Merging the CCA Component Model with the OGSI Framework

Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu,
Aleksander Slominski, Dennis Gannon, Randall Bramley

Department of Computer Science, 150 S Woodlawn Avenue.
Indiana University, Bloomington, IN 47405. Phone: (812) 855-8305.

Abstract

The most important recent development in Grid systems is the adoption of the Web services model as a basic architecture for Grid services. The result is called the Open Grid Services Architecture. This paper describes a component framework for building distributed Grid applications that is consistent with that model. The framework, called XCAT, is based on the U.S. Department of Energy Common Component Architecture but with an implementation based on the standard Web services stack. Using this framework, it is possible for an application programmer to build distributed applications by composing software components running on remote resources. The result is a transient, stateful Web service that represents the executing application instance. This paper describes the basic architecture of XCAT and its integration into the Open Grid Service Infrastructure framework.

Key Words: Computational Grids, Component Architectures, Web Services, OGSA, OGSI, CCA, Composition, Workflow

1 Introduction

A computational Grid [12] is a set of hardware and software resources that provide seamless, dependable and pervasive access to high-end computational capabilities. The Grid has the potential to provide programmers with the capability to explore a new generation of applications that can leverage teraflop computers and petabyte storage systems interconnected by gigabit networks. The success of the Grid will largely depend upon the development of tools and applications that can exploit its potential, and make it easy for the end user to use them.

A programming model for the Grid consists of tools, conventions, protocols, language constructs and a set of libraries that encapsulate a useful functionality. The abstractions provided by the programming model can simplify development of complex Grid applications. Until recently, there has been no consensus on what programming model is appropriate for the Grid. Examples of various models currently in use include MPI [31] for message passing, and GridRPC [26] for doing remote procedure calls.

The Open Grid Services Architecture (OGSA) [17] represents the first effort to standardize Grid functionality and produce a programming model for the Grid that is consistent with trends in the commercial sector. It presents an architecture that is based on the integration of Grid and Web services concepts and technologies. The Open Grid Services Infrastructure (OGSI) [15] refers to the basic infrastructure that OGSA is built on. At its core is the Grid Service Specification [32], which defines standard interfaces and behaviors of a Grid service, building on Web services technologies. OGSA and OGSI come close to defining a component architecture for the Grid.

A component architecture can be defined as a set of rules for specifying the behavior and interfaces of component instances and a framework that allows the component to be composed into applications. A component is a software object or process that satisfies the rules of the architecture. The software engineering benefits of component based software are well known: they enable encapsulation and facilitate in the modular construction of programs and the reuse of existing components, resulting in improved application productivity. Component architectures are well suited for rapid prototyping of complex distributed applications. These systems are of immense utility to scientists who want to build applications by composing existing software components which exploit specialized computing and algorithmic resources, and hold great promise to serve as an effective programming model for the Grid.

Various component models have been successful in industry, as well as in academia. The Microsoft COM [22] component frameworks have been fundamental to application interoperability Windows based applications. Now their Web services oriented .NET framework is also component based and is gaining widespread importance. In the CORBA world, the Object Management Group has released a specification for the Corba Component Model (CCM) [27], whereas Java Beans [24, 7] and EJB [23] have been popular component standards for Java based applications. The CCA [2] project, which is described here, is an initiative by DOE laboratories and universities to develop a common architecture for building large scale scientific applications from well tested software components that run on both parallel and distributed systems.

This paper presents three significant contributions to Grid research. The first is the merging of the CCA with the OGSI through our work with XCAT, which is our implementation of the CCA specification. A unification of the two models benefits both efforts, yet lets each group focus on their particular needs.

Perhaps the most important aspect of using a component Grid architecture is the way in which the application is constructed by composing components. Our second contribution is our classification of composition modalities based on their extents in space and time. We observe that distributed systems may be composed in two ways:

- Composition in space: one component/service directly invokes the services of another component through a logical connection between the two.
- Composition in time: a workflow engine schedules tasks that involve accessing remote services and responding to events.

We argue that both of these are important and they can both be accommodated in the XCAT framework.

Our third contribution is a messaging and notification system that extends the model proposed by OGSI. We also briefly describe how the OGSI factory service can be extended and used in applications.

2 A Brief Tour of Web Services

A Web service is an interface to application functionality that is accessible using well-known Internet standards and is independent of any operating system or programming language. Web services represent a shift in paradigm from a human-centric to an application-centric web.

Stack Layer	*Example Technologies*
Framework	.NET, Sun ONE
Discovery	UDDI
Description	WSDL, RDF
Messaging	SOAP
Transport	HTTP, SMTP, FTP, BEEP

Figure 1. Different layers of the Web service stack and the example technologies for each layer.

The various protocols composing a Web service are commonly divided into a five-layer stack as shown in Figure 1. This stack is evolving with various groups working on defining the standards.

1. **Transport**: The transport layer refers to the technology responsible for transfering messages between applications. The choices for this layer include HTTP, SMTP, FTP, and BEEP [19].
2. **Messaging**: This layer represents the marshaling and unmarshaling of application data so that it can moved over the network. Even though HTML has been widely used for the Web, it is not a suitable format for marshaling because it only describes the presentation of data, and not its semantics. XML, on the other hand, has gained widespread acceptance for representing data for Web services as it allows for a representation in accordance with the meaning of the data. SOAP is a protocol that uses XML as its data format and is the *de facto* standard for messaging in Web services.
3. **Description**: The description of a Web service includes the supported interface, network, transport and packaging protocols. The Web Service Description Language (WSDL) [1] is a widely accepted standard for this purpose. The Resource Description Framework (RDF) [35] specification can also be used, though it is less popular than WSDL.
4. **Discovery**: This layer serves as a registry that enables Web services to be published and discovered. The most widely recognized mechanism for this purpose is the Universal Description, Discovery, and Integration (UDDI) [33] specification.
5. **Framework**: The framework layer provides hooks to other Web Service layers so that applications can use them to build distributed systems. Examples of such frameworks include Microsoft's .NET and Sun Open Net Environment (ONE) [25].

In WSDL terms, a Service is a collection of *ports*. Each port is a named association between a *binding* and some form of network address. A binding is an association of a *portType* with a set of protocols and message formats. A portType defines a set of *operations*, which are defined by the operation name and the *types* of the input, output and fault *messages* that are associated with the operation. A WSDL description of a service is an XML document that defines the types, messages, portTypes, bindings and ports associated with a service.

3 The Open Grid Services Infrastructure

The Open Grid Service Infrastructure extends the Web service model by defining a special set of service properties and behaviors. First, it separates the service naming and service references. A Grid Service Reference (GSR) is a precise description of how to reach a service instance on the network. GSRs can be complete WSDL descriptions of a service instance. A Grid Service Handle (GSH) is an immutable name for a service. The idea is that a Grid Service Reference may change over time as a service is moved or upgraded. Hence a GSH may be bound to different GSRs over time, but the GSH can always be resolved to the official version of the service instance.

The most important contribution of OGSI is the means for specifying and restricting the behavior of a Grid Service by defining a family of standard ports. The most important of these is the Grid Service port. This port provides a type of dynamic service introspection that is a common feature of many component architectures. By invoking queries on the required Grid Service port, a client can discover information such as the other portTypes the service supports, the lifetime of the service instance, and other service-specific internal state data that the service wishes to expose. The information that is conveyed back to the client takes the form of XML fragments called Service Data Elements (SDEs). Each SDE is described by a Service Data Descriptor (SDD), which defines the schema and the content of the SDE.

Another important set of portTypes in OGSI involve notification. A client service can subscribe to changes in the service data of a source service by passing its GSH or GSR via the NotificationSource portType of the source service. The notification source pushes SDEs back to the subscriber when they have changed. This is accomplished by invoking a DeliverNotification operation on the subscribing service. This provides a basic form of service composition, but as we will argue below, it is not sufficient for a wide range of Grid applications.[1]

[1]The Global Grid Forum (GGF) OGSI working group is currently considering extensions to this notification model, so it may be changed by the time this article appears.

4 The Common Component Architecture

The primary emphasis of CCA has been on building applications and components for massively parallel supercomputers, but its semantics do not preclude its applicability to the Grid.

The central idea in CCA is to build applications by composition. Two CCA components are composed by connecting together their *ports*. *Provides ports* represent functionality a component provides to other components. Semantically, these are similar to simple RPC Web service ports. *Uses ports* represent functionality a component may need. Uses ports are essentially bindable references to provides ports. After a uses port is connected to a provides port, any functionality represented by the uses port is obtained by invoking the connected provides port.

The CCA can be compared to the CORBA Component Model (CCM). Like the CCA, the CCM also has the notion of ports. The CCA uses port is analogous to the CCM receptacle, and the CCA provides port is analogous to the CCM facet. Unlike the CCM, however, the CCA envisions connections as a dynamic, run-time activity. Ports can be added, removed, and connected at run-time, and this is considered normal behavior. The CCM does not allow the addition or removal of ports. CCM connections are considered part of application assembly, and not something the end user would usually do dynamically. While the CCA also supports connections used in this manner, the more flexible nature of CCA ports and connections allow it to be used to as part of Problem Solving Environments (PSEs), in which the end-user directly manipulates component connections to solve the particular problem at hand.

Each port is identified by name and is described by an interface of operations. The interface can be described by a Scientific Interface Definition Language (SIDL), or a simple Java interface, or by an XML specification such as WSDL. Figure 2 shows an example of a connection between two components with compatible port types.

Like OGSA, CCA provides a standard set of services and ports. The most important of these services are the component creation service, which allows one component instance to create an instance of another component, and the connection service, which allows the programmer to bind a uses port in one component to a provides port in another component. In the next section we will describe how these operations translate into Grid actions.

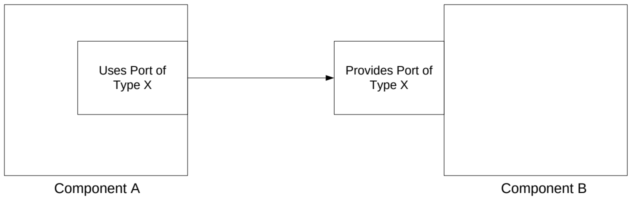

Figure 2. Example of a component connection using CCA. A uses port of type X can be connected to a provides port of the same type.

5 XCAT: A Web Service-Based CCA Implementation

In our previous work [3] we presented an implementation of the CCA specification. It was primarily built as a research vehicle to test the viability of the CCA specification for distributed computing. The system was built using HPC++ [14] and NexusRMI [4] as the underlying communication medium. The binary format of the communication substrate did not lend itself to exposing components as Web services. We redesigned and implemented the second version (now called XCAT 2.0) with SOAP [8] as the communication protocol. XCAT focuses on leveraging the advantages of both the component and Web services world. It implements all the layers of the Web service stack, and it has been implemented in both C++ and Java and provides seamless interoperability between components written in these two languages.

The port interfaces to XCAT components are described using XML documents conforming to a Schema. These documents are even used to generate the wrapper code that shields the users from the low-level details of the communication substrate used by XCAT. The generated code also handles the required conversion for seamless interoperability between C++ and Java based components. Every provides port in the XCAT implementation is a Web Service with one portType. The Web service is described by a schema that has a subset of the features in WSDL. We are currently in the process of moving to full fledged WSDL for this purpose.

XCAT uses the XSOAP [28] communication system for messaging, which provides an elegant model for communication between objects in different address spaces. XSOAP (formerly called SoapRMI) is an implementation of the Java RMI model in Java (XSOAP-Java) and C++ (XSOAP-C++) that uses SOAP as the communication protocol. XSOAP-Java uses the dynamic proxy feature, introduced in Java 1.3, to dynamically generate stubs and skeletons for every remote method invocation. Since C++ does not have introspection capabilities, XSOAP-C++ uses statically generated stubs and skeletons. We are currently working on porting the Proteus Multiprotocol Library [5] to XCAT. This will give us the option of using a multitude of communication libraries that include SOAP, JMS [30] and binary protocols.

XCAT provides a Creation service that allows a component to instantiate other components. This service completely encapsulates the component instantiation mechanism, thus shielding the component developers from the low-level, implementation-specific details of the instantiation mechanisms. This service allows:

1. Creating instances of components by accepting a set of environment values, such as executable location, host machine, and creation mechanism. A new component can be created in the same address space as the creating component or it may be instantiated in a different one on another host, in which case Globus GRAM (via the Java CoG Kit [34]), or ssh (if no queuing is desired, and globus is unavailable) can be used. Upon successful instantiation of the component, the creation service returns a ComponentID that serves as a handle for the new component.
2. Deleting instances of components by using their ComponentIDs.

XCAT also provides a Connection service which allows instantiated components to establish communication links with one another via their typed ports. By providing an external mechanism for connecting ports, the port types and descriptions themselves can remain free of any connection semantics. This service allows:

1. Connection and Disconnection between ports of components. Thus, the components can be dynamically composed at run-time.
2. Exporting the ports of another component as one's own. Thus, special wrapper components can expose selected functionality of other components. This can be used to present a simplified interface to the end-users, shielding them from the lower-level details.

Other features of XCAT that are important for its use as a distributed computing framework are enumerated below.

1. Security: Every remote method call is intercepted by the XCAT-Java framework before it invokes a method on the provides port. This design allows for a security service to be interposed between the provides port and the XCAT framework. This security service can

inspect the call and allow its passage if the security requirements have been met. The current version uses SSL with X.509 certificates and supports both authentication as well as a simple authorization model based on access control lists. This has been discussed in detail in [10].

2. ComponentID: The ComponentID represents a handle to the component that can be shipped to different locations. XCAT uses the remote reference mechanisms provided by XSOAP to represent a ComponentID. This handle can be *stringified* and stored in registries. It can then be retrieved by interested parties and used to invoke methods on the component. The ComponentID in this serialized form is represented as an XML document that describes the component. This XML document can be converted to a WSDL document using a tool provided by the XSOAP toolkit. This makes it possible to expose any provides port of a component as a Web service.
3. Exceptions: XCAT provides an exception model for communication between components. All exceptions thrown during the course of communication between components are caught and returned to the component that initiated the communication. The exceptions are mapped to *SOAP faults* on the wire and mapped to language specific exceptions before handing it to the initiating component.
4. Events and Notification: XCAT uses an event notification system called XMessages [29], which is a messaging middleware system designed for Grid applications that reliably delivers XML messages from publishers to subscribers even if a subscriber moves to a new location, or a publisher is restarted.

5.1 Composition in Space: Component Assembly

To use a component architecture effectively, it is important to be able to describe the various components that constitute an application along with the interconnections between them. In this model, which we can call composition in space, component instances are created on specified hosts and then connected together as a distributed system. It is also possible to create *metacomponents* which are themselves created by composing a number of components together.[2]

To accomplish this kind of composition, the XCAT Services APIs can be used directly by the user to write simple Java programs that can use remote component instances. The Java program can use the Creation Service to create components and obtain references to running instances. The program can then use the Connection Service interface to connect the provides and uses ports of these components. The Naming Service can be used to store and retrieve handles to running instances of components. It is also possible to invoke specific methods on the ports of various components.

The above method (of using Java control programs) is only suitable when we have a fixed set of components which are to be launched and monitored. A more dynamic mechanism to create and manage components on the fly, without the need for any recompilation, is desirable. We use Jython scripts for this purpose. Jython is a pure Java implementation of the Python language. Since XCAT has an implementation in Java, we can provide a Jython interface to the XCAT libraries.

Apart from the above two ways to orchestrate computations (Java control programs and Jython scripts), application-specific GUIs can be easily written and layered on top of the services provided by XCAT. We are also working on a workflow description for computations so that the end user does not need to write any Java or Jython code, but rather interact with the system, using a simple client, e.g. a Web browser.

6 XCAT and OGSA

Tranforming XCAT components into OGSA services is not difficult. A GridService provides port is added to each XCAT component. This provides the standard OGSI introspection. Service Data Elements containing references to each of the provides ports of a component are added to the GridService port, thus providing a list of services (provides ports) that can be accessed. As illustrated in Figure 3, we can combine component instances together using the XCAT composition to form composite Grid Services which may be accessed by any Grid Service client.

The ComponentID uniquely identifies the component, and can function as the GSH. As stated in Section 5, the ComponentID can be converted to WSDL, and eventually to GSDL (with the addition of Service Data Elements), which will function as the GSR. Component lifecyle in OGSI is handled by the `CreateService` and `Destroy` operations on the GridService port. Hence, the XCAT Creation service is modified to make these calls for creation and deletion respectively.

OGSI messaging, in its current form, is based on a very simple, point-to-point, non-reliable event *push* model. The

[2] Web services do not practically enable composition in space, because WSDL 1.1 does not completely define *outgoing* operations, i.e. where the Web service itself is the caller.

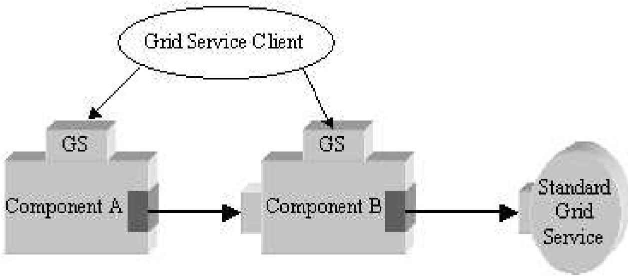

Figure 3. Each XCAT component has one provides port that implements the OGSI Grid Service Port Type. All other provides ports are first-class Web service ports as well as CCA ports. Components can be connected using CCA connection primitives and XCAT uses ports can be connected to conventional Web services.

only messages that are considered are changes in defined service data. The use of XMessages in XCAT provides a reliable, persistent network of message channels which store messages until they expire or they are explicitly removed from the system. A message can be an arbitrary XML fragment or it can be a more structured XML document event such as a OGSA service data element. In addition to having messages *pushed* to clients, a client can *pull* messages from the channel in batches. This allows the client more mobility and easier interoperation with firewalls. It is our feeling that OGSI will profit from having a more general *pub/sub* messaging layer.

OGSI has a Factory portType for instantiating services. Using XCAT, we have implemented an extended version of a distributed factory model for creating instances of applications [13]. Our generic application factory service takes a description of a connected network of components as input and creates an application coordinator component instance which, in turn, creates and links together instances of the described network of components. A reference to the new manager instance is returned to the client of the factory service.

Our current implementation fails to conform to OGSI in many small details. This is partly due to the fact that the standard is still very much a moving target. However, we consider the OGSA important and will continue to make improvements.

6.1 Composition in Time: Workflow

One of the most compelling reasons for the acceptance of the Web service technologies is their ability to combine existing processes and services into new ones that are more useful. *Workflow* can be defined as an organization of processes into a well-defined flow of operations, and can be thought of as the composition of services over time to accomplish a specific goal. This is one area that well surpasses the current OGSI specification.

While the composition of components in space defines how the components are logically connected at any point of time, workflow systems define ways in which flow of control and data can be expressed. As an example, an activity X in a workflow system may involve invoking operations P on service A, Q on service B, and R on service C (in that order), while service C may itself be composed of components D and E in space. Thus, these compositions are orthogonal, and can be applicable at the same time.

Currently, workflow systems for Grid and Web services is an area evoking a high degree of interest, with projects such as WSFL [18], BPEL4WS [6], and GSFL [21] investigating the various aspects of workflow in their respective domains. Since XCAT components are OGSI services (and hence, Web services), we can leverage work done in this field. We are currently looking into a *meta-composition* system that combines the composition of XCAT components in space and time.

7 XCAT Applications

The XCAT Science Portal [11] uses the XCAT implementation as the underlying model for launching distributed applications on the grid. Some projects that use the XCAT system are IU Xports [11], NCSA's Weather Research and Forecasting, (WRF) and Chemical Engineering [16], GRAPPA [20], Collision Risk Assessment (CRASS) [9], and Linear Systems Analysis (LSA) [9]. To illustrate the use of XCAT for composition of components in state and space, we show a typical scenario in Figure 4, which is seen in applications mentioned above, such as CRASS and Chemical Engineering.

As shown in the figure, the whole application is steered by an Application Coordinator, which is responsible for the composition of the other XCAT components. The Application Coordinator first launches a Master component and a set of Worker components, and makes appropriate connections between them. The Application Coordinator then subscribes to the Event Channel, in order to receive a *Simulation Complete* Event from the Master component. When

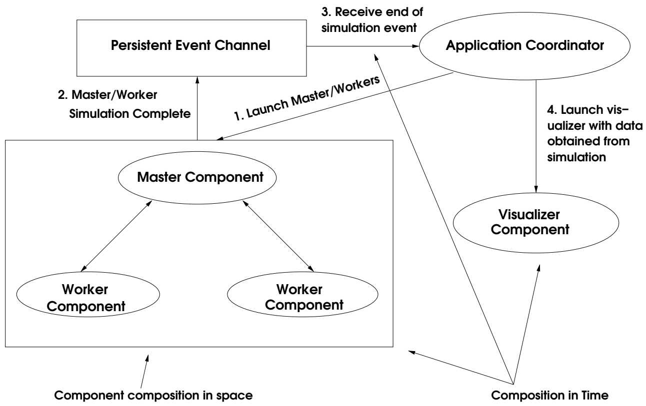

Figure 4. Component composition using XCAT

the simulation is complete, the Master sends the data from the simulation as an asynchronous event to the Event Channel. The Event Channel stores this data in persistent storage (for possible future use), and then relays it to the Application Coordinator. On receiving this event from the channel, the Coordinator spawns a local Visualizer, sending it the data just received. The Visualizer then visualizes the results from the simulation. Thus, XCAT enables composition in space (Master and Worker components), as well as in time (Master/Worker and Visualizer components), so as to allow orchestration of complex distributed applications.

8 Conclusions

We have presented a distributed software component architecture, XCAT, for Grid computing that is compatible with the Common Component Architecture specification (CCA) and also the Open Grid Service Infrastructure (OGSI) being proposed by the Global Grid Forum. CCA components have two types of *ports*. One type of port, which is called a provides port, is essentially identical to a Web service port. The other type, called a uses port, is an external reference from one component to a provides port on another component that can be bound at runtime. XCAT can also make use of an XML-based messaging system that provides a simple way for components to publish or subscribe to messages. This message system is substantially richer than the current OGSI notification scheme.

We have argued that the process of building distributed applications can be accomplished by either composing a collection of concurrently running components by linking their uses and provides ports (composition in space), or by scheduling the workflow between components and synchronizing activities based on the publication of application specific events (composition in time). There will be many cases when both schemes can be used together, and XCAT provides a mechanism to do so.

References

[1] Ariba, IBM, Microsoft. Web Services Description Language (WSDL) Version 1.1. http://www.w3.org/TR/wsdl.

[2] R. Armstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Toward a Common Component Architecture for High-Performance Scientific Computing. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computation*, August 1999.

[3] Randall Bramley, Kenneth Chiu, Shridhar Diwan, Dennis Gannon, Madhusudhan Govindaraju, Nirmal Mukhi, Benjamin Temko, and Madhuri Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of Ninth IEEE International Symposium on High Performance Distributed Computing Conference*, August 2000.

[4] Fabian Breg, Shridhar Diwan, Juan Villacis, Jayashree Balasubramanian, Esra Akman, and Dennis Gannon. Java RMI Performance and Object Model Interoperability: Experiments with Java/HPC++. *Concurrency and Experience*, 1998.

[5] Kenneth Chiu, Madhusudhan Govindaraju, and Dennis Gannon. The Proteus Multiprotocol Library. In *Proceedings of SuperComputing 2002*, November 2002.

[6] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.0, July 2002. http://www-106.ibm.com/developerworks/library/ws-bpel/.

[7] Robert Englander. *Developing Java Beans*. O'Reilly, 1997.

[8] D. Box et al. Simple Object Access Protocol 1.1. Technical report, W3C, 2000. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/.

[9] D. Gannon et al. Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. In *Special Issue on Grid Computing, Journal of Cluster Computing*, July 2002.

[10] Lavanya Ramakrishnan et al. An Authorization Framework for a Grid Based Common Component Architecture. In *Proceedings of the 3rd International Workshop on Grid Computing*, November 2002.

[11] S. Krishnan et al. The XCAT Science Portal. In *Proceedings of SuperComputing 2001*, November 2001.

[12] Ian Foster and Carl Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.

[13] D. Gannon, R. Ananthakrishnan, S. Krishnan, M. Govindaraju, L. Ramakrishnan, and A. Slominski. *Grid Computing: Making the Global Infrastructure a Reality*, chapter 9, Grid Web Services and Application Factories. Wiley, 2003.

[14] D. Gannon, P. Beckman, E. Johnson, and T. Green. *Compilation Issues on Distributed Memory Systems*, chapter 3 HPC++ and the HPC++Lib Toolkit. Springer-Verlag, 1997.

[15] Global Grid Forum. The Open Grid Services Infrastructure Working Group, 2002. http://www.gridforum.org/ogsi-wg/.

[16] Madhusudhan Govindaraju, Sriram Krishnan, Kenneth Chiu, Aleksander Slominski, Dennis Gannon, and Randall Bramley. XCAT 2.0: Design and Implementation of Component based Web Services. Technical report, Department of Computer Science, Indiana University, 2002. TR562.

[17] I. Foster, C. Kesselman, J. Nick, S. Tuecke. Grid Services for Distributed System Integration. *Computer 35(6)*, 2002.

[18] IBM. Web Services Flow Language (WSFL) Version 1.0. www-4.ibm.com/-software/solutions/webservices/pdf/WSFL.pdf.

[19] IETF. The Blocks Extensible Exchange Protocol Core. http://www.ietf.org/rfc/rfc3080.txt.

[20] Indiana University. Grid Access Portal for Physics Applications. http://iuatlas.physics.indiana.edu/grappa/.

[21] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. In *Argonne National Laboratory, Preprint ANL/MCS-P980-0802*, August 2002.

[22] Microsoft. COM. http://www.microsoft.com/com.

[23] Sun Microsystems. EJB. http://java.sun.com/products/ejb/index.html.

[24] SUN Microsystems. Java Beans. http://java.sun.com/beans/.

[25] SUN Microsystems. Sun Open Net Environment. http://wwws.sun.com/software/sunone/.

[26] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova. GridRPC : A Remote Procedure Call API for Grid Computing, July 2002. www.gridforum.org/Meetings/ggf5/pdf/APM11.pdf.

[27] OMG. Corba Component Model. http://www.omg.org/cgi-bin/doc?orbos/97-06-12.

[28] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. Design of an XML based Interoperable RMI System : SoapRMI C++/Java 1.1. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Pages 1661-1667*, June 25-28 2001.

[29] Aleksander Slominski, Yogesh Simmhan, Albert Rossi, Matthew Farrallee, and Dennis Gannon. XEvents/XMessages: Application Events and Messaging Framework for Grid, October 2002. http://www.extreme.indiana.edu/xgws/papers/xevents_xmessages_tr.

[30] Sun Microsystems. Java Messaging Service, 04-20-02. http://java.sun.com/products/jms/.

[31] The MPI Forum. MPI Documents. http://www.mpi-forum.org/docs/docs.html.

[32] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, and P. Vanderbilt. Grid Service Specification, October 2002. http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-04_2002-10-04.pdf.

[33] UDDI.org. Universal Description Discovery and Integration of Business for the Web, 04-12-02. http://www.uddi.org/specification.html.

[34] Gregor von Laszewski, Jarek Gawor, Sriram Krishnan, and Keith Jackson. *Grid Computing: Making the Global Infrastructure a Reality*, chapter 25, Commodity Grid Kits - Middleware for Building Grid Computing Environments. Wiley, 2003.

[35] W3C. Resource Description Framework, 06-01-02. http://www.w3.org/RDF/.