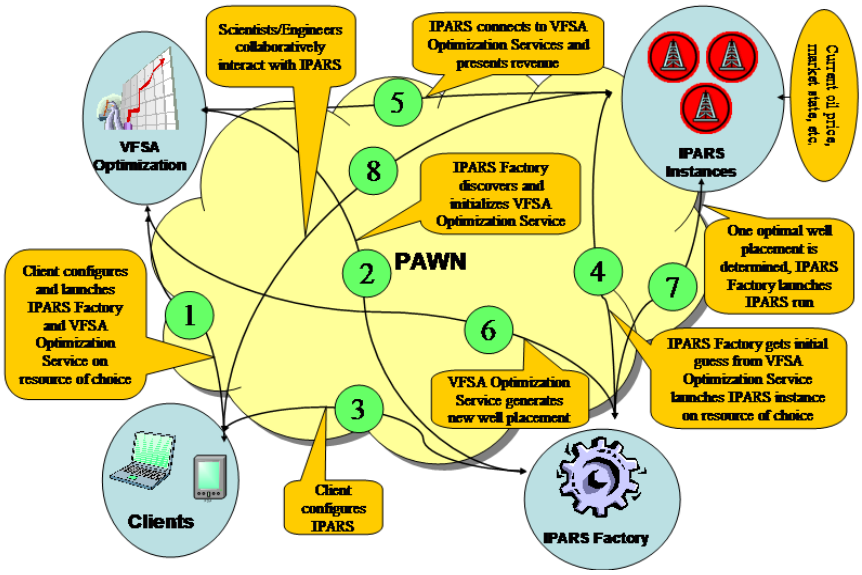

Figure 5.1: Autonomic optimization in IPARS using VFSA

The goal of the prototype application presented in this section is to dynamically optimize the placement and configuration of oil wells to maximize revenue. The overall application scenario is illustrated in Figure 5.1. The primary peer components involved are described below.

5.1 Application Components

5.1.1 Integrated Parallel Accurate Reservoir Simulator (IPARS)

IPARS[23] is a parallel reservoir simulation framework for modeling multiphase, multiphysics flow in porous media. It offers sophisticated simulation components that encapsulate complex mathematical models of the physical interaction in the subsurface, and execute on distributed computing systems. The simulator supports three dimensional transient flow of multiple phases containing multiple components plus immobile

phases (rock and adsorbed components). Phase densities and viscosities may be arbitrary functions of pressure and composition or may be represented by simpler functions. The initial system is isothermal, but nonisothermal calculations may be added later. There are currently ten *physical* models in IPARS, including multiphase gas–oil–water and air–water flow and reactive transport models. IPARS is primarily implemented in Fortran and C and is intrgrated using C++ wrappers and the Java Native Interface.

The **IPARS Factory** is responsible for configuring instances of IPARS simulations, deploying them on resources on the Grid, and managing their execution. Configuration consists of selecting appropriate models from those provided by IPARS, defining the structure and properties of the reservoir to be simulated, specifying required parameters and producing relevant input files. Deployment and management of IPARS instances uses services provided by DISCOVER[10] and Globus[4].

5.1.2 Very Fast Simulated Annealing (VFSA)

The VFSA[17] optimization service is based on statistical physics and the analogy between the model parameters of an optimization problem and particles in an idealized physical system. Simulated annealing (SA) is analogous to the natural process of crystal annealing when a liquid gradually cools to a solid state. The SA technique starts with an initial model, with associated cost or energy. It then draws a new model from a flat distribution of models within the predefined limits. The associated energy for the new model is then computed, and compared against the initial model. If the energy of the new state is less than the initial state, the new state is considered to be good and is accepted to replace the initial model unconditionally. However, if the energy of the new state is larger than the initial state, the new model is accepted with a defined probability. This process is repeated for a large number of times, with the annealing temperature gradually decreasing according to the predefined scheme. With a carefully defined cooling schedule, a global minimum can be found.

Very Fast Simulated Annealing (VFSA) is variant of SA that speeds up the annealing process. VFSA differs from SA in the following ways. The new model is drawn from a temperature dependent Cauchy like distribution centered around the current

model. This change has two fundamental effects. First it allows for larger sampling of the model space at the early stages of the inversion when the temperature is high, and much narrower sampling in the model space as the inversion converges when the temperature decreases. Second, each model parameter can have its own cooling schedule and model space sampling scheme. Therefore it allows for the individual control for each parameter, and the incorporation of a priori information. Applications of VFSA to several geophysical inverse problems can be found in[17].

5.1.3 Economic Modeling Service

The Economic Modeling Services uses the output produced by an IPARS simulation instance and current market parameters (e.g. oil prices, drilling costs, etc.) to compute estimated revenues for a particular reservoir configuration. In general, economic value of production is a function of the time of production and of injection rates in the reservoir. It takes into account fixed costs such as drilling a well, injection, extraction, disposal, and removal of the hydrocarbons, as well as additional fluids which are used in the injection process, or are being extracted from the field.

5.1.4 DISCOVER Computational Collaboratory

DISCOVER[10] is a virtual, interactive computational collaboratory that provides services to enables geographically distributed scientists and engineers to collaboratively monitor, and control high performance parallel/distributed applications on the Grid. Its primary goal is to bring Grid applications to the scientists'/engineers' desktop, enabling them to collaboratively access, interrogate, interact with, and steer these applications using pervasive portals. Key components of the DISCOVER collaboratory include:

- **DISCOVER Interaction & Collaboration Substrate**[9] that enables global collaborative access to multiple, geographically distributed instances of the DISCOVER computational collaboratory, and provides interoperability between DISCOVER and external Grid services. The middleware substrate enables DISCOVER interaction and collaboration servers to dynamically discover and connect

to one another to form a peer network. This allows clients connected to their local servers to have global access to all applications and services across all servers based on their credentials, capabilities and privileges.

The DISCOVER middleware also integrates DISCOVER collaboratory services with the Grid services provided by the Globus Toolkit [4] using the CORBA Commodity Grid (CORBA CoG) Kit [15]. Clients can now use the services provided by the CORBA CoG Kit to discover available resources on the Grid, to allocate required resources and to run applications on these resources, and use DISCOVER to connect to and collaboratively monitor, interact with, and steer the applications.

- **DIOS Interactive Object Framework (DIOS)**[13] that enables the runtime monitoring, interaction and computational steering of parallel and distributed applications on the Grid. DIOS enables application objects to be enhanced with sensors and actuators so that they can be interrogated and controlled. Application objects may be distributed (spanning many processors) and dynamic (be created, deleted, changed or migrated at runtime). A control network connects and manages the distributed sensors and actuators, and enables their external discovery, interrogation, monitoring and manipulation. The DIOS distributed rule engine allows users to remotely define and deploy rules and policies at runtime and enables autonomic monitoring and steering of Grid applications.

- **DISCOVER Collaborative Portals**[10] that provide the experts (scientists, engineers) with collaborative access to other peers components. Using these portals, experts can discover and allocate resources, configure and launch peers, and monitor, interact with, and steer peer execution. The portal provides a replicated shared workspace architecture and integrates collaboration tools such as chat and whiteboard. It also integrates "Collaboration Streams," that maintain a navigable record of all client-client and client-applications interactions and collaboration.

Using the DISCOVER computational collaboratory clients can connect to a local server using the portal, and can use it to discover and access active applications and

services on the Grid as long as they have appropriate privileges and capabilities. Furthermore, they can form or join collaboration groups and can securely, consistently, and collaboratively interact with and steer applications based on their privileges and capabilities.

The components described above need to dynamically discover and interact with one another as peers to achieve the overall application objectives. As can be seen on figure 5.1, the experts use the portals to interact with the DISCOVER middleware and the Globus Grid services to discover and allocate appropriate resource, and to deploy the IPARS Factory, VFSA and Economic model peers (step 1). The IPARS Factory discovers and interacts with the VFSA service peer to configure and initialize it (step 2). The expert interacts with the IPARS Factory and VFSA to define application configuration parameters (step 3). The IPARS Factory then interacts with the DISCOVER middleware to discover and allocate resources and to configure and execute IPARS simulations (step 4). The IPARS simulations now interacts with the Economic model to determine current revenues, and discovers and interacts with the VFSA service when it needs optimization (step 5). VFSA provides the IPARS Factory with optimized well information (step 6), which then uses it to configure and launche new IPARS simulations (step 7). Experts can, at anytime, discover, collaboratively monitor and interactively steer IPARS simulations, configure the other services, and drive the scientific discovery process (step 8). Once the optimal well parameters are determined, the IPARS Factory configures and deploys a production IPARS run.

5.2 Optimizing the Oil Reservoir using the Pawn Framework

In this section, we describe how Pawn is used to support the prototype autonomic oil reservoir optimization application outlined in section 2. Every interacting component is a peer that implements Pawn services.

The IPARS Factory, VFSA, and the DISCOVER collaboratory are Application peers and implement ARC and AEX services. The DISCOVER portals are Client peers and implement AMS and Group communication services. Key operations in the

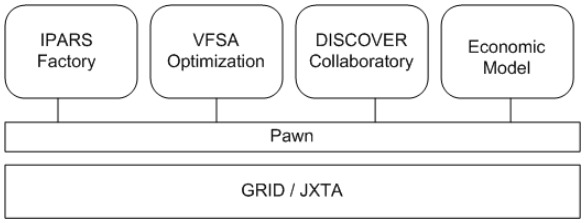

Figure 5.2: Architecture of the Oil Reservoir Optimization Application

process include peer deployment (e.g. IPARS Factory deploys IPARS), peer discovery (e.g IPARS Factory discovers VFSA), peer initialization and configuration (e.g. Expert configures VFSA), autonomic optimization (e.g IPARS and VFSA interactively optimize revenue), interactive monitoring and steering (e.g. Experts connect to, monitor, and steer IPARS), and collaboration (e.g. Experts collaborate with one another). These operations are described below.

5.2.1 IPARS Factory and VFSA Optimization Service Deployment

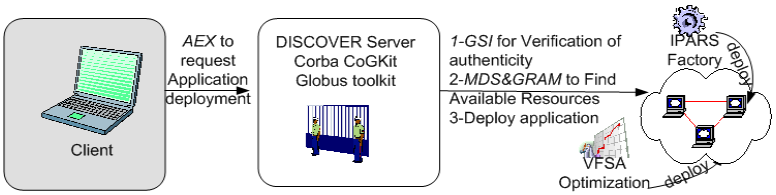

Figure 5.3: Peer Deployment

The IPARS Factory and VFSA Optimization peers are deployed using Globus services accessed through DISCOVER/CORBACoG [15]. The VFSA peer is a Fortran program with C wrappers and is integrated with Pawn using the Java Native Interface. Figure 5.3 presents the sequence of operations involved. The deployment is orchestrated by the Expert through the DISCOVER portal. The portal gives the Expert secure access to all the machines registered with Globus MDS to which the Expert has access privileges. Authentication and authorization is based on the Globus GSI service. Once authenticated, the Expert can use the portal to deploy the IPARS Factory and VFSA peers on machines of choice after verifying their availability and current status (load, cpu, memory). Deployment uses the Globus GRAM service. The portal also gives the Expert access to already deployed services and applications for collaborative

monitoring and steering using DISCOVER services.

5.2.2 Peer Initialization and Discovery

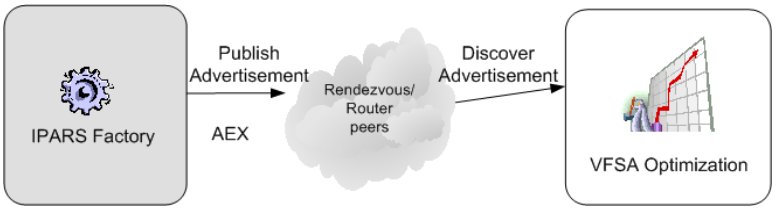

Figure 5.4: Peer Discovery

Figure 5.4 shows the discovery process between the IPARS Factory and the VFSA optimization peers. On startup, peers use the underlying JXTA discovery service to publish an advertisement to the peergroup. This advertisement describes the functionalities and services offered by the peer. It also contains a pipe advertisement for input and output communications, and the RPC interfaces offered by the peer for remote monitoring, steering, service invocation and management. To enable peers to mutually recognize each other, the peer that discovers an advertisement sends its advertisement back to the discovered peer. This discovery process is also used by IPARS instances to discover the VFSA service.

5.2.3 IPARS and VFSA Configuration

The Expert uses the portal and the control interfaces exported to configure the VFSA service and to define its operating parameters. The Expert also configures the IPARS Factory by specifying the parameters for IPARS simulations. The IPARS Factory uses these parameters to setup IPARS instances during the optimization process, and initialize the VFSA service. Note that the Expert can always use the interaction and control interfaces to modify these configurations. The configuration uses the AMS to send application parameters to the IPARS Factory and VFSA peer. A response is generated and sent back (using AEX) to the client to confirm the configuration change.

5.2.4 Oil Reservoir Optimization

The reservoir optimization process consists of 2 phases, an initialization phase and an iterative optimization phase as described below. **Initialization phase:**

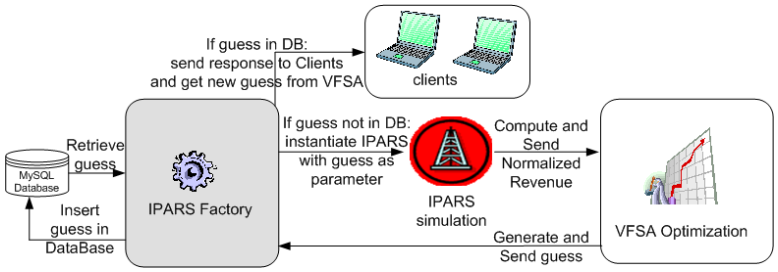

Figure 5.5: Optimization Run

In the initialization phase, VFSA provides the IPARS Factory with an initial guess of well parameters based on its configuration by the Expert and the IPARS Factory. This is done using the channel established during discovery and is used by the IPARS Factory to initialize and deploy an IPARS instance.

Iterative optimization phase: In the iterative optimization phase, the IPARS instance uses the Economic Model along with current market parameters to periodically estimate the current revenue. This revenue is normalized and then communicated to the VFSA. The VFSA uses this value to generate an updated guess of the well parameters and sends this to the IPARS Factory. The IPARS Factory now configures a new instance of IPARS with the updated well parameters and deploys it. This process continues until the required terminating condition is reached (e.g. revenue stabilizes). Figure 5.5 shows the overall optimization process between IPARS Factory, IPARS, and VFSA. Note that Experts can connect to any of these peers at any time and steer the optimization process.

Well Parameter and Normalized Revenue Archive After each iteration of optimization process, normalized well parameters produced by VFSA, and the revenue and normalized revenue produced by the Economic Model are stored in an archive (MySQL database) maintained by an archival peer. During the optimization process, when a new set of well parameters are received from VFSA, the IPARS Factory checks the archive before launching an IPARS instance. If the current guess is already present in the

archive, the corresponding normalized revenue value is sent to VFSA and a redundant IPARS instance is avoided.

Note that peer interactions during the optimization process are highly dynamic and require synchronous or asynchronous RPC semantics with guarantees, rather than the document exchanges typically supported by P2P systems. In Pawn, these interactions are enabled by PawnRPC that provides the same semantics as the traditional RPC in a client-server system, but is implemented in a purely P2P manner.

5.2.5 Production Runs and Collaborative Monitoring and Steering

Once the optimization process terminates and the optimal well parameters are determined, the IPARS Factory allocates appropriate resources, configures a production run based on these parameter, and launches this run on the allocated resources. Experts

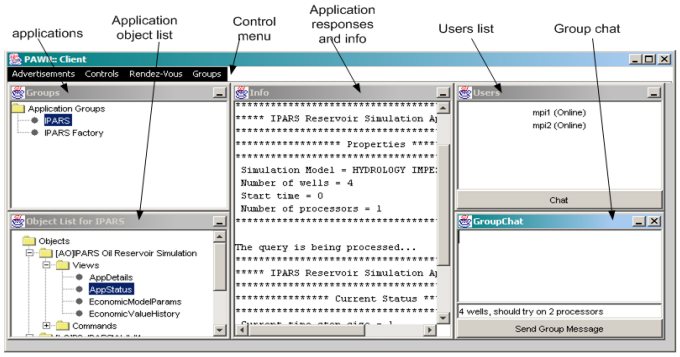

Figure 5.6: Graphical User Interface of the Expert's Portal

can now collaboratively connect to the running application, collectively monitor its execution and interactive steer it. Figure 5.6 presents the client peer's portal interface used by the Experts. The portal interface can also be used to access, monitor and steer the IPARS Factory, the VFSA optimization service, and the Economic Model. The Figure shows the Collaboration Services provided including Chat and Whiteboard.

5.2.6 Sample Results from the Oil Reservoir Optimization Process

Sample results from the oil reservoir optimization process are plotted in Figure5.7. The plots show the well position guess produced by the VFSA optimization service and the

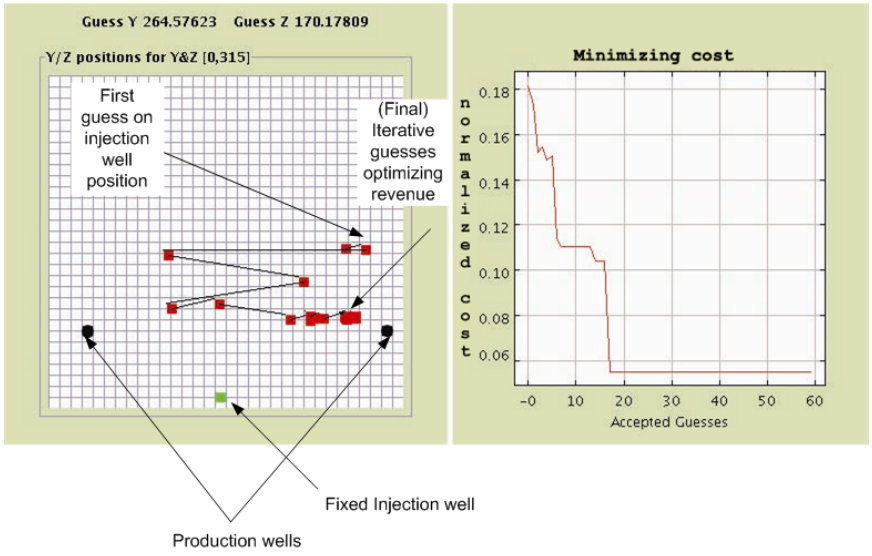

Figure 5.7: Well Positions and Normalized Costs

corresponding normalized cost. The well positions plot (on the left in Figure 5.7) shows the oil reservoir field and the positions of the wells. The black circles are the fixed injection wells. The well at the bottom most part of the plot is a fixed production well. The plot also shows the sequence of guess returned by the VFSA service for the other production well (shown by the lines connecting the light squares). For each guess, the plot on the right shows the corresponding normalized cost value. It can be seen that this value decreases with successive guesses until it stabilizes. This validates the optimization process. These results show that the optimization process required 20 iterations for this example.

Chapter 6

Experimental Evaluation of Pawn

6.1 Experimental Setup

In order to evaluate the performance of the framework, we deployed Pawn on 20 peers part of a Linux cluster located at Rutgers University.

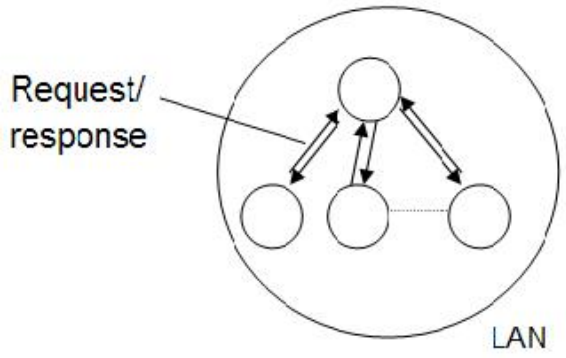

Figure 6.1: On the LAN, a host implementing the AEX service queries peers implementing the AMS service

The Wide Area measurements were performed using two remote peers located at an ISP domain.

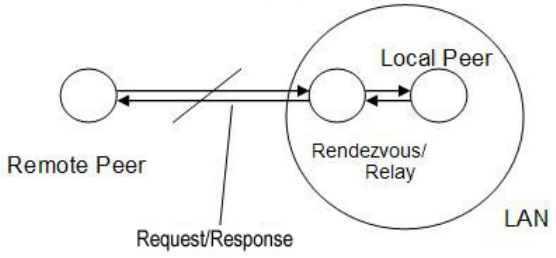

Figure 6.2: On the WAN, a remote host implementing the AEX service queries peers implementing AMS in a LAN. The remote host has to act as an HTTP relay peer to be able to send information through the firewall running on the LAN

The configuration of the test machines is presented in table6.1

Number of peers	Operating System	Processor	Memory(MB)	Usage
20	Linux RedHat 7.2	Pentium IV 1.5 GHz	512	LAN
1	Windows XP Professional	Pentium III 750 MHz	256	LAN&WAN
1	Linux-kernel 2.5.3	Pentium II 350 MHz	256	WAN

Table 6.1: Test hosts configurations

6.2 Round Trip Time Communication on the LAN

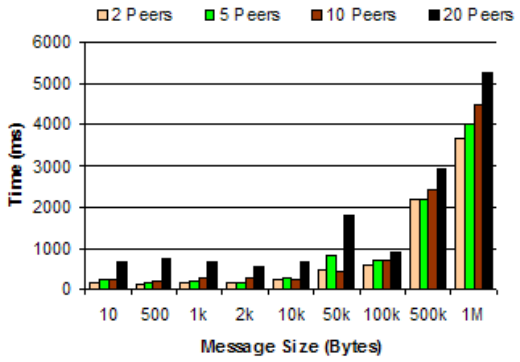

Figure 6.3: Round Trip Time (RTT) measurement in a LAN, shows that despite the increased latency as message size increases, the difference for 2 and 20 peers remains constant. Pawn does not add cost to the growing size of the group but will improve as the underlying JXTA platform improves

This experiment evaluates the round trip time of messages sent from a peer running the *Application Monitoring and Steering service*, to peers running the *Application Runtime Control service* over a LAN. The message size varies from 10 bytes to 1 Megabyte. The Pawn services (AMS and ARC) build the query and response messages that are then sent using the JXTA Endpoint Routing Protocol. The overall message transfer time is tightly bound to the performance of the JXTA platform, and is likely to improve with the next generation JXTA platform. The results are plotted in Fig.6.3. Note that the difference in RTT between 2 peers and 20 peers decreases as the message size increases.

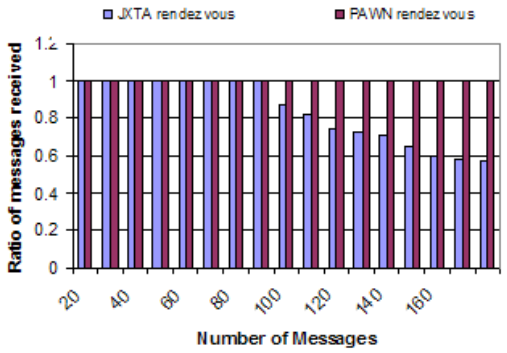

Figure 6.4: Pawn builds on JXTA pipes and offers a message queueing mechanism to avoid dropping messages.

6.3 Effectiveness of Message Queuing

This experiment compares the behavior of a Pawn rendezvous peer implementing the application-level message queuing to the behavior of a core JXTA rendezvous peer. The number of messages published on the rendezvous peer range from 10 to 500. The ratio of messages received is plotted in Figure 6.4. It can be seen that the message queuing functionality guarantees that no application-level messages are dropped even under heavy load.

6.4 Memory Cost of JXTA and Pawn

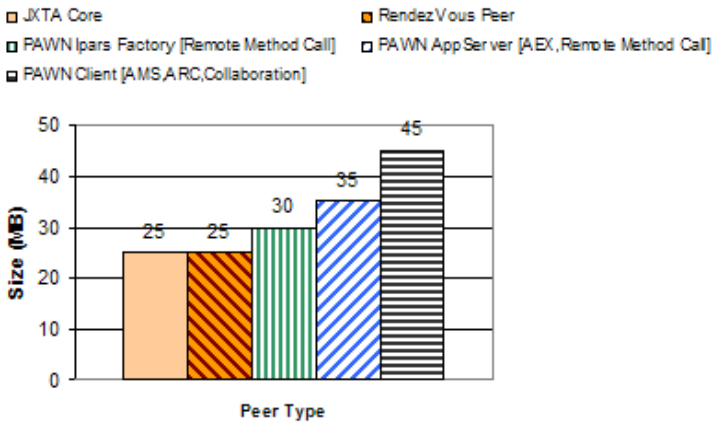

Figure 6.5: Pawn and JXTA memory cost

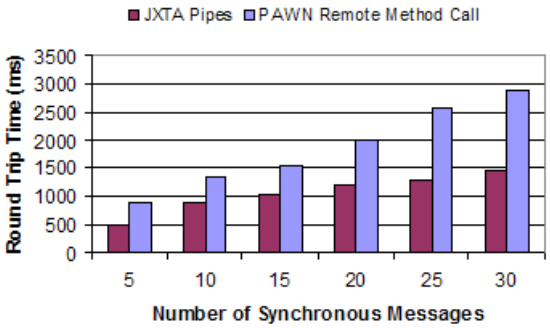

Figure 6.6: PawnRPC adds a cost over the basic JXTA pipes. That cost is associated to marshaling and unmarshaling of the XML remote call invocation process

In order to bootstrap a peer in Pawn, every peer has to load a core application implementing the JXTA and Pawn protocols. Figure 6.5 plots the memory used for loading the Java Virtual Machine, and the core JXTA protocols for every type of peer in Pawn. Pawn services add, on an average, an overhead of 20% to the JXTA core. Note that the client peer loads a Graphical User Interface in addition to the AMS and Collaboration services and is an exception.

6.5 Overhead of PawnRPC on JXTA Pipes

Figure 6.6 shows a comparison between PawnRPC and JXTA pipes. Using PawnRPC, a message describing the remote call is marshaled and sent to the remote peer. The remote peer unmarshals the request, processes it before marshaling and sending back a response to the requesting peer. The marshaling, unmarshaling, and invocation add an overhead on the plain pipe transaction. This overhead remains however less than 50% on average.

Chapter 7

Conclusion

This thesis presented the design, implementation, and evaluation of Pawn, a peer-to-peer messaging substrate that builds on project JXTA to support peer-to-peer interactions for scientific applications on the Grid. Pawn provides a stateful and guaranteed messaging to enable key application-level interactions such as synchronous/asynchronous communication, dynamic data injection, and remote procedure calls. It exports these interaction modalities through services at every step of the scientific investigation process, from application deployment, to interactive monitoring and steering, and group collaboration. Pawn was successfully deployed to enable peer-to-peer interactions for an oil reservoir optimization application on the Grid. Pawn is motivated by our conviction that the next generation of scientific and engineering Grid applications will be based on continuous, seamless and secure interactions, where the application components, Grid services, resources (systems, CPUs, instruments, storage) and data (archives, sensors) interact as peers.

In future work, Pawn will be combined to a dynamic and decentralized rule engine, and extended to support the definition and deployment of components that can interact in an autonomic fashion, to provide self-healing, self-configuring, and self-managing systems. The rule engine will provide the mechanisms and semantics for initializing and distributing rules onto components. These components will support input and output controls for management, monitoring and steering, using Pawn to provide the underlying mechanisms to define, format, and transport messages in a stateful, guaranteed and autonomic manner. Pawn will also be extended to support advanced message filtering and aggregation mechanisms, to optimize message distribution in large collaborating groups by controlling the interest of peers to selectively route messages to.

References

[1] Berkeley Open Infrastructure for Network Computing. Internet: http://boinc.berkeley.edu, 2002.

[2] Françoise Fabret, Arno Jacobsen, François Llirbat, Joao Pereira, Kenneth A. Ross, and Dennis Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe Systems. *ACM SIGMOD Record*, 30(2):115–126, 2001, ACM Press.

[3] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, USA, 1998.

[4] Ian Foster and Carl Kesselman. The Globus Project: A Status Report. In *IPPS/SPDP'98 Heterogeneous Computing Workshop*, pages 4–18, Orlando, Florida, USA, 1998.

[5] Geoffrey Fox, Shrideep Pallickara, and Xi Rao. A Scaleable Event Infrastructure for Peer to Peer Grids. In *Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande*, pages 66–75, Seattle, Washington, USA, 2002. ACM Press.

[6] Nathalie Furmento, William Lee, Anthony Mayer, Steven Newhouse, and John Darlington. ICENI: An Open Grid Service Architecture Implemented with Jini. In *SuperComputing 2002 (SC2002)*, Baltimore, MD, USA, November 2002. 10 pages in CDROM.

[7] Global Grid Forum. Internet: http://www.gridforum.org.

[8] IBM MQSeries. Internet: http://www-3.ibm.com/ software/ts/mqseries/.

[9] Samian Kaur, Vijay Mann, Vincent Matossian, Rajeev Muralidhar, and Manish Parashar. Engineering a Distributed Computational Collaboratory. In *Proceedings of the 34th Hawaii Conference on System Sciences (HICSS)*, Wailea Maui, Hawaii, January 2001. IEEE Computer Society Press. 10 pages in CDROM.

[10] Vijay Mann and Manish Parashar. Engineering an Interoperable Computational Collaboratories on the Grid. M. Thomas G. Fox, D. Gannon, editor. *Concurrency and Computation: Practice and Experience, Special Issue on Grid Computing Environments*, 14(13-15):1569–1593, Hoboken, NJ, USA, 2002, John Wiley & Sons.

[11] Middleware Resource Center. Internet: http://www.middleware.org.

[12] Richard Monson-Haefel and Dave Chappell. *Java Message Service*. O'Reilly & Associates, Sebastopol, CA, USA, December 2000.

[13] Rajeev Muralidhar and Manish Parashar. A Distributed Object Infrastructure for Interaction and Steering. In J. Gurd R. Sakellariou, J. Keane and L. Freeman, editors, *Proceedings of the 7th International Euro-Par Conference (Euro-Par 2001),Lecture Notes in Computer Science*, volume 2150, pages 67–74, Manchester, UK, August 2001. Springer-Verlag.

[14] Napster. Internet: http://www.napster.com.

[15] Manish Parashar, Gregor Von Laszewski, Snigdha Verma, Jarek Gawor, Katarzina Keahey, and Hellen N. Rehn. A CORBA Commodity Grid Kit. Geoffrey Fox, Dennis Gannon, and Mary Thomas, editors. *Special Issue on Grid Computing Environments, Concurrency and Computation: Practice and Experience*, 14:1057–1074, Hoboken, NJ, USA, 2002, John Wiley & Sons.

[16] Project JXTA. Internet: http://www.jxta.org.

[17] Mrinal K. Sen and Paul L. Stoffa. *Global Optimization Methods in Geophysical Inversion*. Advances in Exploration Geophysics 4. Elsevier Science, New York, NY, USA, 1995.

[18] SETI@Home. Internet: http://setiathome.ssl.berkeley.edu.

[19] Keith Seymour, Hidemoto Nakada, Satoshi Matsuoka, Jack Dongarra, Craig Lee, and Henri Casanova. Overview of GridRPC: A Remote Procedure Call API for Grid Computing. In Manish Parashar, editor, *Proceedings of the Third International Workshop on Grid Computing (GRID 2002)*, pages 274–278, Baltimore,MD, USA, November 2002. Springer.

[20] Aleksander Slominski, Yogesh L. Simmhan, Albert Louis Rossi, Matt Farrellee, and Dennis Gannon. XEvents/XMessages: Application Events and Messaging Framework for Grid. Internet: http://www.extreme.indiana.edu/xgws/papers/xevents_xmessages_tr.pdf, September 2002. Indiana University.

[21] Raj Srinivasan. RPC: Remote Procedure Call Protocol Specification Version 2. Internet : http://www.freesoft.org/CIE/RFC/1831/, August 1995. Request For Comments: RFC1831. Category: Standards Track.

[22] Steven Tuecke, Karl Czajkowski, Ian Foster, Jeffrey Frey, Steve Graham, and Carl Kesselman. Grid Service Specification. Internet: http://www.globus.org/research/papers/gsspec.pdf, February 2002.

[23] John A. Wheeler and Malgorzata Peszyńska. IPARS: Integrated Parallel Reservoir Simulator. Internet: http://www.ticam.utexas.edu/CSM. Center for Subsurface Modeling, University of Texas at Austin.

[24] XML: Extensible markup language. Internet: http://www.w3.org/XML.