



OVERVIEW OF COMMON COMPONENT  
AND GRID SERVICES  
ARCHITECTURES  
WORKING DRAFT

WP5

---

Document Filename: **CG-5-TAT-ComponentsServices-001-FINAL**  
Work package: **WP5**  
Partner(s): **Cyfronet**  
Lead Partner: **Cyfronet**  
Config ID:  
Document classification: **CONFIDENTIAL**

---

Abstract:

This document presents a short description of the Common Component Architecture (CCA), Web Services and Grid Services and their possible application to CrossGrid.



### Delivery Slip

	Name	Partner	Date	Signature
<b>From</b>				
<b>Verified by</b>				
<b>Approved by</b>				

### Document Log

Version	Date	Summary of changes	Author
1-0-DRAFT-A	9/3/2002	Draft version	Maciej Malawski, Marian Bubak, Katarzyna Zając
1-0-DRAFT-B	15/3/2002	No major changes	Maciej Malawski, Marian Bubak, Katarzyna Zając
1-0-DRAFT-C	21/3/2002	Proofreading	Maciej Malawski, Marian Bubak, Katarzyna Zając, Piotr Nowakowski

---

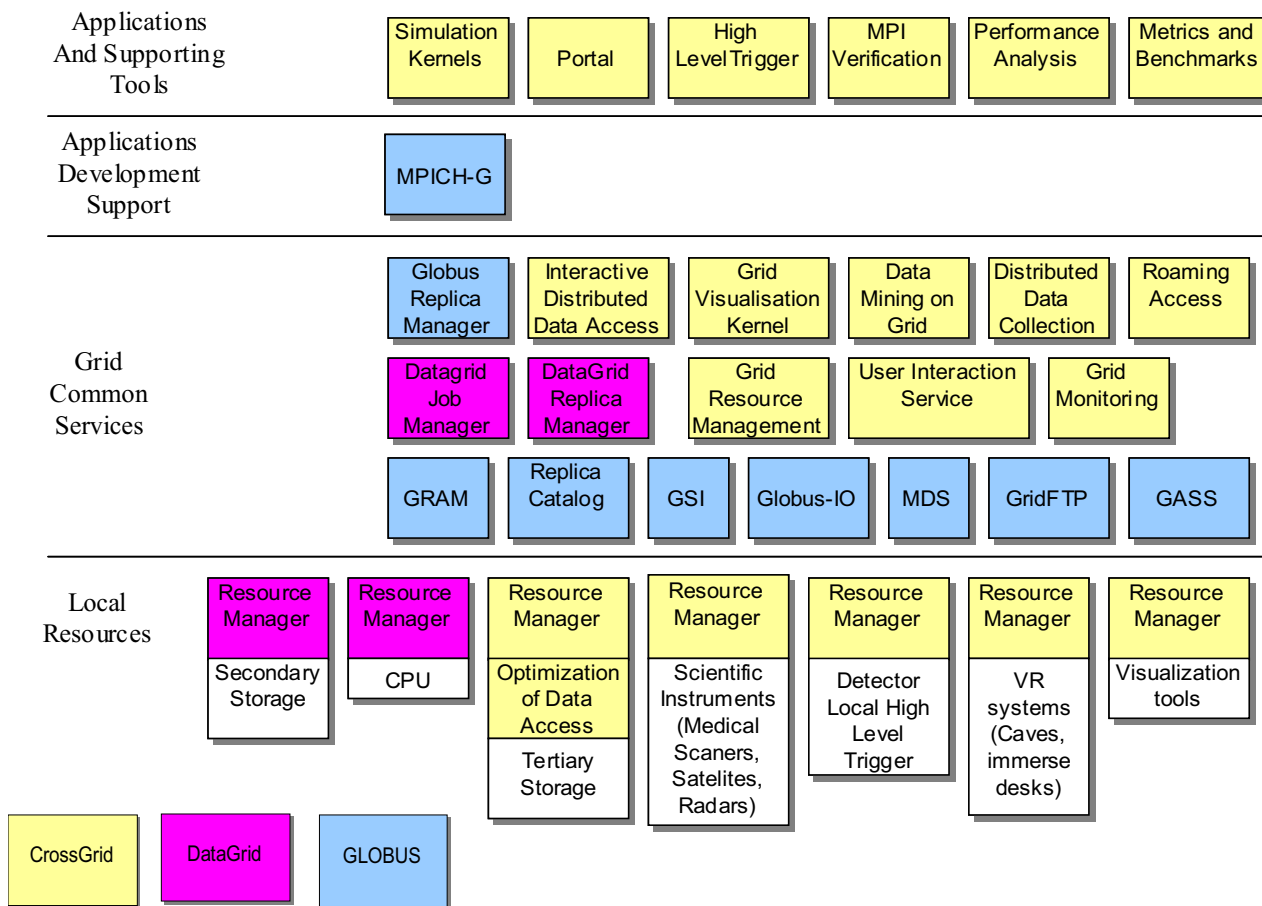
## CONTENTS

<b>1</b>	<b>INTRODUCTION.....</b>	<b>4</b>
<b>2</b>	<b>COMPONENT ARCHITECTURE.....</b>	<b>5</b>
2.1	INTRODUCTION .....	5
2.2	DEFINITIONS .....	5
2.3	CCA AND CCAT – EXAMPLES OF COMPONENT ARCHITECTURE .....	5
2.3.1	<i>CCA Component model</i> .....	5
2.3.2	<i>CCAT – the framework</i> .....	6
2.3.3	<i>Application builders</i> .....	7
2.3.4	<i>Application scenario</i> .....	7
2.4	ADVANTAGES OF COMPONENT APPROACH .....	8
2.5	PROBLEMS .....	8
2.6	PROJECTS RELATED TO CCA.....	8
<b>3</b>	<b>WEB SERVICES OVERVIEW.....</b>	<b>9</b>
3.1	INTRODUCTION .....	9
3.2	SOAP .....	9
3.3	WSDL .....	9
3.4	RELATIONS TO GRID COMPUTING .....	10
<b>4</b>	<b>GRID SERVICES.....</b>	<b>11</b>
4.1	BASIC IDEAS .....	11
4.2	OPEN GRID SERVICES ARCHITECTURE .....	11
4.3	GLOBUS 3.0?.....	12
4.4	APPLICATION TO CROSSGRID ARCHITECTURE .....	12
<b>5</b>	<b>REFERENCES.....</b>	<b>13</b>

## 1 INTRODUCTION

CrossGrid is a large project, in which many European partners intend to cooperate on software development. According to the Annex [ANNEX], CrossGrid Project will develop, implement and exploit new Grid components for interactive computing and data intensive applications, like simulation and visualization for surgical procedures, flooding crisis team decision support systems, distributed data analysis in high-energy physics and air pollution analysis combined with weather forecasting.

The most important components of CrossGrid software are shown in Fig. 1.



**Fig. 1 CrossGrid building blocks**

Besides being complex, these applications will run in a distributed Grid environment. The testbed sites will be distributed across participating European institutions and they will be used by diverse groups of application users, such as medical doctors, flooding teams or physicists. Each of those groups can be viewed as a distinct Virtual Organization (VO), as defined in [ Foster1].

This document is a short overview of technologies designed to aid in building large-scale distributed systems. First, two important approaches are discussed: the component programming model and Web Services technologies. These are followed by a presentation of the latest ideas developed by the Open Grid Services Architecture (OGSA) working groups.

---

## 2 COMPONENT ARCHITECTURE

### 2.1 INTRODUCTION

The component programming model was initially developed to help build commercial applications from compact blocks of software that can link with one another in simple ways i.e. using application builders. Such an approach was implemented in the Sun Java Beans (JB) [JBSpec] and Microsoft Component Object Model (COM) [COM] technologies.

A component architecture for scientific applications was proposed by the Common Component Architecture (CCA) [CCA] Forum. It was designed to support scientific applications operating in distributed environments.

### 2.2 DEFINITIONS

Three main forms of entities comprise the component architecture:

- Components
- Builders
- Frameworks

What are components? The Java Beans Specification [JBSpec] contains the following definition: “A Java Bean is a reusable software component that can be manipulated visually in a builder tool.” The CCA [CCA] definition is: “Components are the basic units of software that are composed together to form applications”.

From these definitions we see that it is necessary to have a tool to manipulate the components and compose application from them. This tool is an application builder which provides a graphical user interface to visually connect components together.

The third basic element is the framework. Its role is to create a runtime environment for components: to instantiate them, enable communication and provide additional basic services. The runtime environment for a component is often called a container.

### 2.3 CCA AND CCAT – EXAMPLES OF COMPONENT ARCHITECTURE

The most important features of component technology will be discussed basing on the Common Component Architecture and its reference implementation, the CCA Toolkit (CCAT).

#### 2.3.1 CCA Component model

In order to be CCA compliant, a component must define its interfaces to other components, called ports. There are two types of ports: the “provides” port and the “uses” port.

The “provides” port is an interface to methods that a component implements and that may be used by other components. It can be viewed as a “service” that the component “provides” to other components and to the framework (see Fig. 2).

The “uses” port is a point where external components’ “provides” ports can be connected. Internal code views it as an object which implements the functionality the component requires. It is the task of the framework to transfer calls invoked in the “uses” component to the code inside the component which “provides” the desired actions.

Port specifications are defined in SIDL (Scientific Interface Definition Language), which can be mapped to Java or C++ procedure calls. SIDL can be viewed as an extension to OMG IDL used for CORBA applications [OMG].

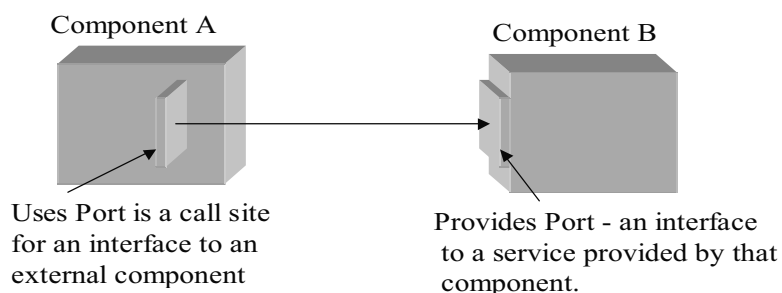


Fig. 2 CCA Ports (from [CCATppt])

### 2.3.2 CCAT – the framework

The CCAT framework was developed as a reference implementation of the CCA model. Its goal was to provide a runtime environment which would enable efficient execution of scientific applications on a massively parallel machine or in a distributed environment. Hence, the Globus toolkit was chosen as middleware, along with its Nexus communication library<sup>1</sup>.

The CCAT framework provides a set of standard services (see. Fig. 3)

- Directory Service - to locate components based on port type and other attributes
- Registry Service - to locate executing instances of components
- Creation Service - to create an executing instance of a component
- Connection Service - to connect the ports of two running component instances
- Event Service - a framework for publish/subscribe messaging between services and components.

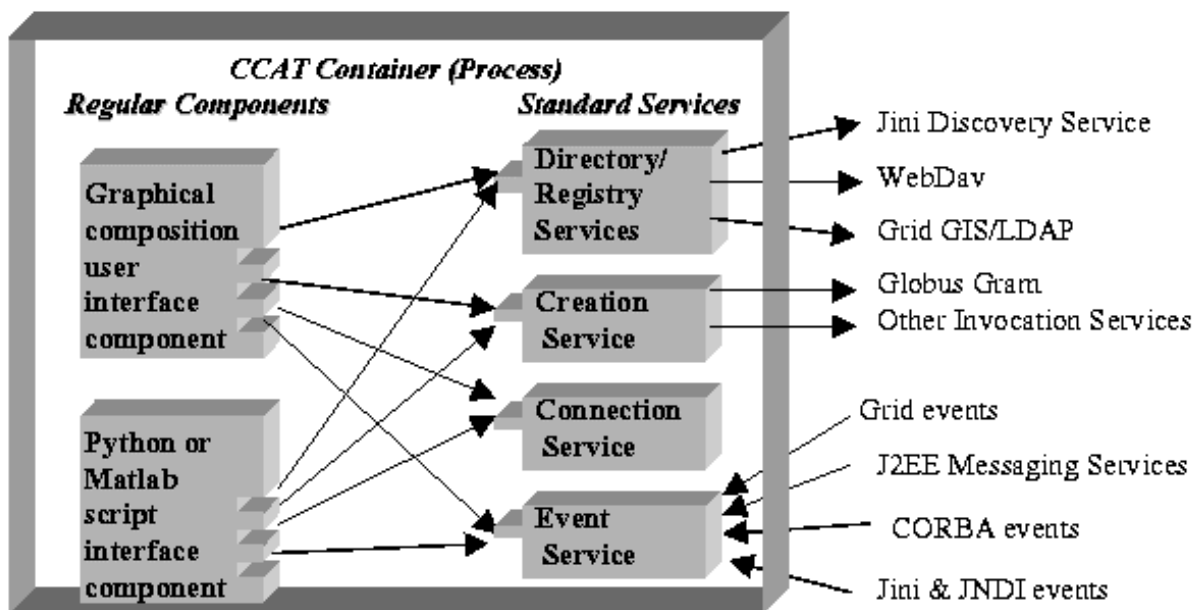


Fig. 3 CCAT Services (from [CCATppt])

<sup>1</sup> Nexus is no longer supported by the Globus project; Globus-IO is proposed instead.

### 2.3.3 Application builders

CCAT provides three ways of building applications from components: a GUI-based builder, a JPython script interface and a Matlab interface.

- The GUI builder is equipped with an interface which enables the user to browse information directories and another one, for registry browsing (see Fig. 4).
- The JPython interface makes it possible to access CCAT services and connect components within JPython scripts.
- The Matlab interface allows for initiating CCAT operations from the Matlab command line.

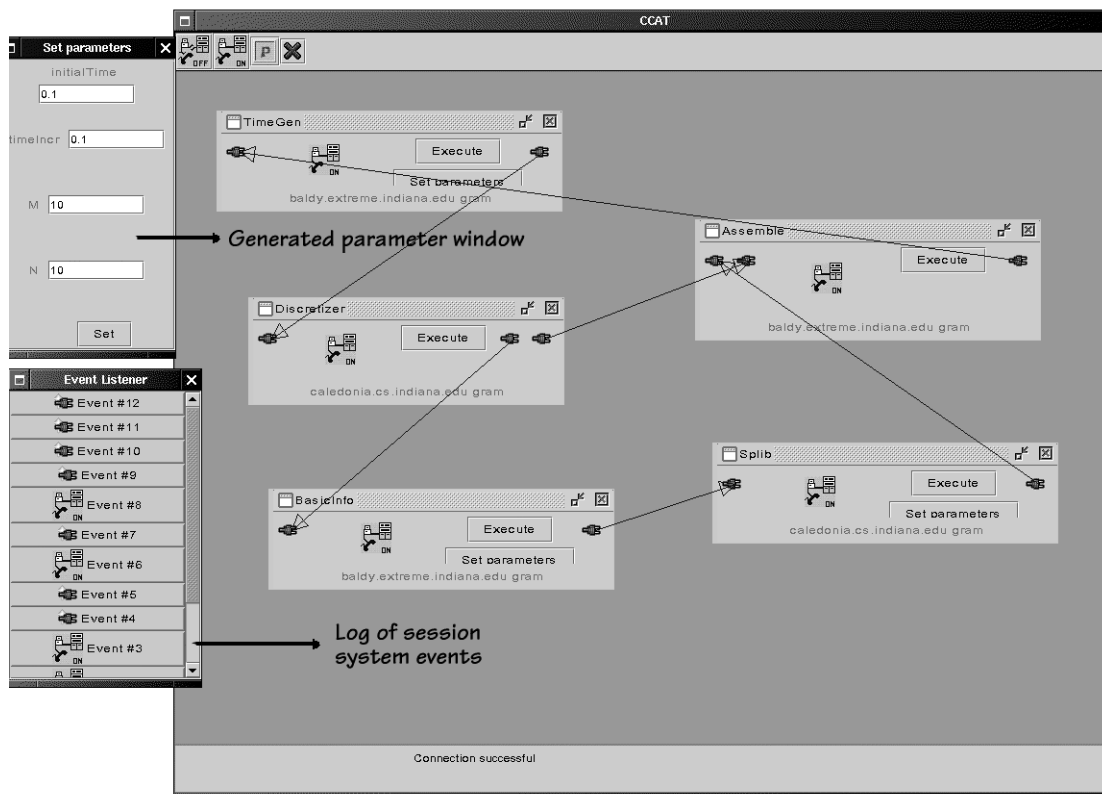


Fig. 4 CCAT Application builder (from [CCATppt])

### 2.3.4 Application scenario

To run a component-based application, a user needs

- a preconfigured framework,
- a set of components registered in a Directory Service (CCAT uses the Globus MDS),
- an application builder or a script to specify the necessary components and their mutual relations.

The user submits the description of an application to the framework, whose task is to instantiate the components (via GRAM) and connect the ports (in CCAT using Nexus). Afterwards, the components can perform their work and communicate directly with one another.

## 2.4 ADVANTAGES OF COMPONENT APPROACH

- Applications can be built from simple blocks,
- External components can be reused (i.e. it is possible to download/buy many Microsoft COM/ActiveX components from specialized developers – why not do the same for the scientific community, esp. within the CrossGrid framework?),
- In a distributed system, many remote components can be combined (depending on their availability).

## 2.5 PROBLEMS

- Heavy dependence on the framework and its constraints – component development requires a stable and reliable framework,
- Is the inter-component communication model efficient enough to meet the interactive CrossGrid applications requirements?

## 2.6 PROJECTS RELATED TO CCA

In addition to the CCAT implementation, there are also other projects related to CCA. First, the XML language (instead of SIDL) was used to specify component interfaces [CCAT2000]. Next, the XCAT project was issued by institutions involved in CCA/CCAT activity [XCAT]. Its objective is to create a science portal which makes it possible to execute applications from within a Web interface. It also uses the XSOAP technology as an RMI (Remote Method Invocation) protocol.

This activity, along with recent concepts proposed in [Gannon2001], indicates the common features of CCA and the latest achievements in Web Services technology. The following section outlines the key points of the Web Services approach and its possible extension towards Grid Services.

---

## 3 OVERVIEW OF WEB SERVICES

### 3.1 INTRODUCTION

The most recent papers by the authors involved in the CCA and Globus projects, [Gannon2001] and [OGSA] note the importance of Web Services as a framework for distributed systems.

The term *Web Services* (WS) describes an important emerging paradigm that focuses on simple, Internet-based standards (e.g., eXtensible Markup Language: XML) to address heterogeneous distributed computing. WS define methods for describing software components which can be accessed, methods for accessing these components, and discovery methods which enable the identification of relevant service providers. WS are programming language-, programming model-, and system software-independent. WS are based on W3C standards, such as SOAP [SOAP] and WSDL [WSDL].

In next subsections we will briefly discuss the W3 Consortium standards and find their connections to the component architecture.

### 3.2 SOAP

As described in W3C standard, SOAP is a lightweight information exchange protocol for a decentralized, distributed environment. It is an XML-based protocol, which consists of

- an envelope that defines a framework for describing the contents of a message and the means of processing it,
- a set of encoding rules for expressing instances of application-defined datatypes,
- a convention for representing remote procedure calls and responses.

SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in the specification describe how to use SOAP in combination with HTTP and the HTTP Extension Framework.

### 3.3 WSDL

The Web Services Definition Language (WSDL) is an XML format for describing network services as a set of endpoints operating on messages which contain either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of the message formats and network protocols used for communication ; however, the only bindings described in the specification describe how to use WSDL in conjunction with SOAP 1.1, HTTP GET/POST, and MIME.

A WSDL document consists of the following parts :

- **Types** – a container for data type definitions using a type system (such as XSD).
- **Message** – an abstract, typed definition of the data being communicated.
- **Operation** – an abstract description of an action supported by the service.
- **Port Type** – an abstract set of operations supported by one or more endpoints.
- **Binding** – a concrete protocol and data format specification for a particular port type.
- **Port** – a single endpoint defined as a combination of a binding and a network address.

- **Service** – a collection of related endpoints.

Web services also define the WSIL (Web Services Inspection Language), which enables publishing service descriptions, and the Universal Description, Discovery, and Integration (UDDI) registry. Such descriptions are accessible on the Web and enable discovery of advertised services.

### 3.4 RELATIONS TO GRID COMPUTING

Web services support a dynamic process of discovery and composition of services in heterogeneous environments through mechanisms for registering and discovering interface definitions and endpoint implementation descriptions. WSDL provides a standard mechanism for defining interface definitions separately from their embodiment within a particular binding (transport protocol and data encoding format).

Web Services are widely accepted by leading companies (Microsoft, IBM, Sun), which have produced many relevant developer tools.

We must note that Web Services, with their SOAP-based implementation, are better suited for applications that rely on exchange of lightweight messages rather than big data transfers. It makes them good for distributed parameter study problems and interaction in heterogeneous distributed environments.

The WSDL standard does not imply the use of SOAP or HTTP GET/POST mechanisms, but it still enables the usage of any other protocol for data exchange, if more stress is placed on the performance issue.

---

## 4 GRID SERVICES

### 4.1 BASIC IDEAS

The most recent papers by the authors involved in Grid and component technologies note the advantages of Web Services and propose to combine Grid computing and WS into a new paradigm, called Grid Services.

The paper [Gannon2001] proposes to merge CCA and WS, creating a Grid Factory Service (GFS) which would then create instances of applications on remote hosts and return some form of handles to the requestors. It should relieve the client of managing such details as environment variables, directories etc. that are required by Globus GRAM. The authors conclude that software component systems and Web services share many important characteristics and can interact as a foundation for building Grid applications. They suggest that WS applications can also be enhanced by adding Grid security protocols by layering SOAP on top of SSL using standard Globus and other X.509 certificates.

### 4.2 OPEN GRID SERVICES ARCHITECTURE

The draft papers “Physiology of Grid” [Foster2002] and “Grid Services Specification” [GSSSpec] propose a way of merging the Globus Toolkit and Web Services into one entity, called the Open Grid Services Architecture (OGSA). The main idea of OGSA is to view everything as a service. Computational resources, storage resources, networks, programs, databases, and the like are all represented as services. This enables uniform representation of all such entities and common methods to describe, register, access and manage these services.

In addition to persistent services, they focus mainly on transient services that can be created on demand. It implies the need of discovery, creation and lifetime management methods to make such transient services usable in distributed environments.

A Grid Service is defined as a Web service that provides a set of interfaces:

- Discovery (uses XML-based GSIE - Grid Services Information Elements)
- Dynamic service creation (Factory interface = GRAM Gatekeeper)
- Lifetime management (for keepalive, finalization and cleanup of transient services)
- Notification for event-based processing
- Manageability

The virtualisation of services makes them implementation-independent – this idea comes from WSDL. OGSA also provides standards for upgradeability (versioning) and naming conventions to manage evolution of services during their lifetime.

For discovery and registration purposes, OGSA defines two ways of service descriptors:

- GSH (Grid Service Handle) - a unique identifier of a service instance (no protocol-related information). GSH remains invariant during the lifetime of a service or when a service is restarted.
- GSR (Grid Service Reference) - provides a way to communicate with a service instance, including the protocol and version specification. GSR may change (expire) during the lifetime of a service.

A Handle Mapper Service maps between GSH and GSR.

To sum things up: within OGSA, everything is represented as a *Grid service* - namely, a (potentially transient) service that conforms to a set of conventions (expressed using WSDL) for such purposes as lifetime management, discovery of characteristics, notification, and so on.

### 4.3 GLOBUS 3.0 ?

What about implementation? The Globus team is currently working on a reference implementation of the OGSA architecture. Some results were presented at the Globus Tutorial, Chicago, Feb. 2002 (<http://www.globus.org/ogsa/deliverables/prototype.html>). The OGSA Architecture is going to be implemented in Globus Toolkit v3.0.

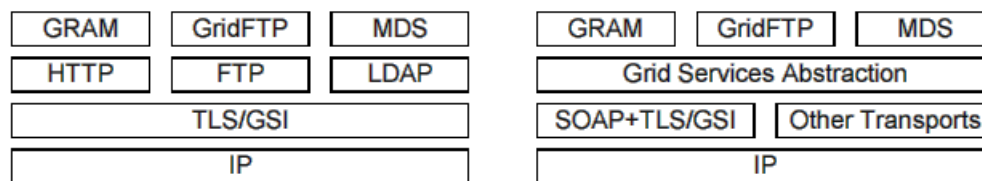


Fig. 5 Refactoring of Grid protocols according to OGSA mechanisms (from[Foster2002])

### 4.4 APPLICATION TO CROSS GRID ARCHITECTURE

OGSA presents a potentially interesting approach to X# architecture: we can describe blocks which comprise the CrossGrid (Fig. 1) as Grid Services that can be made available on testbed sites, registered in a Discovery service and instantiated when needed. Each virtual organization within the CrossGrid may use the CrossGrid portals to access a subset of available services, some of which may or may not be shared between distinct VO. The new Grid Services and Tools (WP3) and the Grid Application Programming Environment (WP2) can also be viewed as Grid Services.

We can observe that services are basically equivalent to components, but:

- Web services are increasingly popular in enterprise software.
- Standard protocols (SOAP, WSDL approved by W3C) and their implementations (Java 2 EE, Microsoft .NET, Apache SOAP, Jakarta Tomcat) are available for developers.
- Component systems, both commercial (Java Beans, Microsoft COM) and scientific (CCA) are evolving in the direction of Web Services.

At present, the following questions need answering:

- What should be the service granularity, i.e. do we want to build distributed simulation applications from low-level services or use other mechanisms for communication instead? Or, perhaps, we should incorporate other protocols into Grid Services for high performance? The other possibility is to wrap larger application modules into Grid Services and give developers a free hand in choosing the underlying architecture.
- Which implementation should we use? Although WSDL and OGSA mechanisms are designed to be implementation-independent and upgradeable, (making them promising for a 3-year project), we need something to start with and to deliver first prototypes. How much can we rely on Globus 3.0 as an OGSA implementation?

---

## 5 REFERENCES

- [JBSpec] Java Beans Specification <http://java.sun.com/products/javabeans/docs/beans.101.pdf>
- [CCAT] <http://www.extreme.indiana.edu/ccat/>
- [CCAT2000] R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In Proc. 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, Aug. 2000. <http://citeseer.nj.nec.com/bramley00component.html>
- [XCAT] XCAT Scientific Portal: <http://www.extreme.indiana.edu/xcat/index.html>
- [Gannon2001] Gannon, D., Bramley, R., Fox, G., Smallen, S., Rossi, A., Ananthakrisnan, R., Bertrand, F., Chiu, K., Farrellee, M., Govindaraju, M., Krishnan, S., Ramakrishnan, L., Simmhan, Y., Slominski, A., Ma, Y., Olariu, C. and Rey-Cenvaz, N., Programming the Grid: Distributed Software Components, P2P, and Grid Web Services for Scientific Applications. In *Grid 2001*, (2001)
- [WSDL] Christensen, E., Curbera, F., Meredith, G. and Weerawarana., S. Web ServicesDescription Language (WSDL) 1.1. W3C, Note 15, 2001, [www.w3.org/TR/wsdl.html](http://www.w3.org/TR/wsdl.html) .
- [SOAP] Simple Object Access Protocol. W3C Note <http://www.w3.org/TR/SOAP>
- [Foster2002] Physiology of the Grid – draft: [www.globus.org/research/papers/ogsa.pdf](http://www.globus.org/research/papers/ogsa.pdf)
- [GSSSpec] GSS draft: [www.globus.org/research/papers/gsspec.pdf](http://www.globus.org/research/papers/gsspec.pdf)
- [CCATppt] CCAT Presentation from: Ninth IEEE International Symposium on High Performance Distributed Computing Conference, Pittsburgh, August 1-4, 2000 [ftp://ftp.extreme.indiana.edu/pub/users/mgovinda/hpdc00D.ppt](http://ftp.extreme.indiana.edu/pub/users/mgovinda/hpdc00D.ppt)