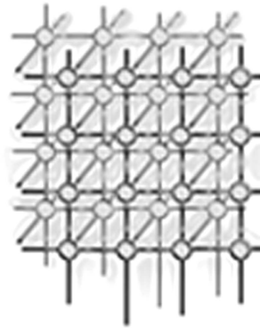
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2000; **00**:1–7 *Prepared using* cpeauth.cls *[Version: 2002/09/19 v2.02]*

Component-Oriented Application Construction for a Web Service Based Grid



Rainer Schmidt, Siegfried Benkner, Ivona Brandic and Gerhard Engelbrecht

Institute of Scientific Computing, University of Vienna
Nordbergstrasse 15/C/3, 1090 Vienna, Austria

SUMMARY

We present the architecture and prototype implementation of a component-oriented programming environment for a Web service based computational Grid. As middleware, we utilize the Vienna Grid Environment (VGE), a framework that enables the provision of compute-intensive parallel applications as configurable, QoS-aware Grid services. Our component model follows the Common Component Architecture and models application Web services as distributed components. We describe a component framework that integrates VGE services with a component model allowing to express and dynamically manage application and performance meta-data as well as dependencies on the infrastructure or other components. Furthermore, we show how the client programming interface is used to compose Grid applications from abstract application components that are mapped against available Grid services by the component framework at runtime.

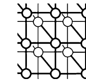
KEY WORDS: component-based programming, compute-intensive Grid applications, performance related meta-data, CCA, Web services

1. INTRODUCTION

The adoption of Web service technology for Grid computing environments has been a major research issue in recent years providing defined access mechanisms for distributed resources based on Web service standards like XML, SOAP, and WSDL. This development has been motivated by the Open Grid Service Architecture (OGSA) [13] and became apparent with the evolution of the Globus Toolkit [18] towards adopting Web service technology and the

*Correspondence to: Rainer Schmidt, Institute of Scientific Computing, University of Vienna, Nordbergstrasse 15/C/3, 1090 Vienna, Austria.
†E-mail: rainer@par.univie.ac.at

Copyright © 2000 John Wiley & Sons, Ltd.



recent implementation of the Web Service Resource Framework (WSRF) specifications [31]. The employment of a service-oriented architecture provides ways to cope with a changing environment as resources can be dynamically discovered by service consumers. A typical client programming pattern would therefore comprise finding a service in a registry (1), binding to a specific transport protocol (2), and invoking the service (3). As described in [12], service-based Grid infrastructures mostly comprise various collaborating services providing capabilities like security, information, data or resource management. In contrast to Web services typically deployed in areas like business-to-business commerce, it cannot be assumed that services in a Grid environment are always fully self-contained and agnostic of their surrounding infrastructure as they may have dependencies on other services resulting in an increased complexity for service configuration and access. Web services do not provide a notion for describing such dependencies as services appear self-contained to the user and only expose their provided interfaces.

Graphical problem solving environments [27] often utilize component abstractions that allow users to compose applications by connecting input and output parameters of the execution units. This abstraction appears more natural to the user rather than dealing with a service-oriented architecture. In order to reduce complexity it is furthermore desirable to compose applications based only on components that are relevant to the problem the user addresses - thus, to provide the user with computational components (e.g. a solver) while hiding infrastructure related entities (e.g. a service to access authorization lists). We therefore target a plug-in environment that allows developers to construct applications by connecting ports of computational components while the component framework handles connections that result from context dependencies automatically.

We present a prototype implementation of a component environment that is based on the Common Component Architecture (CCA) [4]. The system models HPC application Web services as distributed components and provides a component integration framework realized as Web service. The framework integrates application Web services with the component model and exposes a builder interface for component-based Grid application construction. Components may dynamically register provided interfaces and dependencies as well as proxy implementations and meta-data descriptions with the component framework. Application developers may compose Grid applications from abstract application components that are mapped against the available Grid resources by the component framework at runtime.

As Grid middleware our system utilizes the Vienna Grid Environment (VGE) [6], a Grid infrastructure for the provision of HPC applications as Grid Services over standard Web service technology. VGE services provide generic interfaces for remote job execution, monitoring, error recovery as well as for application level QoS support. The VGE service provision framework has been utilized in the GEMSS Project [17] for the Grid provision of time-critical medical simulation services [7] incorporating various compute intensive methods such as Finite-element Modeling and Monte Carlo simulation. Most of these applications consist of various steps of execution for example mesh generation, data analysis, or visualization, which can be deployed separately using VGE. Key aspects of VGE include QoS support for time-critical service provision, support for business models as well as end-to-end security.

The structure of this paper is as follows: We give a brief overview of the Common Component Architecture in section 2; The VGE service provision framework and QoS support mechanisms

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

are explained in section 3; The overall system architecture is presented in section 4; Section 5 deals with the Grid programming model and exemplifies the client API; Related work is presented in section 6; Finally, conclusions and future work are presented in section 7.

2. THE COMMON COMPONENT ARCHITECTURE

Component-based software development based on commodity frameworks such as JavaBeans, EJBs, COM, DCOM, or CORBA is a broadly accepted practice for building business applications. The Common Component Architecture (CCA) specification developed by the CCA Forum [9] provides a component model designed for applications construction in high-performance computing. The fundamental problem addressed is the dynamic composition of applications based on encapsulated pieces of heterogeneous 3rd party legacy codes. CCA is a lightweight, interface-based specification defined in Scientific Interface Description Language (SIDL), which provides support for scientific data types (e.g. dynamic multi-dimensional arrays, complex numbers) and language interoperability.

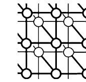
The CCA component programming model is (similar to the CORBA Component Model) based on connecting components by means of provided ports (*provides ports*) and accepted ports (*uses ports*), expressing dependencies. Components are not viable by themselves and therefore have to be integrated with a framework that controls the component's life-cycle and interactions. A CCA component framework exposes (among others) a defined set of operations for dynamic registration and retrieval of component ports. Components may register their provided interfaces as well as dependencies and associated properties, which are stored in a repository. Each component must implement an additional *component interface* required to interact with the component framework over a callback mechanism. As the CCA specification does not define implementation details, frameworks for massively parallel hardware [1] as well as for distributed Grid environments [15] are being developed.

Application construction is established via the *builder service*, a component-based application programming interface (API), providing operations for the instantiation of component types, component and port introspection, as well as component interconnection. During application execution the individual components may be in different phases and loaded dynamically. The *builder service* furthermore provides operations for application decomposition and component destruction. For more information on the Common Component Architecture the reader is referred to [9].

3. THE VIENNA GRID ENVIRONMENT

The Vienna Grid environment is a service-oriented Grid infrastructure for the on-demand provision of HPC applications as Grid services and for the construction of client-side applications that access Grid services. The VGE service provision framework is based on a generic application service model and automates the provision of HPC applications as services based on standard Web service technology such as SOAP, WSDL, WS-Security. VGE utilizes the open-source frameworks Tomcat and Axis [3, 2] for service hosting and deployment. As a

Copyright © 2000 John Wiley & Sons, Ltd.



key feature, VGE supports a flexible QoS negotiation model where clients may dynamically negotiate QoS guarantees on execution time and price with potential service providers. For the construction of advanced client-side applications a high-level application programming interface (API) that hides the complexity of accessing Grid services is provided.

VGE has been utilized and evaluated in the context of the EU Project GEMSS [17], which developed Grid middleware and a testbed for medical simulation applications. Within GEMSS, six medical Grid service prototypes have been realized including the applications maxillo-facial surgery simulation, neuro-surgery support, radio-surgery planning, inhaled drug-delivery simulation, cardio-vascular simulation and advanced image reconstruction. These applications rely on compute intensive methods such as Finite-Element Modeling, Monte Carlo Simulation, and Computational Fluid Dynamics and consist of various steps of execution (mesh generation, data analysis, visualization), which can be deployed separately as VGE Grid services.

3.1. Generic Application Services

VGE services encapsulate native HPC applications, usually parallel MPI codes running on a cluster, and expose their functionality via a set of common operations for job execution, job monitoring, data staging and error recovery. In addition, VGE services may be configured in order to offer QoS guarantees with respect to execution time or price. VGE services are compiled from a set of services during deployment and exposed as a single application service by means of a composite WSDL document.

- The **file handling service** provides operations for uploading and downloading input/output data based on files. Support for direct data transfer between services is provided by corresponding *push* and *pull* operations. VGE uses file transfer via SOAP attachments for data exchange instead of communicating via XML encoded documents.
- The **job execution services** provides operations for launching and managing remote jobs by interfacing with a *compute resource manager*.
- The **QoS negotiation service** enables clients to dynamically negotiate various QoS guarantees such as execution time and price on a case-by-case basis. Resulting QoS contracts between service provider and client are formulated as Web Service Level Agreements [30] (WSLA) and go along with advance resource reservations.
- The **monitoring service** generates structured data regarding the application status and information gathered by individual monitoring scripts.
- The **error recovery service** provides support for checkpointing, restart, and migration, if supported by the application.

3.2. Service Provision Framework

The VGE service provision framework automates the task of transforming HPC applications into Grid services. VGE services are configured by descriptors, based on XML schemes, regarding the underlying application, security, and the exposed functionality. The framework provides tools that assist the user in generating the XML descriptors and packaging and

Copyright © 2000 John Wiley & Sons, Ltd.

```
<application>
        <description>
                <version>...
        <configuration>
                <working-directory>
                        <path>/home/...</path>
                </working-directory>
                <input-files>
                        <file><name>TiH2.in0</name><format>...
                        ...
                <output-files> ...
                <job-script> ...
        </configuration>
        <qos>
                <request-parameters>
                        <param>mesh-size</param><param>#iterations</param>...
                <machine-parameters>
                        <number-of-nodes><param>1</param><param>4</param>...
                ...
                <provider-parameters>
                        <compute-resource-manager-class>at.ac.univie.iss...
                        <performance-model-class>at.ac.univie.iss.apm...
        ...
</application>
```

Figure 1. Example Application Descriptor

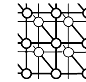
deploying the application services. The *application descriptor* is an XML document that provides meta-data about an existing HPC application and the computational resource it is installed on. It constitutes the application specific part of the service description, which is mapped against the generic methods by the service provision framework.

Figure 1 shows a simplified excerpt of an application descriptor providing structured data about the underlying resource. It specifies a working directory that is used for storing transient job execution data (e.g. i/o data, status files) within generated session directories. Furthermore, information on input/output files (name, data-format), a script for initiating job execution, and scripts for gathering status information have to be provided. The `compute-resource-manager` element may be used to specify an interface to a job scheduler. Currently NEC's COSY [10] and the MAUI [26] scheduler are being utilized. To enable QoS support, a set of request and machine parameters (e.g. number of nodes) as well as a performance model have to be provided. The performance model is parameterized with the request parameters(mesh size, number of iterations), which have to be provided by the user for QoS negotiation. The application descriptor contains application meta information also relevant to service composition like QoS properties, input/output files and data formats - and therefore is incorporated with the component system described in the following sections.

3.3. Dynamic Application Configuration and QoS Support

VGE services may by configured to offer QoS guarantees on execution time or price based on a *QoS module* and meta-data dynamically supplied by the user. The *QoS module* comprises a *QoS manager*, a *compute resource manager*, a *performance model* and optionally a *Pricing*

Copyright © 2000 John Wiley & Sons, Ltd.



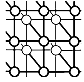
model. For obtaining a *Qos offer* the client may specify a *QoS descriptor* containing QoS constraints (e.g. max. execution time) and a *request descriptor* containing meta-data describing a job request. For a FEM simulation a request descriptor would typically contain the size of the finite-element model, the number of iterations to be performed etc.

In order to provide guarantees on application execution time, the VGE *QoS module* requires an application-specific *performance model* and a *compute resource manager* supporting advance reservation. As input the *performance model* takes the *request descriptor* and *a machine descriptor* specified by the service provider at deployment time. The *machine descriptor* describes the computational resources (e.g. number of processors), which may be provided for an application. This meta-data is fed into the *performance model* to obtain an estimate for the required execution time. It interacts with the *compute resource manager* to check whether a reservation of the required resources can be made within the required time frame. If these requirements are met, a temporary resource reservation (with short lifetime) is made and a corresponding QoS offer in form of a Web Service Level Agreement [30] (WSLA) document is issued to the client. Only when the client, which usually negotiates with multiple service providers, confirms an offer, a signed QoS contract in the form of an WSLA is established. Temporary reservations for offers that are not confirmed by the client expire within a short time frame. When a client has successfully negotiated the required QoS with a service provider and a QoS contract is in place, the usual job execution phase can be entered, this comprises the invocation of service operations for uploading the input data, for starting the job execution and for downloading the result. In order to support direct data transfer between services, corresponding `push` and `pull` operations are supported as well. The generic QoS module of VGE enables the provision of parallel applications as dynamically configurable Grid services. Depending on the requirements of a client, an application may be executed on many processors in short time but for a higher price, or it may be executed on a few processors with a lower price. The VGE QoS mechanisms provide means for qualitative service description, which may be used for automatic resource selection.

4. A SERVICE BASED COMPONENT FRAMEWORK

The basic idea behind this work is to embed distributed Web service resources into a component model for creation, introspection, and composition. In general, Web services are highly self-contained as they only expose technical interface descriptions for accessible ports but do not provide means to specify dependencies on the hosting environment or on other services. This characteristic has the advantage that standalone applications can be easily made available as services but causes drawbacks regarding deployment and composition. A simple application composition scenario and possible resulting dependencies is shown in Figure 2. We distinguish between two classes of dependencies: *Explicit dependencies* that result from connections drawn directly by the application developer (e.g. a connection between reconstruction and visualization service) and *implicit dependencies* that are caused by the Grid infrastructure (e.g. an application service dynamically requesting authorization lists from a security service). We therefore target a component-based programming environment that allows the user to

Copyright © 2000 John Wiley & Sons, Ltd.



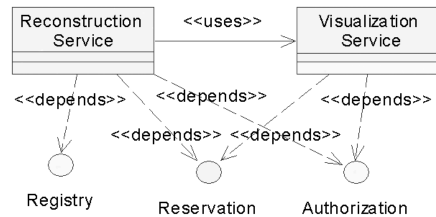


Figure 2. Explicit and Implicit Service Dependencies

construct Grid application by interconnecting application services based on ports while *implicit dependencies* are handled transparently by a component framework.

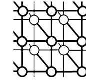
Our environment provides VGE services to client programmers based on the CCA component model and therefore uses mechanisms like *provides/uses ports*, *component interface*, *framework services* and *builder service*. Hence, Grid services are modeled as components and Web service invocations as connections among them. For integrating VGE application services with the component model we implemented three additional software packages: A distributed component framework that mediates between the service-oriented architecture and the component programming model, a service library package providing interoperability between the application services and the remote component framework, and a client programming API for component based application construction.

Component-Based Composition with VGE services

VGE services basically allow to remotely execute native parallel applications preinstalled on computational resources via a set of generic Web service interfaces. As scientific experiments usually comprise multiple steps of execution VGE services may be composed into a larger application assembly. Workflows are controlled by the client based on RPC calls e.g. for status inquiry, as VGE does not provide a notification mechanism. VGE services can be connected directly (in a peer-to-peer fashion) for data transfer using the client API. By integrating the CCA connection model a VGE service may be remotely connected to arbitrary Web services that have complementary CCA ports. Such connections may be constructed for example with security services, registries, or remote data bases for logging and checkpointing. Connections may be explicitly triggered by the client applications via a method call (e.g. data push) if they are time/state dependent or they may be internally used by the component (e.g. retrieving access lists). Component introspection allows a user to identify specific files which can be downloaded (e.g. intermediary results) or being transferred using a file connection. Figure 3 shows a typical setup we use for parameter studies. Using applications such as ant colony optimization [5] or quantum dynamics simulation [8], a client (or a driver component) may concurrently run tests with different input data sets on distributed hosts, whereas a visualization takes place after a successful simulation has finished. The file connections between

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls



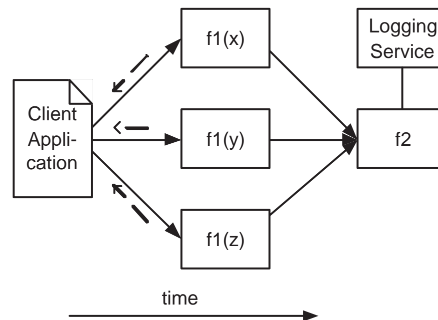


Figure 3. Composed VGE services: Concurrent parameter study (f1) and visualization (f2)

client, simulation (*f1*), and visualization (*f2*) are pushed by the user depending on the job status and intermediary results. The connection between visualization (*f2*) and logging service is implicitly used by the component on demand.

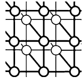
Dynamic Component Registration and Selection

Figure 4a shows the overall system architecture comprising a client application (cmp1,cmp2), the remote component framework, and a set of services (s1, s2, s3). The component framework is implemented as Web service and acts as broker between application demands and available Grid services. It provides the CCA *services interface* allowing a Grid service to register provided/required ports with the framework as well as request them dynamically at execution time (a). We extend the VGE Web services with a *component interface* required to interact with the framework. Furthermore, we provide an application builder interface used by the client environment to construct and execute a component based application. The client application may consist of several components that can be described, instantiated, connected and invoked using the *builder service*. The framework locates the services on behalf of the client application at execution time (b) based on component descriptors. If a service is selected, the framework delivers information required for component introspection (e.g. supported ports, wrapped application, QoS attributes) and component interaction (component handle, proxy) (d) back to the requestor (c).

Dynamic Proxy Handling

Our component framework maintains a proxy repository used to provide the client environment with high flexibility concerning component specific libraries. The idea is to enable client applications to stay compliant with different versions of VGE services, dynamically retrieve proxy updates, and also be able to incorporate non-VGE services. We therefore extended the Java classloader mechanism with a dynamic proxy pattern that allows the client developer

Copyright © 2000 John Wiley & Sons, Ltd.



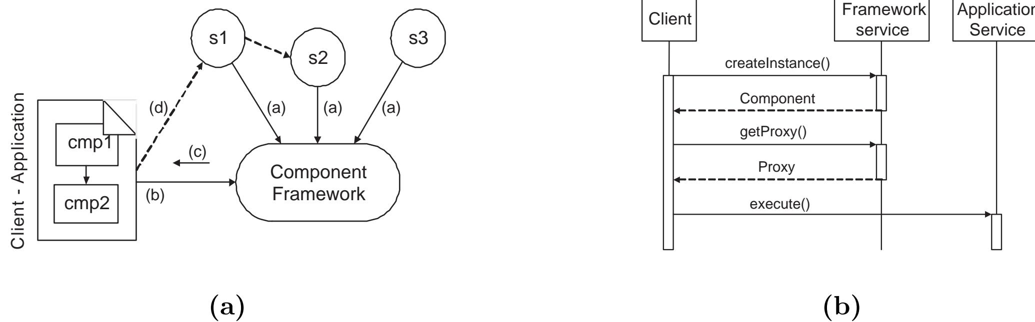


Figure 4. (a) General System Architecture (b) Dynamic Proxy Mechanism

to program against a port interface and dynamically access remote libraries via a framework service at runtime (Figure 4b).

If a port is registered with the component framework it may be associated with a proxy implementation that can be uploaded to the framework. At execution time the client environment contacts the framework services in order to find a component and fill the client-side representations with substantial runtime information (e.g. component handle, supported ports, proxy class name, class loader). Invoking a method on a component port causes the runtime engine to generate a call from the client to a service using the local code base. If the invocation fails or the component is unknown to client-side code repository, the Java classloader dynamically loads a proxy file from the framework where the component is registered. The mechanism of interface based service discovery and proxy lookup, which we have applied to Web services, is well known from Jini [29] based systems.

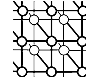
5. COMPONENT-BASED GRID PROGRAMMING MODEL

The client programming environment is currently provided as a Java API and based on the CCA builder service interface. It offers a component-based programming model that allows the client-side construction of composite Grid applications based on composable entities. The available resources are virtualized behind the component abstractions and are selected by the component framework during application execution. This section describes elementary concepts and operations provided by the client environment.

5.1. Application Programming Concepts

Describing the Component: As resources offered via Grid Services (especially if they have generic interfaces) cannot be specified by class or interface names uniquely, it is required to describe them also by their characteristics. For our environment, VGE

Copyright © 2000 John Wiley & Sons, Ltd.



components can be described by associated meta-data such as name and version of the underlying application, input/output data format, and, if supported, by QoS attributes like execution time. To also allow the integration of non-VGE services into the environment (e.g. a database adaptor) it is further required to specify the interfaces a component has to provide. Such a programming model provides the advantage that the described resources can be virtualized by the framework and selected at runtime on behalf of the client.

Component Creation: The `createInstance()` method generates a component object based on a component descriptor. The client application therefore uses the *builder service* to retrieves a handle to a service and optionally proxy code from the component framework. Invoking this method further causes the creation of a client session at the computational resource by a VGE service. These components are being identified by a `ComponentId` consisting of the service handle and a session identifier.

Component Connection: *Web Service Components* may implicitly request other components or can be connected explicitly by the client application using the `connect` operation. In order to enable remote connections, a component has to expose a builder interface allowing to inject a handle to a called component. Connections between VGE components can be seen similar to pipes within a Pipe-And-Filter architecture. After an application has processed and indicates that it is finished the output files are piped to the target service by calling a `push()` operation. The component object provides attributes that allow the client to introspect the component in order to obtain information specified within the corresponding application descriptor. These properties may be used to distinguish between files and attach multiple connections to a component.

Invoking Provides Ports: Components provide direct access to their *provides ports* via a generic `getProxy` operation. The component therefore generates a proxy for a *provides port* of a component that is represented as an object of the requested interface data type. This port abstraction allows to invoke an interface provided by a remote service and may automatically contact the component framework for a proxy implementation if it is unknown to the client.

Decomposing the Application: The operation `destroyInstance()` destroys the component within the client application and invalidates the session at the target resource. Destroying the client session at the computing resource will automatically delete all permanently stored files and resource reservations.

5.2. Application Composition and Execution

The application programming interface should, in general, not be directly provided to application developers but serve as interface for higher-level development environments. End-users may compose applications by utilizing a scripting interface or a visual composition environment. A major design goal for the API is the formulation of abstractions and mechanisms that allow to describe a composite Grid application based on application

Copyright © 2000 John Wiley & Sons, Ltd.

components, ports, and connections. Hence, the client application constitutes an abstract program description that has to be mapped against applicable Grid resources at runtime. Furthermore, it has to be considered that most composite applications may not run autonomously but require user interaction like status inquiries and component introspection on files, formats, etc. Overall, we assume that HPC application provision is based on the VGE middleware, a common Grid security infrastructure is in place, and that application meta-data descriptions follow a common schema.

A code fragment for component description and instantiation using the client API is shown in `Listing 1`. The component is described based on a set of ports it has to provide (job handling, QoS support) using a generated *interface descriptor* and a set of attributes assigned to predefined constants. The *interface descriptor* allows e.g. to distinguish between VGE services providing different sets of ports (section 3.1) but in general may be used to specify arbitrary Web services (e.g. a DB adaptor). The value "SPECT" refers to the name of the application exposed by the VGE service, which is specified in the application descriptor. QoS support in VGE provides a valuable feature for qualitative service description. Components that provide a *QoS negotiation* port may therefore be additionally described by QoS attributes. The method `createInstance` performs a component selection based on the component description and - if supported - a QoS negotiation. Component creation furthermore goes along with the creation of a corresponding client session and optionally a resource reservation at the service.

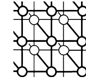
```
//Listing 1: component description and creation
portDsc.addProvidesPort(vge.JobHandlingInterface.class);
portDsc.addProvidesPort(vge.QoSModuleInterface.class);
compDsc.addPortDesc(portDsc);
compDsc.addProperty(CConst.APP, "SPECT");
compDsc.addProperty(CConst.QoSDsc, qosDsc);
compDsc.addProperty(CConst.ReqDsc, reqDsc);
Component comp = builder.createInstance(def);
```

Component ports have to be interoperable technically as well as semantically in order establish a working connection among them. For technical interoperability the framework provides support for dynamic proxy loading using the component repository described in section 4. In order to enhance interoperability, the programming model provides introspection capabilities allowing to retrieve meta-data on the applications and input/output data (e.g. application, version, files, formats) specified within the *application descriptor*.

In `Listing 2`, two VGE components are connected by file handling ports. The components are therefore introspected concerning their supported *provides* and *uses* ports. A connection can be established using the `connect` operation and a complementary pair of ports. A data connection furthermore requires to be provided with the specific files to transfer. The application developer therefore may introspect the components on available input/output files as well as request additional file information such as type and format. VGE file connections are triggered explicitly by the user by invoking a corresponding *push* operation.

```
//Listing 2: file transfer connection
port uport1 = comp1.getUserPort(fileHandlingPortDsc);
```

Copyright © 2000 John Wiley & Sons, Ltd.



```
port pport2 = comp2.getProvidesPorts(fileHandlingPortDsc);
Connection con1 = builder.connect(comp1, uport1, comp2, pport2);
con1.setProperty(CConst.TDATA, comp1.getProperty(CConst.OUTFILE)[0]);
builder.push(con1);
```

The composite Grid applications, we are targeting, usually comprise various steps of execution accomplished by loosely-coupled applications services. These programs require explicit interaction by the user to inquire status and intermediate results, or to access custom application ports. For these reasons we offer mechanisms to directly access provided component ports by a client through a proxy object. In `Listing 3` proxy objects for VGE job steering and monitoring ports are requested and used for a direct method invocation.

```
//Listing 3: direct access to component ports
vge.JobHandlingInterface job = (...) ppJob.getProxy();
vge.MonitoringInterface monitor = (...) ppMon.getProxy();
job.start();
while(monitor.getJobStatus() != State.FINISHED) {
// ...
}
```

6. RELATED WORK

In this section we give a brief overview on some related projects addressing distributed, component-based scientific computing.

The GrADS [22] project is developing an architecture and software environment (GRADSoft) for distributed and heterogeneous computing of scientific applications within a computational Grid. It provides a program preparation and closed-loop execution system supporting technologies like compiler analysis, resource negotiation, runtime performance analysis and optimization, and continuous monitoring. The system targets to provides a continuous process of adapting an application to a specific problem instance and a changing environment in order to maintain the overall performance. Applications are therefore being encapsulated as *configurable object programs* and provide performance contracts stating the expected performance of the application modules. The runtime environment configures the object code based on the available resources and may interrupt and reconfigure the application during execution. The VGrADS [28] project extends GrADS in terms of usability and introduces the abstraction of *Virtual Grids*. Vgrids are described by a resource specification presented by an application and submitted to a *Virtual Grid Execution System* for dynamic resource selection.

The ICENI [24] framework provides scientific computing based on performance aware components over distributed resources. The system maintains component meta-data concerning component meaning, behavior, implementation, and performance characteristics. Through separation of concerns it can maintain multiple implementations for a given abstraction. The scheduling architecture provides performance prediction, reservation and workflow aware scheduling, which allows ICENI to select optimized combinations of component

Copyright © 2000 John Wiley & Sons, Ltd.

implementations. ICENI is based on a service-oriented architecture and has been implemented on top of Jini, JXTA, and OGSI.

Triana [27] is a distributed problem-solving environment that allows to compose applications from a set of components using Grid as well as peer-to-peer computing. It provides a graphical composition interface and writers/readers for choreography languages like BPEL4WS and Triana XML. Triana is middleware independent through the use of the Grid Application Toolkit (GAT) [16]. GAT has been developed within the GridLab project and provides a high-level Grid API with bindings for various middleware implementations (Web services, Globus, JXTA).

XCAT3 [25] is a distributed, CCA implementation written in Java that allows component-based Grid application construction based on CCA and OGSI compliant Grid services. For remote component instantiation XCAT supports Globus GRAM [18] via the Java CoG kit [19] as well as ssh, and uses XSOAP [32] as communication protocol. An XCAT C++ [21] implementation that uses the Proteus multi-protocol library as communication substrate is currently under development [11]. LegionCCA [20] is a distributed CCA implementation that maps distributed Legion objects as CCA components by linking them against a specific library.

Both, XCAT3 and the work presented in this paper are utilizing Web services technology for hosting and connecting distributed components based on CCA. The systems target related problem domains but differ in many aspects of design and their application, which is briefly discussed in the following paragraph:

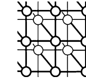
XCAT3 basically implements a CCA component as a set of Grid services that represent the component and its associated *provides ports*, stubs are dynamically registered with the framework and represent the *uses ports*. XCAT3 allows custom components being written and created locally or remotely at runtime. A client application e.g. in form of a Jhyton script may furthermore use the builder service API to remotely connect corresponding component ports, execute component assemblies, or directly invoke ports. The XCAT3 implementation has been utilized with several applications and scenarios [14] [33].

The system described in this paper maps CCA concepts to a Grid based on VGE application Web services. We model an application service as component, the exposed Web service ports as *provides ports*, and dependencies on other Web services as *uses ports*. VGE services are being deployed once and may then be used by multiple clients simultaneously as they are multi-threaded and session based. We therefore provide a globally accessible CCA framework service that serves as registry and broker for deployed components, ports, and proxies. A major difference to XCAT is that VGE application services are typically connected via a generic file transfer mechanism. Our client API provides a builder interface that allows remotely connecting VGE services with each other as well as connecting them to ancillary Web services. Data transfers between VGE services are activated by the client, other component connections may be implicitly used and/or requested by the components.

7. CONCLUSIONS

We presented ongoing work on a programming environment that allows to construct complex Grid applications from distributed, compute-intensive application components. Our system

Copyright © 2000 John Wiley & Sons, Ltd.



is based on the Vienna Grid Environment, a Web service based Grid environment that enables the provision of parallel applications as QoS aware Grid services. We presented a prototype implementation of a component framework that is based on the Common Component Architecture, and able to integrate Grid services with a component model for managing context dependencies and dynamic resource selection. Application construction is based on a programming model that allows to specify components based on meta-data description and performance-related QoS constraints. As the system is still in an early stage, we plan to make further improvements by experimenting with scenarios from various scientific domains.

REFERENCES

1. Benjamin A. Allan, Robert C. Armstrong, Alicia P. Wolfe, Jaideep Ray, David E. Bernholdt and James A. Kohl. *"The CCA Core Specification in a Distributed Memory SPMD Framework."* In *Concurrency : Practice and Experience,* 14(5):323-345, 2002.
2. Apache Axis. http://ws.apache.org/axis/.
3. Apache Tomcat. http://jakarta.apache.org/tomcat/.
4. 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"*, Proceedings of the High-Performance Distributed Computing Conference, August 1999, pp. 115-124.
5. S. Benkner, K.F. Doerner, R.F. Hartl, G. Kiechle, M. Lucka. *"Cooperative Ant Colony Optimization on Clusters and Grids"*, Proceedings International Workshop on State-Of-The-Art In Scientific Computing, PARA 04, Lyngby, Denmark, June 2004.
6. S. Benkner, I. Brandic, G. Engelbrecht, R. Schmidt. *"VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing"*, Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004), Pittsburgh, PA, USA, November 2004.
7. S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S.E. Middleton, R. Schmidt. *"GEMSS: Grid Infrastructure for Medical Service Provision"*, Journal of Methods of Information in Medicine, Vol. 44, 2005.
8. J. Caillat, J. Zanghellini, and A. Scrinzi. *"Parallelization of the MCTDHF code"*, Aurora Technical Reports, 04-19 (2004), available at http://www.vcpc.univie.ac.at/aurora/publications/
9. The Common Component Architecture Forum. http://www.cca-forum.org/ [August 2005].
10. The NEC COSY Job Scheduling System. http://www.ccrl-nece.de/ falk/COSY/cosy.shtml.
11. Deger Cenk Erdil, Kenneth Chiu, Madhusudhan Govindaraju, and Michael J. Lewis. *"A Proteus-Mediated Communications Substrate for LegionCCA and XCAT-C++."* In proceedings of Workshop on Component Models and Frameworks in High Performance Computing, Atlanta, GA, June 22-23, 2005.
12. I. Foster, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, J. Von Reich. *"The Open Grid Services Architecture, Version 1.0"*, Global Grid Forum, 29 January 2005, http://www.gridforum.org/documents/GFD.30.pdf [August 2005].
13. I. Foster, C. Kesselman, J. Nick, S. Tuecke. *"The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration"*, Globus Project, 2002. Available at http://www.globus.org/research/papers/ogsa.pdf.
14. Dennis Gannon, Sriram Krishnan, Liang Fang, Gopi Kandaswamy, Yogesh Simmhan, and Aleksander Slominski. *"On Building Parallel and Grid Applications: Component Technology and Distributed Services"*, Challenges of Large Applications in Distributed Environments, CLADE 2004, June 2004.
15. Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthakrishnan, Felipe Bertrand, Ken Chiu, Matt Farrellee, Madhu Govindaraju, Shriram Krishnan, Lavanya Ramakrishnan, Yogesh Simmhan, Alek Slominski, Yu Ma, Caroline Olariu, Nicolas Rey-Cenevaz. *"Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications"*, Journal of Cluster Computing, 2002.
16. Gridlab, Grid Application Toolkit. http://www.gridlab.org/WorkPackages/wp-1/index.html [August 2005].
17. The Gemss Project. http://www.gemss.de [July 2005].
18. The Globus Alliance. http://www.globus.org [August 2005].
19. The Commodity Grid (CoG) Kit. http://www.cogkit.org/ [August 2005].

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls

20. Madhusudhan Govindaraju, Himanshu Bari, and Michael J. Lewis. *"Design of Distributed Component Frameworks for Computational Grids."* In proceedings of The International Conference on Communications in Computation, pp. 160-166, June 2004.
21. Madhusudhan Govindaraju, Michael R. Head, Kenneth Chiu. *"XCAT-C++: Design and Performance of a Distributed CCA Framework."* The 12th Annual IEEE International Conference on High Performance Computing (HiPC) 2005, December 18-21, Goa, India.
22. The Grid Application Development Software (GrADS) Project. http://www.hipersoft.rice.edu/grads/ [14 August 2005].
23. GSX (Grid Service eXtensions). http://www.extreme.indiana.edu/xgws/GSX/ [November 2005].
24. ICENI - Imperial College e-Science Networked Infrastructure. http://www.lesc.ic.ac.uk/iceni/ [August 2005]
25. S. Krishnan, D. Gannon. *"XCAT3: A Framework for CCA Components as OGSA Services"* Proceedings of the Ninth International Workshop on High-Level Parallel Programming Models and Supportive Environments, April 2004, pp. 90-97.
26. Maui Cluster Scheduler. http://www.clusterresources.com/products/maui/
27. The Triana Project. http://www.trianacode.org/
28. The VGrADS Project. http://vgrads.rice.edu/ [August 2005].
29. J. Waldo. *"The Jini Architecture For Network-Centric Computing"* Communications of the ACM, 42(7):76-82, July 1999.
30. Web Service Level Agreement (WSLA) Language Specification. http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf, IBM 2001-2003
31. OASIS WSRF TC. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf [August 2005].
32. XSOAP toolkit. http://www.extreme.indiana.edu/xgws/xsoap [August 2005].
33. S. Zhou, W. Kuang, W. Jiang, P. Gary, J. Palencia, G. Gardner. *"High-Speed Network and Grid Computing for High-End Computation: Application in Geodynamics Ensemble Simulations"* Workshop on Component Models and Frameworks in High Performance Computing (Compframe 2005), Atlanta, GA, USA, June 2005.

Copyright © 2000 John Wiley & Sons, Ltd.

Prepared using cpeauth.cls