Management of Data Streams for a Real Time Flood Simulation

Harshawardhan Gadgil[†] Jin-Yong Choi[‡] Bernie Engel[‡] Geoffrey Fox[†]
Sunghoon Ko[†] Shrideep Pallickara[†]
Marlon Pierce[†]
[†]Community Grids Lab, Indiana University, IN, USA
501, N. Morton St. Suite 224, Bloomington, IN - 47404
hgadgil@cs.indiana.edu, {gcf, suko, spallick,marpierc}@indiana.edu
[‡]ABE Bldg, Purdue University, West Lafayette, IN - 47907
engelb@purdue.edu, jychoi@ecn.purdue.edu

Abstract

We examine the challenge of coupling applications to real-time data sources in a Grid environment. Such problems are important to the emerging field of Grid-based emergency planning and crisis response, in which real-time data sources must be coupled to modeling applications to provide timely forecasting to emergency planners and responders. In this paper we present a Grid-based scripting workflow system that is capable of managing both streaming data sources and service-based applications. We apply this system to flood modeling codes coupled to simulated NEXRAD Doppler Radar data.

Keywords: Grid Applications, Stream Data Processing, Workflow, Web Services, Grid Programming Models, Emergency Preparedness and Response, Geographic Information Systems

1. Introduction

Applications that generate rapid, continuous and large volumes of stream data include ATM and credit card operations, financial tickers, web server log records, readings from sensors used in a variety of applications, such as High Energy Physics experiments, Weather sensors and Network sensors. Most of this data is archived in a database at an off-site warehouse making accesses prohibitively expensive [1]. Ability to make decisions and infer interesting patterns in real-time, is crucial for several mission critical tasks.

1.1. Stream based vs. File based applications

Workflow systems such as BPEL [2] focus on flow of control, and data flow is a side-effect of control flow. This is true in case of business applications where the amount of data involved is very less as compared to a scientific environment. In execution of a workflow in a scientific environment we need to transfer large amounts of data between processes.

In traditional architecture, the entire data set is transferred using protocols such as GridFTP [3] before invoking the application for processing the data. During processing, the application may generate more data that must be stored until the computing is finished. This mandates huge amounts of space for temporary data. Some of the workflow systems illustrated in [4] focus on building workflows for grid but use traditional file transfer methods to transfer data between processes.

Processing data in a stream is advantageous since data processing can begin on the fly thereby alleviating the need for extra temporary storage for gathered data.

Stream data can be found in many applications. For example audio / video data applications [5] are stream based. Data provenance and consistent reproduction of analyzed data through workflow from stored raw data sources is another well known example [6] to which the streaming approach applies.

1.2. HPSearch

We have been developing HPSearch [7] as an extension to an existing scripting language that binds Uniform Resource Identifiers [8] (URI) to the scripting language. In future work we may also provide support for WS-Addressing [9] to refer to resources. This would allow

us to access URI's either as variables or through a search interface (For e.g. to Google web service).

Every data source, filter or sink is identified by a URI on the web. In our case we use a scripting environment to create objects that can be used to access a variety of data sources. This is known as *"Binding URI to the Scripting Language"*. We have the following URI bindings in HPSearch.

- Reading from (or writing to) files (using `file://`)
- Reading files via http or ftp protocols (`http://` or `ftp://`)
- Reading from (or writing to) a raw socket (`socket://ip:port`)
- Reading from (or writing to) a topic in a brokering system (`topic://`)
- Reading from a database and streaming the results of SQL queries as XML (`jdbc:`). We have implemented a simple scheme to map the result set to XML. (Refer, *http://www.hpsearch.org/notes/scripting*)

The HPSearch shell uses NaradaBrokering (Section 1.3) to route data streams. In effect, processing is done by passing data through various programs that are accessible as services, in a Pipe-Filter fashion.

This architecture is popular in UNIX systems. In this architecture every component (also called as a filter) has a set of input and output ports. The filter transforms or filters the data it recieves on its input ports. The processed data is sent out on the output port to the next filter. The pipe connecting the two filters is a directional stream of data usually represented by a data buffer to store all the data until the next filter has time to process it.

HPSearch is thus designed as a scripting interface [10] to the Internet (Grid). We currently use the Rhino [11] implementation of Javascript, although any other scripting language like Python or Perl can be used. Scripting poses numerous advantages as observed by [12]. Rhino further allows us to define custom Javascript objects called as *host-objects* that help to dynamically access the host system.

This feature can be useful to create objects that help manipulate data streams. In the prototype explained later, we make use of a proxy based Web Service that encapsulates the processing of stream-data. The overall application, which is made up of numerous such data filters is then controlled via HPSearch objects.

1.3. NaradaBrokering for Stream handling

NaradaBrokering [13, 14, 15] 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 may summarize some of the important features of NaradaBrokering as follows

- Implements high-performance protocols (message transit time less than 1 ms per broker)
- 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 [16]
- Fault tolerant data transport
- Support for different underlying transport implementations such as TCP, UDP, Multicast, SSL, RTP, HTTP.
- Discovery Service to find nearest brokers / resources.

2. Related Work

Systems that implement workflow support for combining high-performance computing elements exist; however, they have little support for managing data streams and data-flow between the connected components. We highlight three of these systems below.

Triana [17] is a graphical Problem Solving Environment that provides a user portal to enable the composition of scientific applications. Triana can be used as a Grid Computing Environment and can dynamically discover and choreograph distributed resources such as Web Services. Triana, however, lacks explicit support for control constructs and these constructs are handled by specific components.

Taverna [18] is an open source set of language and software tools to facilitate easy use of workflow and distributed compute technology within the eScience community. It features support for stateful grid services, nested / recursive workflows and implicit iteration support.

Kepler [19] (based on Ptolemy II [20]) is a set of Java packages supporting heterogenous, concurrent modelling, design and execution. The system supports activity-oriented programming and graphical user interface for composing complex workflows.

3. Grid technologies

Grid Computing technologies [21, 22, 23, 24] enable discovery of remote data sources and applications and carry out the transactions in a secured environment. Grid technologies are well established in high performance scientific computing and have more recently addressed problems in secure, large scale distributed data access, collaboration and information management [25, 26].

Various technologies are being used to implement traditional grids [27]. Grids are now defining *Web Service interfaces* [28, 29] that are independent of the underlying implementation, allowing us to dynamically discover services and connect them using various workflow technologies [2].

We have identified an important convergence in Grid, Web Service and Geographical Information Systems (GIS) technologies that will enable Grid based crisis management and emergency preparedness and response [30]. Crisis Grid proposes to connect data-sources, high performance modelling applications, computing resources, visualization services and collaboration services. High quality, high resolution data such as LIDAR (LIght Detection And Ranging) and online satellite data are available at unprecedented levels and the addition of small, networked sensor devices continue to drive the trend. With more and more data sources becoming accessible as Web Services, these services may be coupled with various core Grid Services to create a crisis response system for emergency preparedness, response and disaster informatics communities. The challenge lies in creating a unifying framework that connects these services in standard ways.

In general, the intended interaction with the services can be summarized as follows -

1. Discover and access remote data sources, filters and sinks, both archival and real-time.

2. Process Incoming data. This involves several steps including data filtering (to process the raw data and format it correctly for further processing) as well as computational modelling.

3. The final results may be delivered to visualization applications or published to interested subscribers including real-time collaboration and archival storage for Semantic Grid [31] technologies.

Figure 1 illustrates how a particular critical infrastructure can be constructed from core Grid services combined with special set of auxillary Grids (sensors, GIS, visualization, computing and collaboration).

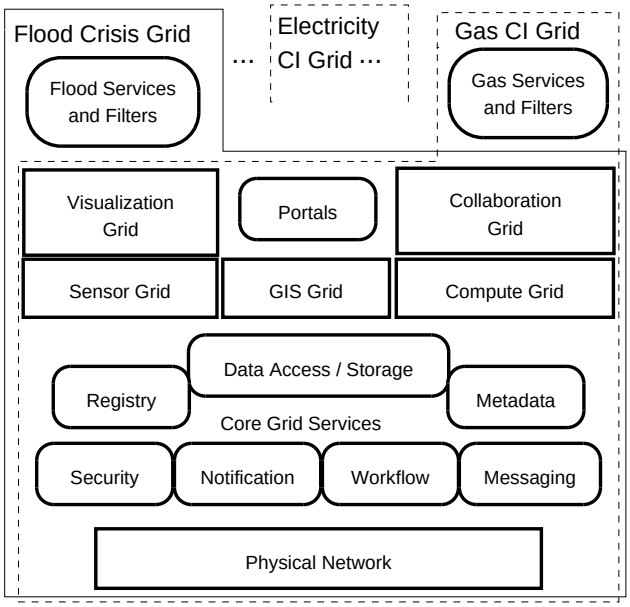

Figure 1. Critical Infrastructure Grids built in Composite fashion

3.1. Application to Flood Runoff computation

We consider the particular case of Grid-based flood modelling. Flood Crisis Grid will enable better flood prediction and assessment. The Crisis Grid approach can be adopted to many other applications including severe weather forecasting, earthquake modelling and wind-born contaminant modelling, to name a few.

The flood runoff computation (Refer figure 2) involves the following steps

- The `MapService` is an archival storage that sends out the spatial information for the terrain under consideration.
- The `RainFallPublisher` is a front-end to the sensor network that sends out the readings from the sensor to the `ComputeService`.
- The `ComputeService` uses the rainfall data and the spatial information to operate the runoff model for flood analysis.
- The results are then sent over to the `Visualization` service which displays the computed predicted model results in the form of an animation.
  - The visualization tool may be a standalone application or a servlet application, that subscribes to the topic representing the

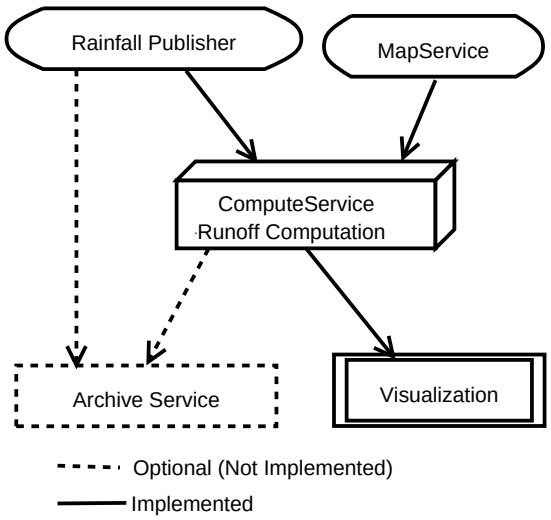

Figure 2. Grid based Flood Modelling

stream on which the `ComputeService` sends out the computed images.

- Alternatively, the computed data may also be sent over to the `Archive` service for future analysis.

4. Data flow

Workflow technologies [2] help us to link various Web Services together to create a much more powerful distributed application. In the flood grid computation we expect to link together disparate services (`MapService`, `RainfallPublisher`, `ComputeService` and `VisualizationService`) and orchestrate the functioning of these services. Further the data-handling requirement can be satisfied by using a Web Service based proxy that can be invoked in an existing Web Service fashion but can also handle data streams.

4.1. WSProxy Interface

The WSProxy is illustrated in Figure 3. The `WSSConnector` is a Web Service that wraps an `InvocationContainer` object. We use Apache AXIS [33] to deploy the Web Service. Using Web Service invocation, we can request the `InvocationContainer` to invoke a particular `ProxyWebService`. Additionally we can also specify the input and output URI corresponding to the required data sources and sinks respectively. The `InvocationContainer` is responsible for making the data sources / sinks available to the service as input / output streams. Thus, we may suggest stream topic names as input and output, effectively making the service process the data in a stream.

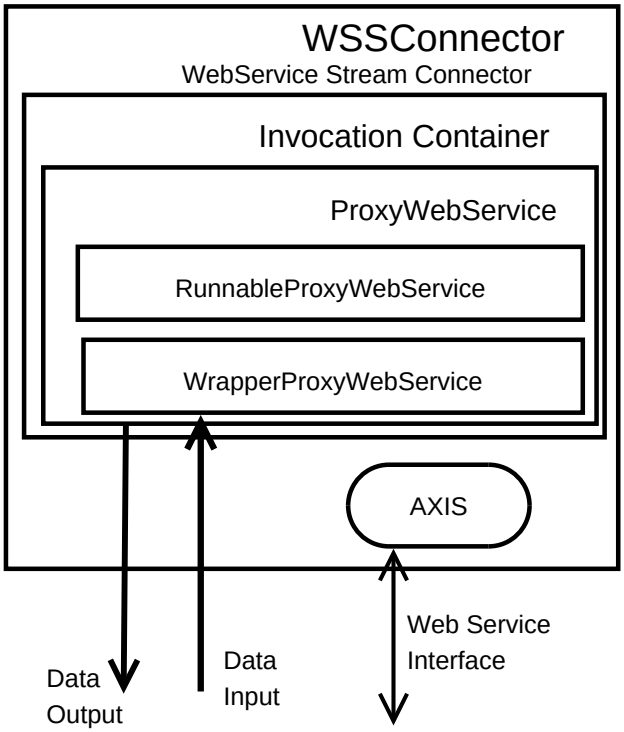

Figure 3. WS Proxy Architecture

As shown in the Figure 3, we also provide two interfaces namely `RunnableProxyWebService` and `WrapperProxyWebService`.

The functionality provided by them is as follows.

- **`RunnableProxyWebService:`**
  This presents a thread-safe interface with additional control operations, namely `start`, `suspend`, `resume` and `stop`. A service that wishes to operate on chunks of data may implement the `process` function. The thread associated with the process then repeats the following sequence

  1. Sleep for a *specified* timeout.
  2. Execute the `process` function. This executes the application defined process on a chunk of data.
  3. Check if the process was suspended or stopped. If not, goto step 1, else wait until the user issues a `resume`.

- **`WrapperProxyWebService:`**
  Using this interface is useful when the application contains the logic to start, stop, resume, suspend or handle the equivalent operations.

An advantage of this scheme is that the flow controlling engine need not handle the data streams, a desired feature noted in [32]. Furthermore this kind of a proxy can be invoked by any workflow engine and is thus workflow engine / language independent.

The `ProxyWebService` also exports the steering operations (*start, suspend, resume, stop*) via Web Service in order to provide steering capability to the data-flow application. Refer to Ref [34] for examples on the usage of WSProxy in the flood modelling prototype.

4.2. Scripting for controlling data flow

In our case we make use of the HPSearch engine to control setting up of the data flow. HPSearch can also be used to set up custom actions in the event of an error in processing. Using a scripting engine like HPSearch allows for *rapid prototyping* of the application. The HPSearch-Javascript is shown in Figure 4.

Setting up the service involves

- Discovering the service (`NBDiscover`)
- Creating a `WebServiceHandler` object to interact with the `WSSConnector` in setting up the service and starting it.
- (*Optionally*) Setting the various parameters of the service in question.
- Creating a `Flow` object to handle the distribution of various flow components to various workflow engines[1] in the system. Each of the workflow engines should be chosen such that it is nearest to the resources it has to accesse[2].
- Finally start the flow.

The various objects used, are as follows

- **`NBDiscover:`** Interfaces a custom *Discovery* service which serves as a registry of available services. This may be extended to discover brokers via the NaradaBrokering discovery service.
- **`WebServiceHandler:`** Serves as the client side to invoke and monitor the proxy webservice.
- **`Resource:`** Binding to handle URI's which point to file (`file://`), web resource (`http://` or `ftp://`), NaradaBrokering stream topic name (`topic://`) or a database (`jdbc:`).
- **`Flow:`** Handles the flow of data. We specify the components of the flow and the start activities (components that are responsible for inputting the data in the data flow pipeline). The components of the flow are initialized first followed by the start activities so that the components of the pipeline are ready to recieve and process the data.

[1] Refer: http://www.hpsearch.org/notes

[2] This is an optimization issue and has currently not been addressed

4.3. Error Handling

Since the components of applications (such as sensor networks) are very dynamic, we may have services and sensors going up and down. The nature of errors that may occur and the possible actions that might be taken vary between various data flow applications. We list below, some of the possible errors and the corresponding actions that might be taken

- The sensors participating to provide the rainfall data, fault or are washed away. The location of the new service is different. The user may check this by re-discovering the service, invoking it and resuming the computation.
- The `ComputeService` computation results in an incorrect result (infinity or NaN). The user may choose to ignore the readings for the current computation and carry on or take some other action. These kinds of errors are (`ComputeService`) implementation dependent and must be handled in application specific ways.
- The `MapService`'s archival storage throws a read error. Choices are
  - Ignore the error and try to read again.
  - If sufficient tries have been done, we may conclude that the `MapService` database is no longer available and either try to re-discover some other `MapService` or stop the computation altogether.

Thus from the above observations it is pretty clear that most of the errors are application specific and dynamic in nature and not much information is available until run-time. HPSearch shell provides a way to define action scripts corresponding to specific errors and notifications thrown by the WSProxy. Network Sensor Prototype [35] shows how these handler scripts may be defined.

5. Future work

The prototype works with static images from `MapService`. The `RainfallPublisher`

```
mapService = "org.hpsearch.demo.CrisisGridServices.MapService";
mapServiceLoc = "http://156.56.105.176:9090/axis/services/WSSConnector?wsdl";
mapSource = new WebServiceHandler(mapService);
mapSource.setEndPointURI(mapServiceLoc);

rainFallPublisher = "org.hpsearch.demo.CrisisGridServices.RainFallPublisher";
rainFallPublisherLoc = "http://156.56.104.176:6060/axis/services/WSSConnector?wsdl";
rainFall = new WebServiceHandler(rainFallPublisher);
rainFall.setEndPointURI(rainFallPublisherLoc);

computeService  = "org.hpsearch.demo.CrisisGridServices.ComputeService";
computeServiceLoc = "http:/156.56.104.176:7070/axis/services/WSSConnector?wsdl";
compute = new WebServiceHandler(computeService);
compute.setEndPointURI(computeServiceLoc);

archive = new Resource();
archive.port[0].subscribeFrom("topic://Visualization");
archive.port[0].publishTo("file://u/hgadgil/Visualization.dat");

crisisGridFlow = new Flow();
crisisGridFlow.addComponents(compute, archive);
crisisGridFlow.addStartActivities(mapSource, rainFall);
crisisGridFlow.start();
```

Alternate means of processing results. For illustration only.

Figure 4. HPSearch Javascript to create the flow

publishes random numbers as rainfall readings when the `ComputeService` specifically requests the next reading.

In an actual scenario we would be using the Purdue NEXRAD Doppler Radar [36]. Other rain guages [37] could be a part of an automated observing system like the *Automated Surface Observing System (ASOS)*[3] and the *Automated Weather Observation System (AWOS)*[4]. They generate atmospheric state variables of temperature, pressure, relative humidity and wind (speed and direction). The frequency of observation (and hence the data generation) ranges from 5 minutes for ASOS to 20-minutes for AWOS.

Other examples of rain guage / atmospheric state data include *Purdue Automated Agricultural Weather Station Network (PAAWS)*, a network of 9 stations at each of Purdue University's Agricultural Research Centers and *NOAA/NWS Cooperative Observer Program* which provides for 240 daily observations of precipitation in the state of Indiana as well as 15 minute observations at 77 sites in the state of Indiana.

Sensor and associated service failure, incorrect readings and other such (*runtime*) errors are entirely conceivable and must be handled without crashing the entire application. We discuss below some of these issues.

[3]Operated by National Weather Service (NWS), Federal Aviation Administration (FAA) and Department of Defense

[4]Operated by FAA and various state agencies

5.1. Assigning Attributes to Discovery and Service Invocations

It should be possible to invoke services with a different sets of parameters in order to run various simulations and check different use-case scenarios. All parameter settings could be stored in metadata management services by the researcher in order to keep track of simulations and results. For e.g.

- Suppose the flood modelling above relies on sensors that measure the rainfall and water levels in neighboring rivers and lakes. Suppose a researcher is only interested in the measured values for the state of Indiana, he may be able to set a parameter to get values for the state of Indiana.
- Another scenario is the NaradaBrokering discovery service. Suppose the `discover` operation returns more than one result for a particular service, the user should be able to query each service's metadata (service data) to find which service best suits his needs. Further the `NBDiscover` object should be augmented by the ability to do a template based search for services. The template could specify various capabilities of the service.

5.2. Service information

Services allow querying them to get specific service data or resource properties (WS-Resource Properties [29]). The

WSProxy implements a simple query scheme on getting the current status of the service (running, suspended, stopped). Other service specific service data querying would be included in future.

5.3. Error handling and Notifications

As shown in Section 4.3, HPSearch allows us to define custom actions using scripts. We also plan to implement a more sophisticated notification system to handle errors as well as notifications [38] and operations for rediscovery and subsequent handling of resources.

5.4. Security

Security [39] is paramount to the functioning of any system. Although the shell and WSProxy do not currently implement any security schemes, we plan to use the security features provided by NaradaBrokering [40]. We consider security in two parts.

- By implementing NaradaBrokering security features we can secure topics and the data sent on them. The WSProxy may be sent special security tokens / certificates which should be used when the WSProxy subscribes or publishes data on a particular topic.
- The invocation of WSProxy takes place as a normal Web Service. We use SOAP messaging for this purpose. This communication can be secured by using policies defined using WS-Security [41].

6. Conclusion

In this paper we describe how a complete Grid application can be built by linking data sources and filters. The paper also illustrates how a data-flow can be setup between disparate components by using NaradaBrokering and how data can be processed in a pipelined fashion. Although existing flow specifications help us link components, the WSProxy architecture helps us to control streaming data sources and manage the data-flow among the various components. By using HPSearch as the scripting engine we can now easily incorporate more data sources, services and end-user applications to create a more usable Grid based application.

References

[1] Alin Dobra et.al, *Processing Complex Aggregate Queries over Data Streams*, ACM SIGMOD 2002

[2] *Business Process Execution Language* http://www-106.ibm.com/developerworks/library/ws-bpel/

[3] GridFTP protocol, Project page: http://www.globus.org/datagrid/gridftp.html

[4] GGF-10 Workflow in Grid Systems, Berlin March 9, 2004 http://www.extreme.indiana.edu/groc/ggf10-ww/

[5] Ahmet Uyar, Wenjun Wu, Hasan Bulut, Geoffrey Fox *An Integrated Videoconferencing System for Heterogeneous Multimedia Collaboration* 7th IASTED International Conference on Internet and Multimedia Systems and Applications, IMSA 2003 August 13-15, 2003 Honolulu, Hawaii, USA

[6] Particle Physics Data Grid Project page: http://www.ppdg.net

[7] HPSearch, Project Page: http://www.hpsearch.org

[8] *Uniform Resource Identifiers*, IETF RFC, T. Berners-Lee et.al. Available at http://ftp.ics.uci.edu/pub/ietf/uri/rfc2396.txt

[9] *Web Services Addressing*, Specification available at http://www-106.ibm.com/developerworks/library/specification/ws-add/

[10] Geoffrey Fox, Harshawardhan Gadgil and Shrideep Pallickara *HPSearch and NaradaBrokering: Workflow Scripting and Stream Management*, Edinburgh, Dec 2003. Presentation available at http://grids.ucs.indiana.edu/ptliupages/publications/presentations/edinburghworkflowdec03.ppt

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

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

[13] Geoffrey Fox and Shrideep Pallickara ,*The Narada Event Brokering System: Overview and Extensions*, Proceedings of the 2002 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02). CSREA Press (2002) edited by H.R. Arabnia Volume I pages 353-359.

[14] 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

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

[16] Web Services Reliable Messaging http://www-106.ibm.com/developerworks/webservices/library/ws-rm/

[17] Triana Project, http://www.triana.co.uk/

[18] The Taverna project, Project page: http://taverna.sourceforge.net

[19] *Kepler: Towards a Grid-Enabled System for Scientific Workflows* , I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludscher, S. Mock, In the Workflow in Grid Systems Workshop in GGF10 - The Tenth Global Grid Forum, Berlin, Germany, March 2004.

[20] The Ptolemy project http://ptolemy.eecs.berkeley.edu

[21] I. Foster, C. Kesselman, and S. Tuecke, *"The Anatomy of the Grid: Enabling Scalable Virtual Organizations."* International J. Supercomputing App., 15(3), 2001.

[22] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke,*"The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Architecture"* Global Grid Forum Draft 2.9 (June 22, 2002), http://www.ggf.org/ogsi-wg/drafts/ogsa_draft2.9_2002-06-22.pdf.

[23] F. Berman, G. Fox, and T. Hey, eds. *Grid Computing: Making the Global Infrastructure a Reality* John Wiley and Sons, Ltd, Chichester (2003). Available from http://www.grid2002.org

[24] G. Fox and D. Walker, *e-Science Gap Analysis*. Prepared for the UK e-Science Programme, 2003. Available from http://grids.ucs.indiana.edu/ptliupages/publications/GapAnalysis30June03v2.pdf

[25] I. Foster and C. Kesselman, eds, *The Grid: Blueprint for a New Computing Infrastructure* Morgan Kaufmann Publishers, Inc, San Francisco. (1999).

[26] I. Foster and C. Kesselman, eds, *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kauffman, San Francisco (2003).

[27] NSF Middleware Initiative Website: http://www.nsf-middleware.org

[28] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, D. Snelling, and P. Vanderbilt, *Open Grid Service Infrastructure (OGSI)* Global Grid Forum Working Draft, 6/27/2003. Available from http://www.globus.org/research/papers /Final_OGSI_Specification_V1.0.pdf.

[29] *WS-Resource Framework* Refer http://www.globus.org/wsrf/

[30] Geoffrey Fox *Grids of Grids of Simple Services*, CISE Magazine July/August 2004

[31] The Semantic Grid. Project Page: http://www.semanticgrid.org/index.html

[32] Sriram Krishnan, Peter Wagstrom, Gregor von Laszewski *Grid Services Flow Language: A Workflow Framework for Grid Services* Available from http://www-unix.globus.org/cog/projects/workflow/

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

[34] Crisis Grid Prototype, Project page: http://www.hpsearch.org/demo/crisisGrid

[35] Network Sensor Prototype, Project page: http://www.hpsearch.org/demo/NetworkSensor

[36] Purdue NEXRAD Doppler radar http://roskilde.eas.purdue.edu/~level2

[37] Indiana Hydrometeorological Networks, http://www.ofps.ucar.edu/gapp/networks/indiana/indiana.html

[38] *Web Services Notification* Available from http://www-106.ibm.com/developerworks/library/specification/ws-notification/

[39] *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

[40] *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.

[41] *Web Services Security (WS-Security) Version 1.0* 05 April 2002, B. Atkinson, et al. Available from http://www-106.ibm.com/developerworks/webservices/library/ws-secure/.