A Scripting based Architecture for Management of Streams and Services in Real-time Grid Applications

Harshawardhan Gadgil[1], Geoffrey Fox[1], Shrideep Pallickara[1], Marlon Pierce[1],
Robert Granat[2]
[1]Community Grids Lab, Indiana University, Bloomington IN – 47404
hgadgil@cs.indiana.edu, (gcf@, spallick@, marpierc@)indiana.edu
[2]NASA, Jet Propulsion Lab, Pasadena, CA – 91109
granat@aig.jpl.nasa.gov

Abstract

Recent specifications such as WS-Management and WS-Distributed Management have stressed the importance of management of resources and services and propose methods towards querying Web Services to gather the meta-data associated with these services. Management often entails system setup, querying system metadata, manipulation of system parameters at runtime and taking actions based on the system parameters to tune system performance.

Real-time applications require rapid deployment of application components and demand results in real-time. In this paper we present the HPSearch system which enables dynamic management of the system including both streams and Web Services, and rapid deployment of applications via a scripting interface. We illustrate the functioning of the system by modeling a data streaming application and rapidly deploying the system and application components.

KEYWORDS: System Management, Data streaming, Grids, Scripting, Managing middleware

1 Introduction

Many applications require filtering of data before they can process the data. These data filtering applications are often deployed statically while the data is transferred using files. Each application reads data from one or more input files and creates one or more output files which are then transferred to the next application. An obvious disadvantage of this solution is the inefficient utilization of space (for storing temporary data) and time (for getting the entire data processed); a feature not desired in real-time applications. Real-time applications require data processing and results in real-time and usually employ data streaming to achieve high performance. In a world where every data source is potentially distributed, filtering this data to suit the needs of a particular application at run-time is a challenge and is usually addressed by employing distributed data processing.

Applications which generate or work on huge data sets such as Audio / Video applications and earthquake modeling in critical infrastructure systems require data filters to reduce the size of data to achieve high performance. Data filtering may also be employed to rearrange data in a specific format before data processing can begin.

With the growing complexity of systems, the management of these systems has increased in importance. We view system management in two parts namely (1) Deployment of system components (2) Querying dynamic system meta-data for anomalies and changing the parameters of the system to improve the performance of the system.

To address the issues of system management and rapid deployment of distributed services for real-time processing, we introduce the HPSearch system. HPSearch helps us to setup distributed service applications and manage the data streams using a high performance messaging substrate, NaradaBrokering (Refer Section 1.1). HPSearch takes the view that a stream of data can be broken into small messages and thus data transfer can be made using messaging. HPSearch also presents interfaces to quickly create data filtering applications or create pluggable components from existing applications and expose them as a Web Service.

The rest of this paper is organized as follows. We continue the discussion below with an introduction to the NaradaBrokering system. Section 2 describes the HPSearch architecture. Section 3 introduces our test application and illustrates how we can use the scripting interface to setup the system and deploy the application. Section 4 presents related work. In sections 5 and 6 we outline future work and conclusions respectively.

1.1 NaradaBrokering

NaradaBrokering [1, 2, 3] is an event brokering system designed to run on a large network of cooperating broker nodes. Communication within NaradaBrokering is asynchronous and the system can be used to support different interactions by encapsulating them in specialized events. NaradaBrokering guarantees delivery of events in the presence of failures and prolonged client disconnects, and ensures fast dissemination of events within the system. Events could be used to encapsulate information pertaining to transactions, data interchange, system conditions and finally the search, discovery and subsequent sharing of resources.

We summarize some of the important features of NaradaBrokering as follows

- Implements high-performance protocols (message transit time of 1 to 2 ms per hop)
- Order-preserving *optimized* message transport
- Quality of Service (QoS) and security profiles for sent and received messages
- Interface with reliable storage for persistent events, reliable delivery via WS-Reliable Messaging [4]
- Support for different underlying transport implementations such as TCP, UDP, Multicast, SSL, RTP, HTTP
- Discovery Service to find nearest brokers / resources

2 HPSearch

We have been developing HPSearch [5] as an extension to an existing scripting language that binds Uniform Resource Identifiers (URI) to the scripting language. By binding URI as a first-class object we can use the scripting language to manage the resource identified by the URI. Thus, we can read information in the form of messages from arbitrary streams (specified by a topic URI) such as performance monitoring services and service metadata publishers. Other possible actions include reading databases and reading from or writing to files and sockets. We have implemented a simple scheme to map the results of database queries to XML for streaming purposes.

HPSearch uses NaradaBrokering to route data between components of a distributed application. This data transfer is handled transparently by the HPSearch runtime using NaradaBrokering. Further, each of the distributed components is exposed as a Web Service which can be initialized and steered by simple SOAP requests.

HPSearch is currently implemented using Mozilla Rhino [6], a Java based implementation of Javascript although other scripting languages like Python or Jython may be supported in the future. Rhino allows us to create custom *host-objects* that help to dynamically access the host system. This feature can be useful to create objects that help manipulate data streams and aid system management tasks such as discovering services and data sources, initializing these services and steering them.

2.1 Architecture

Figure 1 shows the HPSearch architecture and its components. As shown in the figure, HPSearch consists of one or more *HPSearch Kernel*. Each of these kernels consists of a management and control shell (currently a Javascript based shell), a Service Manager and other system objects. We also provide an interface (WSProxy) to compose services that can process data in a stream. The WSProxy ensures that the data flow between the components of any application flows directly between the components and does not involve the HPSearch kernel.

2.2 Kernel Components

Shell: The shell is a command line interface based on Mozilla Rhino. Scripting provides numerous advantages as observed by [7]. The Shell contains the various system management objects that help us to setup and manage the system.

NaradaBrokering Objects: These objects are specific to the NaradaBrokering system and aid system management tasks in NaradaBrokering. We illustrate below, two of the currently implemented objects, namely `NaradaBroker` and `PerfMetrics`.

`NaradaBroker` is a front-end for instantiating new brokers and links between brokers for system setup or for efficient routing. This may be combined with the performance metrics for creating new brokers and links between existing brokers to achieve higher throughput by avoiding busy routes. For example the following code creates a linear topology consisting of 3 brokers as shown in Figure 2.

```
b = new NaradaBroker
            ("school.cs.indiana.edu");
b.create("");
b_connLink = b.connectTo
      ("156.56.104.170",
      "5045", "t", "");
b.requestNodeAddress(
      "156-56-104-176.dhcp-
            bl.indiana.edu:5045",
      "0");

c = new NaradaBroker
```

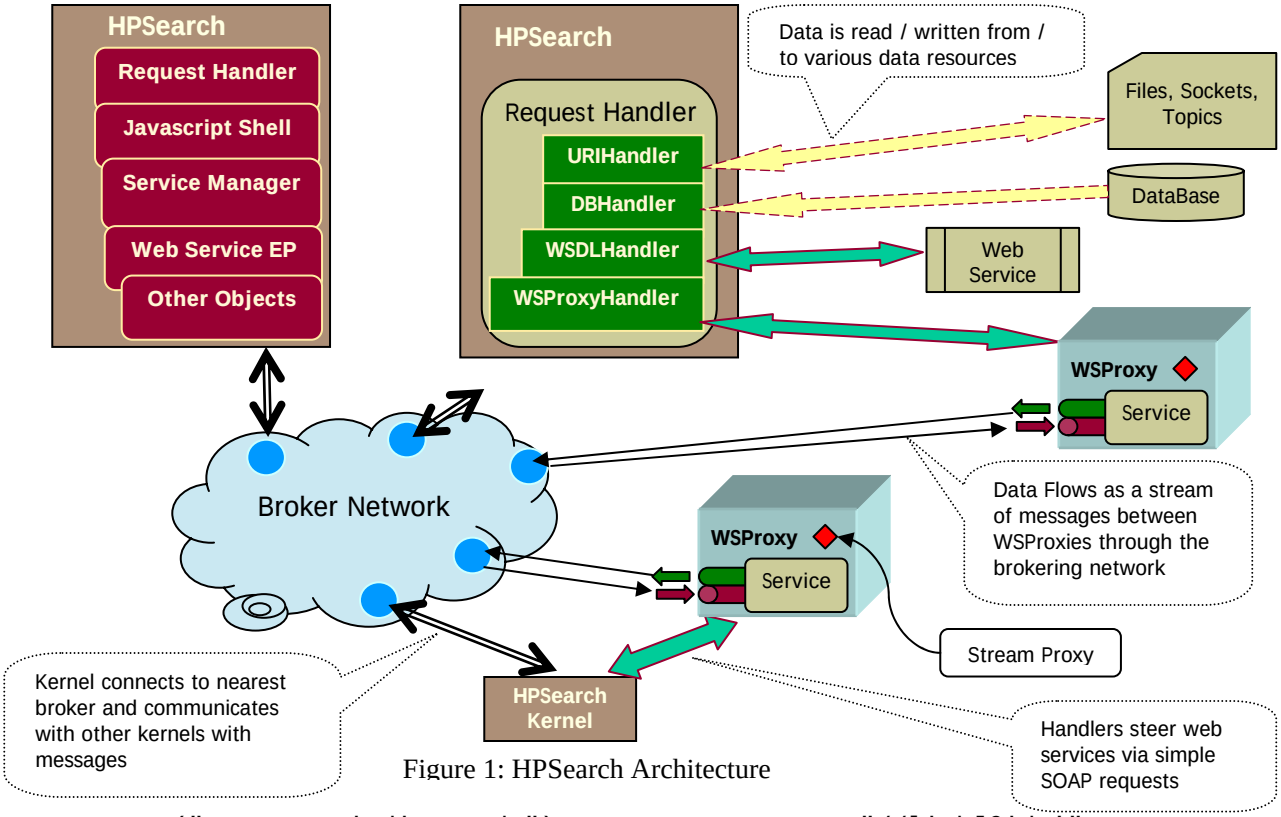

Figure 1: HPSearch Architecture

```
        ("trex.ucs.indiana.edu");
c.create("");
c_connLink = c.connectTo
      ("156.56.104.170",
      "5045", "t", "");
c.requestNodeAddress(
      "156-56-104-176.dhcp-
            bl.indiana.edu:5045",
      "0");
```

The `PerfMetrics` object reads the performance metrics published by NaradaBrokering's Performance Monitoring Service on a specialized topic for performance data (such as `cgl/narada/perfdata`). The performance monitoring service regularly sends messages containing the aggregated metrics on the existing brokers and links in the system. These metrics are stored by the HPSearch engine, which allows querying of these metrics using the scripting interface. As an illustration, the following code queries the accumulated metrics to find a link with an average latency greater than 5.0 and then re-queries to find the jitter for such a link.

```
badLink = PerfMetrics.query(
      "//link[avgLatency > 5.0]/@id");

for(i = 0; i < badLink.length; i++) {
      jitter = PerfMetrics.query(
            "//link[@id='" +
            badLink[i] +
            "']/jitter");
      Sys.print("Link: " + badLink[i]
            + "Jitter: " + jitter);
}
```

Here we can use XPath expressions to query the performance metrics.

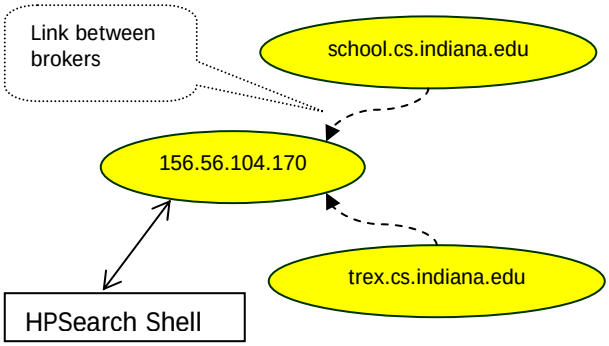

Figure 2: Creating Brokers and Links

Service Manager: In order to set up an application using distributed services, we create host-objects to describe the components of the application. These components may be discovered dynamically at runtime using one or more discovery mechanisms [8, 9]. The Service Manager then distributes the task of handling these components to other

HPSearch engines. This may be done to achieve load-balancing when necessary.

Request Handler: On receiving a service description, the HPSearch kernel spawns a RequestHandler. The request handler contains handlers for initializing and managing a specific type of resource as identified by the URI in question. Thus we have handlers for handling database requests (Read data from database as a result of an SQL query), handling generic URIs (files, sockets, topics), handling simple web-service invocations (parse a given WSDL and invoke an operation) and handle WSProxy units. The request handler's job is to monitor the execution of the service assigned to it. This involves monitoring notifications and errors generated by the resource being handled and taking appropriate action. Finally upon completion of the resource's job, the request handler notifies the kernel of the job completion.

2.3 WSProxy

The WSProxy (Web Service Proxy) is an Apache AXIS [10] based wrapper which facilitates wrapping existing programs as Web Services. WSProxy provides Java interfaces to compose filtering applications in two flavors, viz. *Runnable* and *Wrapped*.

The *Runnable* interface is used to create data filtering applications that can be steered by simple SOAP requests to the Web Service. This interface gives more control over the steering of the service and is useful for developing data processing applications that read a block of data, process this data and send out the results repeatedly.

The *Wrapped* interface is used to wrap existing applications (For e.g., executables, Perl / Matlab scripts). This interface provides lesser degree of control on the steering and is limited to starting and stopping the service only.

The WSProxy also wraps data input and output streams. WSProxy views input and output URI as topics to which it subscribes on behalf of the service. Furthermore as events are delivered to the WSProxy by the brokering system, the WSProxy buffers this data which then constitutes the input stream for the application. Similarly, when the application writes data out, the data is packed in events and sent to the brokering system which is then routed to its next destination. Thus the processing code is just presented with the input and output streams and the actual data streams are handled transparently.

The WSProxy also contains a component called *StreamProxy* that helps to negotiate ideal transport characteristics whenever possible.

We present below a proposed architecture of the StreamProxy and discuss how it can help in optimizing transport characteristics for high performance data transfer.

2.4 Data Stream Negotiation

Recent specifications such as WS-Transfer [11] and WS-Enumeration [12] that have been proposed allow transferring service related data, resource properties and other sequences of XML elements such as logs, message queues or other linear information models using SOAP based protocol. SOAP [13] provides a lightweight protocol to exchange information in a decentralized, distributed environment. WS-Enumeration defines a simple SOAP based protocol that allows the data source to provide a data abstraction called *enumeration context* which represents a logical cursor through a sequence of data items. The XML element information can then be transferred using this *enumeration context* over a span of one or more SOAP messages.

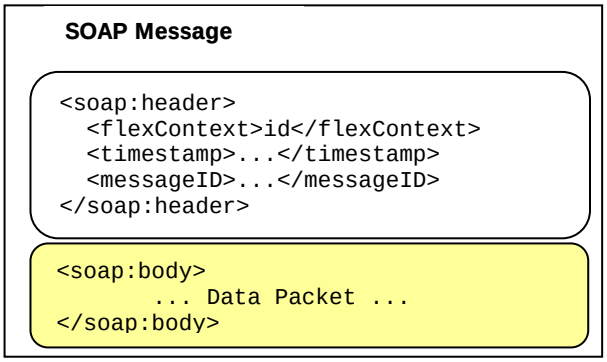

Figure 3: Flexible Representation

We present below a scheme to show how Web Services can transport streams of messages. This only works for streams (sets of messages) but this is perhaps the most important case where performance is needed. The essential ideas are as follows

- The SOAP header for messages in a stream share unchanging data which needs to be transported only once
- One negotiates using conventional SOAP over HTTP regarding several issues such as the optimal representation, transport protocol and firewall/network Quality of Service (QoS) issues
- Data is transported on a different (in general) channel in optimized fashion
- The optimized transport supports Web Service Reliable Messaging so one can use UDP based protocols with application layer reliability and flow control.
- This mechanism can support Web Service security

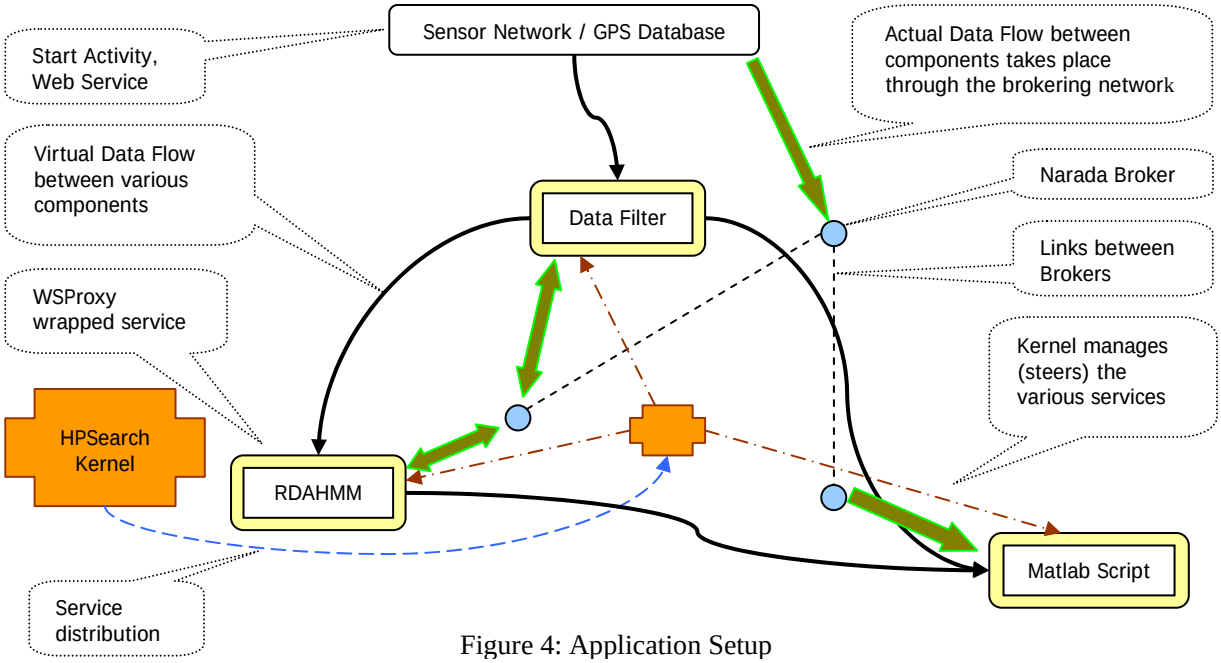

Figure 4: Application Setup

To realize the maximum performance and high throughput in case of a distributed data flow, the WSProxy creates a *StreamProxy* component which would allow the data transfer using SOAP messages as explained above. *StreamProxy* would negotiate the best possible transport characteristics for a particular application. This negotiation occurs with the NaradaBrokering substrate and is dictated by reliability, volume, security and the rate of data transfer required by a particular component. Finally, data can be sent using simple SOAP requests containing the header (contains the message and context ids) and the body (containing the data).

This negotiation usually proceeds by exchanging simple SOAP messages between the component and the substrate. This negotiation can be similar to the negotiation outlined in WS-Secure Conversation [14] or creating a context data structure as outlined in WS-Context [15]. At the end of negotiation both end-points have agreed upon various features such as the type of transport to use for data transfer (e.g. TCP, UDP), whether to use security, security tokens (if applicable) among other features of the transport. This data is called as the *FlexibleRepresentationContext*. This context contains sufficient information to process the transport of data in full compliance with SOAP semantics.

Following the negotiation, the two endpoints create a pathway depending on the agreed protocols. The data is then sent on this pathway. Further, each packet of data being sent is tagged with a *FlexibleRepresentationContextToken* that uniquely identifies the pathway characteristics. This would allow the context to be located and referenced. Now to send data using SOAP (or as stream of messages, in general), we simply include the context token, message ids and timestamp information in the SOAP header and the data in the SOAP body as shown in the Figure 3.

3 Test Application

To demonstrate the use of our system, we considered modeling the Regularized Deterministic Annealing Hidden Markov model (RDAHMM) data analysis tool in a distributed data flow. RDAHMM [16] is a time series analysis program that uses a fit of a hidden Markov model (HMM) to the data to determine, on a statistical basis, the different modes of the system and their probabilistic descriptions. It employs a regularized deterministic annealing expectation-maximization algorithm to optimize the model fit; this method greatly reduces the risk of producing a sub-optimal solution for complex, unconstrained time series data sets.

Our application consists of a GPS database that contains surface displacement time series collection by the Southern California Integrated Geodetic Network (SCIGN). The GPS database gives out the readings which contain the estimate and error values along with the date and time of observation and the name of the observation station. This data needs to be filtered to obtain only the estimate and error values. The filtered data is then analyzed by the RDAHMM application and the results of the analysis along with the filtered data are fed to a

Matlab script to generate a graphical output that allows a user to easily identify the modes in the time series and determine which points or segments of interest lie in each mode.

3.1 Setup

The test application setup is shown in Figure 4. Using the WSProxy interfaces, we wrapped the GPS data filter, the RDAHMM application and the Matlab script. The WSProxy wrapper manages the data streams on behalf of the wrapped services.

In order to quickly compose the above application we also make use of the HPSearch's administrative functions. An important functionality is deploying brokers and the links between them for rapid application deployment. For our test application we used three brokers in a linear fashion as shown in Figure 4.

The NaradaBrokering messaging middleware provides us with a host of features such as reliable delivery, multicasting, message level security and a variety of transport protocols such as RTP, UDP and TCP. Depending on the requirements of a particular application, we can use different protocol for every data stream link using flexible negotiation as discussed in Section 2.4. The stream negotiation is important since it would allow us to use the most optimal transport suited to a particular application to achieve processing in *real-time* or *near real-time*.

For our test data, we simulated a sensor data source that outputs the date, time and the observed values. Since the RDAHMM application requires only the observed values, we install a filter (which is again a WSProxy based *Runnable* filter that reads each observation, extracts the relevant information and send the output to the RDAHMM application). The RDAHMM process is executed on these observations and the results sent to the visualization service which creates a graph by combining the results and observations.

3.2 The HPSearch Script

The HPSearch shell provides a suite of host-objects to construct the data flow application and execute it. The initialization works as follows.

First we deploy a virtual broker network (shown by circles in Figure 4). A partial script to achieve the same is shown below

```
b = new NaradaBroker(
            "school.cs.indiana.edu");
b.create("");
b.connectTo("156.56.104.176",
            "5045", "t", "");
b.requestNodeAddress(
      "156-56-104-176.dhcp-
            bl.indiana.edu:5045",
      "0");
```

This creates a broker on the host `school.cs.indiana.edu` and creates a link between it and the broker on host `156.56.104.176`. We can deploy the other broker in a similar fashion (following the discussion in Section 2.2). The next step is to deploy the actual application. The steps followed are as follows.

The first step is to define the data streams used in the process. Here we use the Non blocking TCP as the default transport.

```
gpsDataTopic = "topic:///GPSData";
gpsFilteredData =
      "topic:///GPSFilter/filteredData
";
gpsComputedData =
      "topic:///GPSFilter/computedData
";
```

Then we define each data processing element and the service parameters. For e.g. the RDAHMM service is defined as follows

```
rdahmmfilter = "processes.RDAHMM";
rdahmmfilterLoc =
"http://trex.ucs.indiana.edu:6500/axis
/services/WSSConnector?wsdl";
rdahmm = new WSProxyResource(
      rdahmmfilter,rdahmmfilterLoc);

rdahmm.setInput(gpsFilteredData);
rdahmm.setOutput(gpsComputedData);

rdahmm.setParameter("OPTIONS",
      "-D 3 -N 2 -output_type gauss –
regularize
      -omega 0 0 1 1.0e-6 -anneal –
annealstep
      0.01 -seed 1");

rdahmm.setParameter("VERBOSE", "NO");
```

Here we use a static location of the service's WSDL file. However we plan to use the Discover host-object when implemented. This would serve as a front-end to a discovery service, either based on NaradaBrokering or some external registry service. The rest of the services are similarly defined. The `WSPRoxyResource` is an HPSearch object that encapsulates the description of the WSProxy service.

Finally after all the components of the flow are defined we create a Flow object and start the flow as follows.

```
f = new Flow();
f.addStartActivities(db);
f.addComponents(filter, rdahmm, viz);
f.start("1");
```

This step submits the flow component description to the *Flow Handler*, which distributes the tasks to distributed *HPSearch engines* using the *Task scheduler*. Here we also distinguish between the start activities and the normal components of the application. The start activities put the data into the data flow and are started after all the components have been initialized so that these services are ready to process the data as it comes through.

Thus in our example the filter, RDAHMM, and the visualization services are automatically started. Once the shell's flow handler receives a confirmation for the initialization of these components, it signals the data source to start sending the data. The data gets processed at various steps as it percolates through various services.

Further using the scripting interface we can query the flow status.

```
Sys.print("Flow Status: " +
            f.status())
```

4 Related Work

The art of scripting has been popular for quite some time as a means of rapid application deployment medium. Scripting presents a higher-level application interface and glues together existing applications. Scripting has proved to be a useful tool in the area of system management as shown by the Perl scripting language.

Browsers employ scripting languages such as VBScript or Javascript to do application and system related tasks such as checking forms for errors before submission and reacting to user input.

Scripting systems such as Sash from IBM [17] allows users to quickly create and deploy applications (called Weblications) to perform a variety of tasks from reading from databases and LDAP registries to invoking Web Services while providing a GUI based interface.

Recently, efforts such as WSRF:Lite and OGSI:Lite [18] have used Perl as a hosting environment to rapidly compose and deploy Grid services. Jython is a popular scripting language and is used in various applications such as GeoDISE [19]. Finally we have Matlab which is used by majority of scientists for algorithm development, data visualization, data analysis, and numerical computation. GeoDISE provides Matlab and Jython interfaces for deploying applications on the grid via Globus [20].

Our approach uses Javascript to manage the NaradaBrokering system and also deploy applications in a Service-Oriented architecture. By binding URIs to the scripting language we can access a variety of resources by creating handlers to the specific URIs. This allows us to combine disparate resources and compose them into a distributed application.

5 Future Work

The HPSearch system presented above is our initial approach towards building a scripting interface for management of system, streams and services. However with the increasing complexity of applications, the management of metadata associated with the system becomes increasingly complex and important [21]. We plan to study how the system and the metadata management would scale with the growing number of components. Further, services are inherently non-reliable. For systems to reliably deliver results, we plan to investigate how to incorporate new and alternate services into the application at runtime without affecting the performance and disrupting the flow of information. Again, this would require rigorous management of services and system metadata.

Security [22] is paramount to the functioning of any system. Although the shell and *WSProxy* do not currently implement any security schemes, we plan to investigate the security features provided by NaradaBrokering [23] and WS-Security [24].

One disadvantage of using default transport characteristics like TCP is that it is not optimal for every kind of application. For example, UDP might be more suitable for audio/video streams with reliability handled at the application layer rather than the transport layer. We are also currently working towards constructing the *StreamProxy* (discussed in section 3.4) to negotiate ideal transport for streams in Web Service architecture.

Finally we plan to add more handles to the HPSearch management console to interface different aspects of NaradaBrokering's substrate such as replay of events, security, discovery of brokers and services and heartbeat and performance monitoring for brokers and services [25] among others.

6 Conclusion

In this paper we presented HPSearch, a scripting interface towards management of the NaradaBrokering system and deployed services. We believe that the scripting interface

to NaradaBrokering would make it easier to setup and manage the NaradaBrokering system at runtime. Further we can quickly create data filtering applications and rapidly deploy them using NaradaBrokering as the messaging substrate.

Acknowledgement

This project is supported in part by the *Complexity Computational Environments: Data Assimilation SERVO Grid* project in the Advanced Information Systems Technology area of NASA's Earth Science Technology Program.

References

[1] Geoffrey Fox, Shrideep Pallickara and Savas Parastatidis, *"Towards Flexible Messaging for SOAP Based Services"* Proceedings of the IEEE/ACM Supercomputing Conference 2004, Pittsburgh, PA

[2] Shrideep Pallickara, Geoffrey Fox "NaradaBrokering: A Distributed Middleware *Framework and Architecture for enabling Durable Peer-To-Peer Grids"* Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware - 2003, Rio Janerio, Brazil June 2003

[3] NaradaBrokering, Project Page: http://www.naradabrokering.org

[4] *Web Services Reliable Messaging* http://www-106.ibm.com/developerworks/webservices/library/wsrm/

[5] HPSearch, Design and Development via Scripting, Project web-site: http://www.hpsearch.org

[6] *Rhino: JavaScript for Java*, Project Page: http://www.mozilla.org/rhino

[7] John K. Ousterhout. *Scripting: Higher Level Programming for the 21st Century* http://home.pacbell.net/ouster/scripting.html

[8] *Web Services Dynamic Discovery*, Available from http://xml.coverpages.org/WS-Discovery20040217.pdf

[9] *Universal Description, Discovery and Integration*, http://www.uddi.org

[10] The Apache AXIS project, http://ws.apache.org/axis

[11] WS – Transfer specification. Available from http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-transfer.pdf

[12] WS – Enumeration specification, Available from http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-enumeration.pdf

[13] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., and Nielsen, H. (2003), SOAP Version 1.2 Part 1: Messaging Framework. W3C Recommendation 24 June 2003. Available from http://www.w3c.org/TR/soap12-part1/

[14] WS – Secure Conversation Available from ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf

[15] Web Service Context Service Specification, Available from http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf

[16] Robert A. Granat, Regularized Deterministic Annealing EM for Hidden Markov Models, Doctoral Dissertation, University of California, Los Angeles, June, 2004

[17] IBM Sash, Project Website: http://sash.alphaworks.ibm.com

[18] OGSI:Lite and WSRF:Lite Project website: http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[19] GEODISE, Project page, http://www.geodise.org

[20] GLOBUS, http://www.globus.org

[21] Web Service Management, Available from http://www.intel.com/technology/manage/downloads/ws_management.pdf

[22] *A Security Architecture for Computational Grids*, I. Foster, C. Kesselman, G. Tsudik, S. Tuecke. Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998

[23] *Implementing a Prototype of the Security Framework for Distributed Brokering Systems* Yan Yan, et. al. Proceedings of the 2003 International Conference on Security and Management. Volume I pp 212-218

[24] Web Service Security Specification, Available from http://www-106.ibm.com/developerworks/webservices/library/ws-secure/

[25] *Implementing a Performance Forecasting System for Metacomputing: The Network Weather Service* Rich Wolski, Neil Spring, Chris Peterson, in Proceedings of SC97, November, 1997.