Managing Grid Messaging Middleware

Harshawardhan Gadgil, Geoffrey Fox, Shrideep Pallickara and Marlon Pierce

***Abstract*— Management in distributed systems has gained much importance in recent years. With the increasing complexity of applications, there is a need for effective management of components of the application. As application components span different administrative domains, differing security policies restrict access to these components. The problem gets more complicated in a dynamic environment where application components and the environment is in a constant state of flux, so that failure is the norm. In this paper we explore the issues related to management in dynamic and heterogeneous environments. We propose a scalable, fault-tolerant and Web Services - compliant management architecture that addresses these issues of management and also illustrate the functioning of our framework with respect to the NaradaBrokering messaging middleware.**

***Index Terms*— Messaging middleware, Web Services Management, Fault tolerance**

I. Introduction

MANAGEMENT in distributed systems has gained much importance in recent years. With the increasing complexity of applications, there is a need for effective management of components of the application. Management usually involves common operations such as the ability to control the resource (e.g. start, stop), ability to configure the resource for a specific task and monitor the status (e.g. heartbeat) of the resource. The Web Service community has recently introduced two competing specifications, namely, WS-Management [1] and WS-Distributed Management [15] for service-oriented management. The key idea inherent to both these specifications is modeling manageable resources as Web Service endpoints and managing these services by sending an appropriate message to this endpoint. In heterogeneous environment, the ability to manage a resource is restricted by presence of network address translation devices, firewalls and restricted transports. In this paper we address issues related to management and show how we can make management scalable and fault-tolerant.

A. Scalable and Fault-tolerant Management Framework

Figure 1 shows the various components of our framework. The entity being managed is any application specific component. We term such a resource as a *manageable resource*. Typically, existing Web Services would be augmented with specific ports for enabling remote management. A service adapter or a proxy is required when the entity being managed is not a Web Service. In such cases, the service adapter provides a Web Service interface for such entities. Thus this adapter is an entity specific proxy that has a Web Service interface on one end and an entity-specific interface on the other end. The adapter serves as translator of messages to commands specific to the entity being managed.

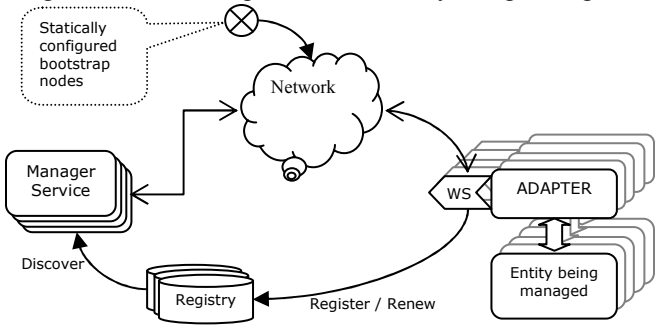

Figure 1 Basic Management Framework

In our architecture, we assume there could be multiple such services that require management. Examples of systems with large number of manageable resources are cell phone networks, large clusters of machines or even brokers in distributed brokering systems. The scheme should be scalable and incorporate management of a large number of manageable resources.

A Manager Service is the component of management architecture responsible for actually managing the manageable resources and acts upon the management tasks specified the user. The Manager serves mainly to invoke management actions on the managed entity as specified by the user. System state is maintained mainly in the registry while the Managers store short term state such as heartbeat information of components which is useful for quickly detecting component failures. Thus in a general case, the manager process talks with the service adapter using messages based on one of the Web Service based management specifications. Thus multiple managers would be present in the system to achieve scalability while fault-tolerance of the management process is guaranteed when at-least one of the manager processes is running. If one

This work is supported by the NASA Advanced Information Systems Technology (AIST) program, NSF Information Technology Research (ITR) program, project number 0427264 and the NSF Division of Earth Sciences project number EAR-0446610.

Harshawardhan Gadgil is with the Community Grids Lab, Indiana University, Bloomington, IN 47404, USA. He is also a graduate student in the Computer Science Department, Indiana University, Bloomington, IN 47405 USA (phone: 812-856-0756; e-mail: hgadgil@cs.indiana.edu).

Geoffrey Fox is the director of Community Grids Lab, Indiana University, Bloomington, IN 47404, USA. He is also with the Department of Computer Science, Department of Physics and School of Informatics, Indiana University, Bloomington, IN 47405 USA (e-mail: gcf@indiana.edu).

Shrideep Pallickara is with Community Grids Lab, Indiana University, Bloomington, IN 47404, USA. (e-mail: spallick@indiana.edu).

Marlon Pierce is with Community Grids Lab, Indiana University, Bloomington, IN 47404, USA. (e-mail: marpierc@indiana.edu).

or more manageable resources are unreachable (possibly because the manager and the manageable resources lie in different administrative domains), then the manager should try alternate means of transport for reaching the manageable resource. A simple load balancing technique may be used to determine which manager talks with which service adapter.

The system employs a shared space (registry) which is implemented using a fault-tolerant database. The registry is used to store important system state information and is used by the managers and users to share management related tasks. The service adapters are responsible for registering their presence in a global registry such as a *Universal Description, Discovery and Integration* (UDDI) registry and renewing their existence at regular intervals. The registry also helps store system state such as required management actions specified by the user, which are picked by one or more manager processes and acted upon.

Finally the system contains a set of statically configured bootstrap nodes which can be leveraged to start discovering the system components and tying them together. The bootstrap nodes function as a scalable messaging substrate to deliver messages between managers and service adapters. Fault-tolerance of these nodes is achieved by employing multiple bootstrap nodes. This is akin to the DNS (Domain Name Service) system where, if one DNS Server is unavailable, the client tries the next DNS service to achieve fault-tolerance. Thus failure of one of these nodes does not affect the entire management system and we expect that the failed node can be restored in finite amount of time.

B. Desiderata

We now summarize the desired characteristics of the management architecture, below:

Remote Management: The management system should enable us to manage resources irrespective of their location as long as there exists a way to access the resource.

Traverse firewalls and NAT: Application components may span administrative domains. The presence of firewalls and network address translation further complicates management by preventing specific transports and / or blocking access to internal machines. Frequently, by providing correct authentication, it is possible to tunnel messages, such as over the HTTP transport. The management architecture should work equally well in such heterogeneous environments by leveraging available transports.

Extensible: Management interfaces are generally resource specific. As the application infrastructure evolves, it should be possible to incorporate management of newer services with little or no modification. We address this issue by employing service-oriented management architecture.

Scalable: The management architecture should be administratively scalable such that, the complexity of management does not increase when a subset of the components are distributed over multiple administrative domains. Further, the management architecture should also be scalable in terms of the number of resources managed. We show how we can leverage a distributed messaging middleware to achieve scalability.

Fault-tolerant: The management architecture should itself be fault-tolerant. Failure of any services or transport should automatically trigger search for next possible transport. As an illustration, a resource may become unreachable due to various network conditions such as blocked ports, disabled transport such as UDP or multicast and failed services. In these cases the management adapter should try to avoid faulting by doing a best-effort-try to check for alternate means of services and transports. The architecture ensures fault-tolerance by employing multiple managers. Failure of a manger keeps the management process running when at-least one manager process is running. This is achieved by reassigning management tasks to a running manager when the original task manager has failed. This is possible because the managers are stateless services and the system state is maintained in the registry.

C. Scope of this paper

In this paper we present a WS-Management based architecture for managing services. We specifically target the management of a distributed messaging infrastructure since it employs a large number of distributed dynamic peers. We present results on the overhead introduced by our system and also present ways to improve performance. We also present an architecture which leverages a distributed messaging substrate to make management scalable.

The rest of the paper is organized as follows. We introduce NaradaBrokering and discuss the need for management in Section II. We describe the broker management architecture in Section III. Section IV presents the resources modeled using WS-Management. We present results obtained by testing our prototype implementation in Section V. Section VI is conclusion.

II. Managing the Distributed Brokering Infrastructure

Messaging based distributed brokering infrastructures have gained much popularity in recent years in the distributed computing community. They have been instrumental in helping to provide clear demarcation between the application logic and Quality of Service aspects such as reliable delivery, security, persistent storage, compression / decompression and fragmentation / de-fragmentation of messages. These brokering systems employ a large number of connected peers called brokers which form a messaging substrate. To get the maximum benefit from the services provided by the messaging substrate, it is required to setup these brokers and connect them in topologies specific to the application.

Various topologies [2] on connecting these peers exists, each based on differing routing, fault-tolerance and cost characteristics. Run-time metrics are gathered using monitoring techniques [3] which measure various aspects of the system that enable us to understand the performance of the system and in some cases, provide hints on improving the performance. This naturally leads to re-deployment of the brokering network with a different configuration. To

summarize, we need an architecture that enables us to rapidly bring-up and tear down a broker network. It is also required to set specific configuration settings for every broker and have the ability to change the configuration on-the-fly. We term these actions collectively, as management of the brokering infrastructure.

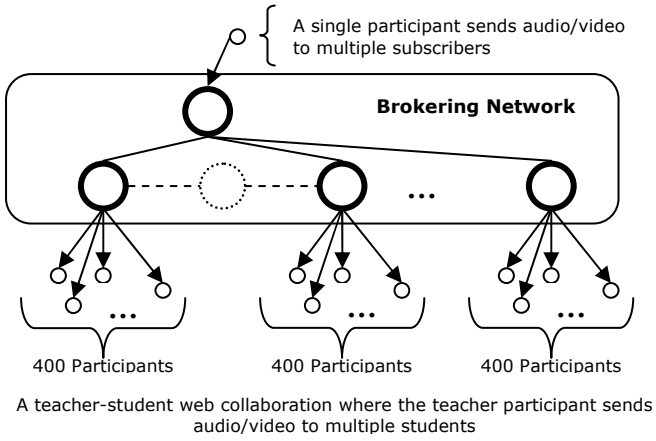

Figure 2: Teacher - student relationship based collaborative session

A. Example

Consider the problem of deploying a brokering network for supporting 10000 clients in a collaborative [4] fashion. Ref. [5] shows that a single broker can support up to 1500 simultaneous participants with audio streams with very good quality audio while about 400 participants can simultaneously receive video with acceptable quality. For a higher number of participants, we can employ a tree-based structure as illustrated in Figure 2. The problem lies in deploying the brokering topology suitable for supporting multiple clients. With a growing number of clients, one may wish to deploy a network of multiple brokers (For e.g., 10000 / 400 = 25 brokers in the above scenario) so that all clients may receive acceptable audio / video transmission. Further, for fault-tolerance purposes, one may also want to have multiple links between brokers such that the failure of a subset of links may not crash the entire system

B. NaradaBrokering

NaradaBrokering [6] - [9] is an open-source, distributed messaging infrastructure based on the publish/subscribe paradigm. The smallest unit of this distributed messaging substrate intelligently processes and routes messages, while working with multiple underlying communication protocols. We refer to this unit as a broker. The broker network in NaradaBrokering is based on hierarchical, cluster-based structure [6]. This cluster-based architecture allows NaradaBrokering to support large heterogeneous client configurations. The routing of events within the substrate is very efficient [9] since for every event, the associated targeted brokers are usually the only ones involved in disseminations. Furthermore, every broker, either targeted or en route to one, computes the shortest path to reach target destinations while eschewing links and brokers that have failed or have been failure-suspected.

The substrate incorporates support for both JMS and the WS-Eventing specification. Work is currently underway on incorporating support for the WS-Notification suite of specifications. The NaradaBrokering substrate also incorporates support for WS-ReliableMessaging [10] and WS-Reliability [11] that facilitates reliable messaging between Web Services. Subscription formats supported within the substrate include "/" separated Strings, Integers, <tag, value> pairs, regular expressions, XPath and SQL queries. In NaradaBrokering entities can also specify constraints on the qualities of service (QoS) related to the delivery of events. The QoS pertain to the reliable delivery, playbacks, order, duplicate elimination, global timing services, security and size of the published events and their encapsulated payloads. Additional information regarding NaradaBrokering can be found in Refs [6] - [9].

C. Related Work

The most commonly used management protocol is the Simple Network Management Protocol (SNMP) [12]. SNMP is an application layer protocol that facilitates the exchange of management information between network devices. Based on a reliable connection oriented protocol, SNMP enables network administrators to manage network performance, find and solve network problems. SNMP however lacks any authentication capabilities, which results in vulnerability to a variety of security threats such as masquerading occurrences, modification of information, message sequence and timing modifications, and disclosure. Due to lack of authentication many vendors do not implement Set operations, thereby reducing SNMP to a monitoring facility. CMIP [13], an improvement over SNMP provides improved security that supports access control, authorization and security logs. CIM [14] is an object-oriented model that represents and organizes the information in a "managed environment" while consolidating and extending existing management standards.

To make management more interoperable, the Grid community has been implementing support for Web Service Distributed Management (WSDM) by treating all managed resources as a WS-Resource. WSDM [15] based management leverages the Web Service Resource Framework [16] principles to create managed resources with specific lifetimes. Work [17] is underway to map CIM constructs to Web Service based management standards such as WSDM. Our architecture does not specify lifetimes for created resources (brokers) and we expect the broker resource to remain available and running until the machine hosting the broker goes down or the broker is explicitly killed by sending an appropriate message. The brokering network may use topic lifetime to determine the period until when a peer may subscribe to receive specific events generated by the managed entity. Further, WS Management and WSDM only specify the wire-protocol using SOAP messages for enabling management. The architecture presented in this paper implements the WS Management specification while leveraging a distributed messaging infrastructure to provide scalability to the management process.

In the area of P2P systems, deployment of peers is usually

done via some static rules and the overall topology is not easily modifiable at runtime. TreeP [18] uses a B+-Tree based topology for range querying. Baton [19] is a Peer-to-Peer (P2P) network based on balanced tree structure useful for exact and range queries. Both the systems use a fixed tree structure topology to connect individual peers. P2P systems based on distributed hash table such as Chord [20] use a bootstrap node to get a node address. Future additions automatically get address from one or more previously initialized nodes when they join the network. CAN [21] uses a similar approach where an incoming node contacts a bootstrap node to retrieve a set of randomly chosen nodes. The new node then connects to a randomly chosen node from the retrieved set. However, CAN and Chord do not take network distances in to account when creating the routing table. This may result in certain lookups resulting in overlay hops spanning the entire diameter of the network. Tapestry [22] and Pastry [23] construct and maintain locally optimal routing tables at initialization that helps reduce routing stretch.

III. ARCHITECTURE

We now describe the application of the management framework from the broker management point of view. The architecture consists of two main components, the Broker Service Adapter (henceforth, BSA) for configuring and initializing brokers and the Broker Network Manager (henceforth BNM) that functions as a client to the BSA and helps deploy specific topologies. A BSA is required because NaradaBrokering brokers are not Web Services.

We have modeled the most commonly used management actions on brokers using simple GET, PUT, CREATE and DELETE verbs from the WS – Transfer [24] specification of WS-Management. Since the broker does not create extensive log of activities, we do not require support for WS – Enumeration [25] in the BSA. A broker however may wish to send notifications on events such as broker liveness (heartbeat) and link failure which may help the BNM take decisions at runtime.

A. Broker Service Adapter (BSA)

The Broker Service Adapter (shown in Figure 3) is the component that wraps a broker and is responsible for invoking management related commands on the broker. Specifically, the Broker is the managed entity and the BSA functions as the adapter. For purposes of or architecture, we assume that the BSA instantiates a broker (when it receives a CREATE message) in the same JVM as the BSA. The BSA models various properties of the Broker as resources as specified by WS Management.

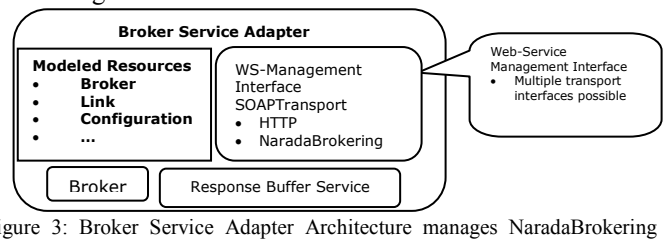

Figure 3: Broker Service Adapter Architecture manages NaradaBrokering brokers

Specifically, we model the following characteristics of the Broker interface, namely, Broker, Link and Configuration. Additional resources that have been modeled is the NetworkAddress and GatewayAddress that are required whenever a broker which does not have a network address connects to a broker which has a network address.

The event delivery is handled by the SOAPTransport interface that can either use a direct HTTP connection for sending and receiving SOAP messages or use the brokering infrastructure to deliver SOAP messages wrapped as NaradaBrokering events. The response can be sent to a specific endpoint by inspecting the ReplyTo field in the endpoint reference. Sometimes, a direct connection between the service and client is not possible, in which case responses to requested operations may not be delivered directly. In such cases, the responses are buffered by the ResponseBufferService. This approach is similar to the one illustrated in [26] and [27]. These responses can then be retrieved from the buffering service by a separate call to the service.

An advantage of using NaradaBrokering wrapped transport for delivering SOAP messages is that it allows the managed entities to scale in number. Further, NaradaBrokering supports a variety of transports for event delivery such as TCP, UDP, NIOTCP, HTTP, SSL and MULTICAST. On initialization, the BSA cycles through a list of all possible transports and connects via the first available transport. This provides fault-tolerance against unavailable network protocols.

B. Broker Network Manager (BNM)

In our initial implementation a broker network manager (Refer Figure 4) functions as a manager as well as a user of the system. This component essentially provides 2 services, (1) provide a function interface to manipulate various resources managed by the BSA and (2) provide an interface to deploy arbitrary topologies. A TopologyGenerator allows users to create application specific topologies given a set of BSA endpoints. This topology is then translated to appropriate commands and submitted to the broker network manager. In case a certain action cannot be carried out, the broker network manager throws an exception and allows the user to take corrective action.

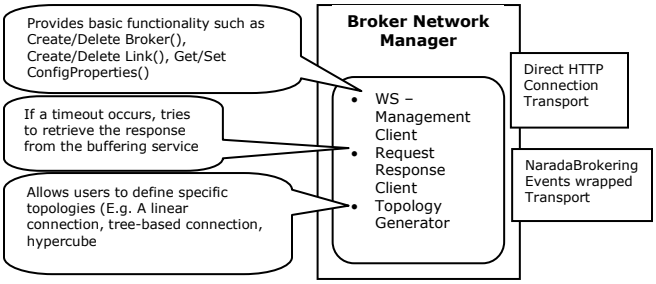

Figure 4: Broker Network Manager (aids in deployment of Broker network)

The broker network manager also has a HTTP to NaradaBrokering transport mapper that wraps SOAP messages and publishes it to a specific topic. This scheme is described in more detail in Section 2). The HTTP Mapper service can either be part of the static broker or a separate

process running on the same host as the static broker.

C. Deploying topologies

The overall architecture is illustrated in Figure 5. The system consists of a set of 3 static brokers which serves as bootstrap nodes. These nodes are only responsible for providing NaradaBrokering wrapped transport for SOAP messages. We have placed these static brokers on 3 geographically distributed machines. Specifically we have used a machine at Indianapolis, and two machines in the lab at Indiana University, Bloomington and are accessible through the domain names gridservicelocator.org, messageservicelocator.org and webservicelocator.org.

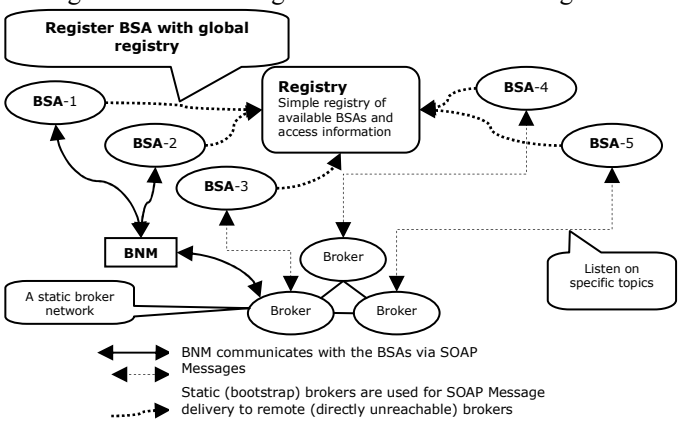

Figure 5: Overall Architecture

Consider the 5 BSAs shown in above figure. These BSAs are distributed over different machines. BSAs (1 and 2) are directly accessible and hence the broker network manager may communicate directly with these services. BSAs (3 – 5) however are possibly in different administrative domain with respect to the broker network manager. The broker network manager then leverages NaradaBrokering wrapped message delivery for sending and receiving SOAP messages. Although the broker network manager may directly behave as a client to the static brokers, we use a HTTP Mapper service to wrap SOAP Message as an NaradaBrokering Event. This process is detailed in Section 2).

1) Registering BSAs

When the BSA starts up, it can be started to either use the direct HTTP transport or leverage the brokering network to transfer SOAP messages. The advantage of the later is that BSA present in a different administrative domain can be accessed by transporting SOAP messages wrapped as NaradaBrokering Events using HTTP tunneling. Once the BSA has initialized, it registers itself in a registry service such as a UDDI registry. For our purpose, we have implemented a simple registry service that stores the endpoint address for each registered BSA.

The BSA cycles through all three static brokers trying connection to each of these brokers. This helps us provide fault-tolerance against unreachable hosts. Broker Discovery mechanism [28] may be used to find the nearest best broker to connect to. As long as the BSA can connect to at least one of the static brokers, it can be managed by the BNM.

2) Mapping topics to Endpoint References

To route SOAP messages to each BSA, the BSA creates a unique 128-bit UUID based topic during initialization. The BSA then proceeds to register this UUID as its endpoint address in the registry.

To interact with the BSA, an interested BNM publishes events on topic of the form `BSA/UUID`. However, to provide transparency of operation, we use a HTTP Mapper service that runs on the same host as the static broker. The BNM makes a normal HTTP call to the HTTP Mapper service. The `<wsa:To>` addressing header in the SOAP message is then modified as follows

```
<wsa:To>http://gridservicelocator.org:6000/12345678-1234-1234-123456789012</wsa:To>
```

This specifies that the SOAP Message is sent to the HTTP Mapper service running on `gridservicelocator.org` on port `6000` and it should be forwarded to the BSA whose `UUID` is `12345678-1234-1234-123456789012`.

The HTTP Mapper puts its topic as an immutable property in the NaradaBrokering event header while the SOAP message is copied as the payload. This event is then published on the topic `BSA/12345678-1234-1234-123456789012`. Once the response is generated by the destination BSA, the response is sent back to the HTTP Mapper service which was responsible for forwarding the SOAP message. The BSA uses the HTTP Mapper's topic represented by the immutable header property in the original event to determine the correct destination.

IV. WS – Management Based Modeling

WS-Management requires all managed resources to be identified by a `<ResourceURI>`. This has been defined in the BSA schema available at `http://www.hpsearch.org/schemas/2005/11/BSA`. Our initial model defines the following Resource URIs,

`BROKER` that identifies the broker in question

`LINK` that identifies a link between 2 brokers

`CONFIGURATION` that identifies the broker configuration with which the broker is to be initialized

`CONFIGURATIONPROPERTY` that identifies each individual property in the broker configuration

`NODEADDRESS` that refers to a node address as required by the broker when it joins a broker network

`GATEWAYADDRESS` that refers to a gateway address when a broker in cluster connects to a broker in another cluster

`BUFFEREDMESSAGE` that is used to retrieve the response to a previously sent request from the request buffering service.

The BSA is an implementation of the WS Management processor which is part of a framework for deploying WS-Management compatible management services. Our initial implementation consists of WS – Transfer and WS-Enumeration while we plan on leveraging WS – Eventing [29] provided by NaradaBrokering.

A. Management Operations

In this section we list the various management operations as defined by the BSA. Note that not all resources may support all operations. In the case where an operation is requested on a

resource which is not supported, an `UnSupportedOperation` fault is thrown. The various operations are listed in Table 1.

B. Enumeration and Eventing

WS-Management allows managed resources to enumerate large set of values from managed containers using the interface defined in WS-Enumeration. Although we provide support in the framework for WS-Enumeration, we do not see any suitable requirement in BSA that requires us to provide any implementation. WS-Management also allows managed resources to publish events that give regular updates on the status of the resources. Although our initial implementation does not provide any support for WS-Eventing, we are currently working on integrating support for WS-Eventing by leveraging the WS-Eventing container recently added in NaradaBrokering.

TABLE 1 SUMMARY OF VARIOUS OPERATIONS MODELED IN THE BROKER SERVICE ADAPTER

Resource	Supported Operations	Operation Detail
BROKER	Create	Initializes a broker using the current configuration and returns a unique BrokerID. This broker is initialized in the same JVM as the BSA.
	Delete	Kills the broker identified by the BrokerID
	Get	Retrieves information about the broker specified by the BrokerID
LINK	Create	Creates a link by trying a connection to the specified broker using the specified transport. On success, returns a LinkID corresponding to the link
	Delete	Deletes the link identified by the LinkID
CONFIGURATION	Get	Retrieves the value(s) of specified configuration property
CONFIGURATION PROPERTY	Put	Replaces the value(s) of the specified configuration with a new value. If the broker has already initialized we do not allow this operation.
NODEADDRESS GATEWAY-ADDRESS	Create	When a broker is initialized and connects to an existing broker in the broker network, this operation enables the new broker to request a node address from the broker to which it is connected to. This is a required step because of the inherent design of NaradaBrokering.
BUFFERED-MESSAGE	Get	Retrieves a previously buffered response

C. Discovery of Managed Resources

WS – Management Catalog [30] defines a metadata format for the discovery of management functionality of resources. Current implementation provides static binding to the BSA's WS-Management interface in the BNM. We provide discovery by means of a simple registry service that lists the endpoints associated with individual BSAs.

V. RESULTS

WS-Management relies heavily on SOAP 1.2 specification for conveying various faults and details associated with exceptions. During development we noted that the Soap with Attachments API for JAVA (SAAJ API) library provided with JDK 1.4 implements SOAP 1.1 while support for SOAP 1.2 is provided in the newer releases of SAAJ (version 1.3 EA) which is shipped with Java WSDP 2.0. In order to provide maximum compatibility with various leveraged 3rd party softwares we have implemented our own SOAP message sender and receiver.

We benchmarked our architecture with 2 topologies. In both cases, we employed as simple topology generator that generated links such that the ith broker is connected to (i-1)th broker (for all i > 1). Table II summarizes the machine configuration.

TABLE II TEST MACHINE CONFIGURATION

Machine	Machine Specification	Java Version	Network
Benchmarking machine trex.ucs.indiana.edu	Linux, 2.6.5-7.155.29-default, Pentium 4 2.53 GHz 512 MB RAM	Java HotSpot(TM) Client VM (build 1.4.2_03-b02, mixed mode)	Linked via 100 Mbps link
Grid Farm machines in CGL, Bloomington (gf1.ucs.indiana.edu – gf8.ucs.indiana.edu)	Linux 2.4.22-1.2188.nptlsmp, 4 - Intel Xeon 2.4 GHz CPUs, 2 GB RAM		
Home Machine: For Testing between different administrative domains (behind a home DSL router)	Athlon 64 3400+ Processor 2.41 GHz, 1 GB RAM Windows XP Pro w/ SP2	Java HotSpot(TM) Client VM (build 1.4.2_08-b03, mixed mode)	Linked via a 1.5 Mbps Home DSL network

For manipulating XML, we leveraged Apache XMLBeans toolkit (version 2.0.0). Table III lists the specifications that were implemented during the development phase.

TABLE III VERSIONS OF WEB SERVICE RELATED SPECIFICATIONS THAT WERE IMPLEMENTED

WS - Specification	Spec Release Date
WS Management	June 2005
WS Transfer	Sep 2004
WS Enumeration	Sep 2004
WS Addressing	Aug 2004
SOAP	Version 1.2

We tested the time it takes to deploy a broker topology consisting of 8 brokers. This test uses direct HTTP transport since all machines are accessible from the machine running the BNM. We first set different configuration in each BSA and then initialize all 8 brokers. Once all of them have initialized, we also time the process of creating links between brokers. Finally we shutdown all brokers. We have used a high resolution timer which reports times to microsecond accuracy to time the various operations.

The average timing was found by running the test several times, removing outliers (to remove effects of initialization costs) and selecting last 10 readings. We report the average cost and the standard deviation of each step. The actual time is the time it takes to execute the operation while the total time includes the actual time plus the overhead in marshalling / unmarshalling XML and transporting the request and reply between communicating entities. We also compute the

overhead as the difference of the total and actual time. The last column shows the average overhead per broker involved in the operation. We note that the average overhead is (about 72 mSec) consists of marshalling the SOAP Envelope, transporting the SOAP message, unmarshalling the SOAP Envelope and extracting the actual request / response. Table IV shows the results. The "Actual Time" reported for the `GetConfiguration` operation is `0` since, in the implementation, the BSA always updates the configuration whenever an operation occurs. There is no special processing done during this call and the only work done comprises of marshalling the already existing information as XML and shipping it across to the BNM.

TABLE IV TIMINGS AND OVERHEAD WHEN DEPLOYING A NETWORK OF 8 BROKERS

Action	Number of Brokers	(All values reported are in milliseconds) Total Time benchmarked on trex.ucs.indiana.edu		Actual Time benchmarked on (benchmarked on Gridfarm machines)		Over-head	Avg. Over-head
		Mean	Std. Dev.	Mean	Std. Dev.		
Set Config	8	717.24	118.4	128.78	7.93	588.46	73.56
Create Broker	8	729.02	81.59	237.79	42.87	491.23	61.40
Get Config	8	488.88	132.7	0	0	488.88	61.11
Create Link	7	746.52	343.3	128.07	14.92	618.45	88.35
Get Node Address	7	594.69	106.03	112.12	13.41	482.57	68.94
Delete Broker	8	828.59	279.54	228.43	155.59	600.16	75.02

The second test involves using the NaradaBrokering wrapped transport for delivering SOAP Messages. Here we tested with 3 brokers and 2 links. Note that using this particular method of transport, we can access BSAs in different administrative domains. Due to the restriction on the need of IP address for connecting brokers, it may not be always possible for brokers in different administrative domains to be connected together. In this case we propose using a proxy broker that sits in the open network and links the two different broker networks. A set of brokers can however be managed from a different administrative domain. We believe that with the proliferation of IPv6 address space, this problem may be resolved to some extent. Table V shows the timing associated with this topology.

To simulate heterogeneous environments with restricted transports and open ports, we turn off the relevant transport links in the bootstrap broker hosted on `www.webservicelocator.org`. The average overhead was found to be significantly higher than the direct HTTP connection. This was due to wrapping of SOAP message using NaradaBrokering, which resulted in multiple hops between the BNM and the BSA.

We also observe that in both cases all of the operations listed above are executed serially. Certain operations such as setting individual configurations and starting brokers are independent of each other and could be parallelized in order to decrease the total time to deploy the broker network.

TABLE V TIMINGS AND OVERHEAD WHEN DEPLOYING A NETWORK OF 3 BROKERS WITHIN A DIFFERENT ADMINISTRATIVE DOMAIN (BEHIND A HOME DSL ROUTER)

Action	Number of Brokers	(All values reported are in milliseconds) Total Time benchmarked on gf4.ucs.indiana.edu		Actual Time benchmarked on home machine		Over-head	Avg. Over-head
		Mean	Std. Dev.	Mean	Std. Dev.		
Set Config	3	517.59	61.2	1.56	0.64	516.03	172.01
Create Broker	3	512.16	46.87	63.61	3.61	448.55	149.52
Get Config	3	536.07	27.85	0	0	536.07	178.69
Create Link	2	308.35	22.78	22.36	4.52	285.99	143.00
Get Node Address	2	290.22	33.14	1.55	0.27	288.67	144.34
Delete Broker	3	415.67	46.36	21.45	1.74	394.22	131.41

We also conducted preliminary tests to see how many clients a single broker can support before it saturates and is unable to deliver events. Note that in our architecture there is typically only one sender (publisher) and one receiver (subscriber) per message being sent. In Ref. [5] there are potentially multiple subscribers per publisher. In our tests we ran one broker on `www.webservicelocator.org`. WS-Management recommends that if a `MaximumEnvelopeSize` is specified in the request then it should have a value of at-least 8192 octets which is the safe minimum in which all faults can be reliably encoded. We noted that a single broker can support about 300 simultaneous publishers publishing at a rate of 1000 messages per second, each message of size 8192 bytes. The latency was found to be about 30ms (average) and about 95 ms (maximum) over 150 runs.

A. Addressing Requirements

In this section we show how each of the desiderata mentioned in Section I-B is addressed through this architecture.

Since the architecture uses a distributed messaging substrate to handle communication between manageable resources and managers, we can enable remote management of resources.

Our system uses bootstrap nodes based on the NaradaBrokering messaging middleware for delivering messages. NaradaBrokering supports a variety of protocols such as TCP, NIOTCP, UDP, HTTP and tunneling through

firewalls. The service adapter is responsible for connecting to an available bootstrap node by leveraging one of the supported transport protocols. Once the service adapter is connected to a bootstrap node the managed entity served by this service adapter can be managed. Thus the architecture can enable management by traversing through firewalls and NAT devices.

Our management architecture is based on the Web Services paradigm. Evolving management specifications such as WS-Management and WSDM allow management to be extensible by providing the basic management framework consisting of common operations such as creation and deletion of resources and setting and querying resource specific properties. An implementation is free to extend beyond the basic operations by supporting functionality specific to the target resource. Discovery of available functionality is done via a Web Service discovery mechanism such as WS-Management Catalog.

Service adapters (BSA, in our case) are responsible for connecting to a bootstrap node. In presence of firewalls or blocked ports, alternate means of transport may be used. The architecture ensures management if the BSA can connect to at-least one of the bootstrap nodes via available protocols. This ensures administrative scalability. Further we noted that a single bootstrap node (message broker) can handle about 300 simultaneous clients before it saturates. We can achieve scalability in terms of the number of resources managed by adding more bootstrap nodes and managers.

This paper specifically targets quality of service by addressing fault-tolerance against unavailable protocols. The architecture is composed of several crucial components. Bootstrap nodes are assumed to be highly failure resilient and we assume that a failed node can be brought up in a finite amount of time. If a bootstrap node fails, the service adapters may switch to another bootstrap node. Although one would employ multiple managers to improve scalability of the system, the management process would run as long as there is at-least one manager process running. Data sharing is handled through the registry which we expect to be fault-tolerant thorough some implicit means.

VI. Conclusion and Future Work

In this paper we have presented our scheme for Web Service based management interface for easy configuration and deployment of middleware components. We have shown how management can be made scalable and fault-tolerant in presence of heterogeneous environments and have presented results associated with a prototype of the management architecture that manages the NaradaBrokering messaging infrastructure. The costs obtained are one-time initialization costs during deployment of the network and are hence quite acceptable.

We are currently working on implementing a heartbeat mechanism based on WS-Eventing to monitor broker liveness and various other events that enable the BNM to take runtime decisions. We plan on investigating support for WS – Management Catalog for describing BSA interaction schema and set of managed resources. The WS Management and WS Distributed management specifications are converging [30] towards a common specification. Our current work is based on WS Management specification and we plan to incorporate the changes as the new specification is published. We also plan on researching suitable means of effectively providing multiple managers to manage a large set of resources and conducting detailed tests that stress the scalability of the architecture.

Acknowledgement

The authors would like to thank Prof. Gordon Erlebacher (erlebach@csit.fsu.edu) for his critique on the HPSearch project.

References

[1] *Web Service Management (WS – Management)*, Sun, Microsoft, Intel, et al., June 2005. Available from https://wiseman.dev.java.net/specs/2005/06/management.pdf

[2] Network Topologies: http://en.wikipedia.org/wiki/Network_topology, Visited Feb 6, 2006

[3] Gurhan Gunduz, Shrideep Pallickara and Geoffrey Fox *An Efficient Scheme for Aggregation and Presentation of Network Performance in Distributed Brokering Systems* (To Appear) in Journal on Systemics, Cybernetics and Informatics 2004. Available from http://grids.ucs.indiana.edu/ptliupages/publications/PAS-Framework-journal.pdf

[4] GlobalMMCS (Global Multimedia Conferencing System), Project page: http://www.globalmmcs.org

[5] Ahmet Uyar, *Scalable Service Oriented Architecture for Audio/Video Conferencing*, Ph.D. Thesis, May 2005

[6] Shrideep Pallickara and Geoffrey Fox. *NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids*. Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware-2003.

[7] Geoffrey Fox, Shrideep Pallickara, Marlon Pierce, Harshawardhan Gadgil. *Building Messaging Substrates for Web and Grid Applications*. In special Issue on *Scientific Applications of Grid Computing* in Philosophical Transactions of the Royal Society, London, Volume 363, Number 1833, pp 1757-1773, August 2005.

[8] Geoffrey Fox and Shrideep Pallickara. *Deploying the NaradaBrokering Substrate in Aiding Efficient Web & Grid Service Interactions.* Invited paper for Special Issue of the Proceedings of the IEEE on Grid Computing. Vol 93, No 3. pp 564-577. March 2005.

[9] Shrideep Pallickara and Geoffrey Fox. *On the Matching Of Events in Distributed Brokering Systems*. Proceedings of IEEE ITCC Conference on Information Technology. April 2004. pp 68-76 Volume II.

[10] Web Services Reliable Messaging Protocol (WS-ReliableMessaging) ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200403.pdf

[11] Web Services Reliable Messaging TC WS-Reliability. http://www.oasis-open.org/committees/download.php/5155/WS-Reliability-2004-01-26.pdf

[12] Simple Network Management Protocol (RFC: 1157, http://www.ietf.org/rfc/rfc1157.txt l) Also http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.htm

[13] The Common Management Information Services and Protocols for the Internet, http://www.ietf.org/rfc/rfc1189.txt

[14] Common Information Model, http://www.dmtf.org/standards/cim/

[15] *Web Service Distributed Management (WSDM)*, HP ; et al. Refer: http://devresource.hp.com/drc/specifications/wsdm/index.jsp

[16] The Web Services Resource Framework. http://www.globus.org/wsrf/

[17] Proposal for a CIM mapping to WSDM, IBM. Available from ftp://www6.software.ibm.com/software/developer/library/ws-wsdm.pdf

[18] B. Hudzia, M-T. Kechadi, and A. Ottewill, "*TreeP: A Tree-Based P2P Network Architecture*", International Workshop on Algorithms, Models and tools for parallel computing on heterogeneous networks (HeteroPar'05), Boston, Massachusetts, USA, September 27-30, 2005.

[19] H.V. Jagadish, Beng Chin Ooi, Quang Hieu Vu, *BATON: A Balanced Tree Structure for Peer-To-Peer Networks*, In Proceedings of the 31st VLDB Conference, Trondheim, Norway, 2005

[20] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek and Hari Balakrishnan, "*Chord: A scalable peer-to-peer lookup service for internet applications*" in Proceedings of SIGCOMM, Aug 2001.

[21] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp and Scott Schenker, "*A scalable content-addressable network*" in Proceedings of SIGCOMM, Aug 2001

[22] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John Kubiatowicz *Tapestry: A Resilient Global-scale Overlay for Service Deployment,* IEEE Journal on Selected Areas in Communications, January 2004, Vol. 22, No. 1

[23] Antony Rowstron and Peter Druschel, "*Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*" in Proceedings of Middleware, Nov 2001

[24] *Web Service Transfer (WS – Transfer)*, Microsoft et al., September 2004. Available from http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-transfer.pdf

[25] *Web Service Enumeration (WS – Enumeration),* Microsoft, BEA, et al., September 2004 Available from http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf

[26] Aleksander Slominski, Alexandre di Costanzo, Dennis Gannon, and Denis Caromel. *Asynchronous Peer-to-PeerWeb Services and Firewalls*. (To Appear) In *7th International Workshop on Java for Parallel and Distributed Programming (IPDPS 2005)*, April 2005

[27] Kyle Brown, et al., *Web Services Polling*, Oct 2005. Available from http://www.w3.org/Submission/ws-polling/

[28] Shrideep Pallickara, Harshawardhan Gadgil, Geoffrey Fox; *On the Discovery of Brokers in Distributed Messaging Infrastructure*, (To Appear) In Proceedings of the IEEE Cluster 2005 Conference. Boston, MA

[29] Web Services Eventing (WS – Eventing), Microsoft, IBM & BEA, August 2004. Available at http://ftpna2.bea.com/pub/downloads/WS-Eventing.pdf

[30] The Web Services Management Catalog, Sun et al., June 2005 Available from http://developers.sun.com/techtopics/webservices/management/WS-Management_Catalog.June.2005.pdf

[31] HP, IBM, Intel and Microsoft, *Toward Converging Web Service Standards for Resources, Events, and Management*, Available from http://msdn.microsoft.com/library/en-us/dnwebsrv/html/convergence.asp