A COMPONENT PLUGIN MECHANISM AND FRAMEWORK FOR APPLICATION WEB SERVICES

Rainer Schmidt, Siegfried Benkner, and Maria Lucka
Department of Scientific Computing
University of Vienna
Nordbergstrasse 15/C/3
1090 Vienna, Austria
rainer@par.univie.ac.at

Abstract We present the architecture and application of VGE-CCA, a distributed component framework that is layered atop a Web service based Grid environment. The framework implements the CCA component model and utilizes the Vienna Grid Environment (VGE) as underlying middleware. In this paper, we introduce the concept of application specific component libraries that can be easily plugged into the container. Moreover, we report work on coupling distributed and concurrently running application components that are dynamically assembled and executed as single application composites by clients. For co-scheduling the various application components, the system makes use of advance resource reservation as provided by the VGE QoS module. Furthermore, we discuss the component and composition model as well as its application to a service-oriented architecture.

Keywords: Grid, Web Services, Service-oriented Architecture, Component Architecture

1. Introduction

Grid technology provides tools and infrastructures for the coordinated sharing of computational resources that are physically distributed, spanning multiple administrative domains. The adoption of Web service technology for Grid computing environments has been a major research issue in this area, providing defined access mechanisms for distributed resources based on Web service standards like XML, SOAP, and WSDL. Service-based Grids typically comprise of various collaborating services providing capabilities like security, information, data or resource management as described by the OGSA [8] specification. An important challenge in this area is the development of software engineering methods for Grid applications that are built upon a multitude of services as well as programming models that hide the complexity of the underlying environment.

Component technology provides a powerful way for constructing complex software systems by decoupling software implementation from application assembly. Several successful frameworks for developing distributed scientific application exists (e.g. XCAT3 [13], ProActive [2], ICENI [9], Paco++ [15], MOCCA [14]) implementing and extending a variety of component models including Corba, CCA, Fractal, or Web services. The Common Component Architecture [5] (CCA) specification defined by the CCA Forum [6] is specifically designed for the development of large scale scientific applications. The architecture focuses on the integration of existing scientific software libraries into a framework for component creation, introspection, and composition, which fits well into the Web/Grid services model as described in [11].

In this paper, we present the architecture and application of VGE-CCA, a distributed CCA implementation that allows to develop, deploy, and assemble component-based high performance applications for a Web service based Grid environment. The framework builds upon the Vienna Grid Environment [4] (VGE) - a Grid infrastructure for secure, automatic and QoS aware provision of compute-intensive applications running on parallel hardware over standard Web service technology. We introduce a mechanism allowing to extend VGE components using application specific software libraries that can be easily plugged into the container. Furthermore, we report work on coupling distributed and co-scheduled application components that are dynamically assembled by clients and run as single application composites. The following sections provide an overview of the architecture and implementation of the VGE-CCA framework as well as the underlying Grid middleware. We discuss the component and composition model as well as its application to a service-oriented architecture. Finally, we present conclusions and future work.

2. The VGE Grid Infrastructure

2.1 Architectural Overview and Technologies

The Vienna Grid Environment [4] 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-Addressing, and WS-Security. VGE supports a flexible QoS negotiation model where clients may dynamically negotiate QoS guarantees on execution time and price with potential service providers. A VGE Grid usually comprises multiple services and clients, one or more service registries for maintaining a list of service providers and the services they support, and a certificate authority for providing an operational PKI infrastructure and end-to-end security based on X.509 certificates. VGE is being utilized and evaluated in the context of the EU Project GEMSS [3] and @neurist [1] which develop Grid infrastructures for medical simulation services and data access.

2.2 Services provided by VGE Containers

VGE generic application services are configurable software units that provide common operations for remote job management, data staging, error recovery, and QoS negotiation.

The **file handling service** provides operations for uploading and downloading input/output data based on file transfer via SOAP attachments. Support for direct data exchange between services is provided by corresponding *push* and *pull* operations. The **job execution service** provides operations for launching and managing remote jobs by interfacing with a *compute resource manager*. VGE does not provide means for clients to send job scripts to the server and only allows application providers to control which scripts are to be executed on the respective machines. The **QoS negotiation service** enables clients to dynamically negotiate with VGE services on a case-by-case basis on various QoS guarantees such as execution time and price. Resulting QoS contracts between service providers and clients are formulated as Web Service Level Agreements (WSLA) and go along with advance resource reservations. The **monitoring service** generates XML 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. The Component and Composition Model

Scientific component frameworks implement and extend a variety of component models including Corba, CCA, Fractal, or Web services. A distinguishing aspect of existing component frameworks is the way they implement and exploit the various concepts of the component model. Another important design issue is the integration and leverage of a component framework with respect to the capabilities of the underlying system architecture. In the following, we briefly describe the component model and mechanisms implemented by the VGE-CCA framework.

3.1 Service-Oriented Architecture

The VGE-CCA component framework provides an abstraction layer and functionality that resides atop a Service-Oriented Architecture (SOA). This layer allows the construction of distributed Grid applications based on CCA mechanisms and transparently utilizes the underlying Web services layer. A SOA provides essential benefits such as loose coupling, location and implementation transparency. Well defined sequences of service invocations used to control remotely executing applications can be specified and executed using workflow representation and enactment techniques. VGE-CCA implements mechanisms that extend the service-oriented programming model allowing to directly interlink Web service components along accepted and provided interfaces, independently from workflow orchestration. The approach is powerful, enhancing VGE towards dynamic component interaction, data-flow, and the coupling of co-scheduled application components.

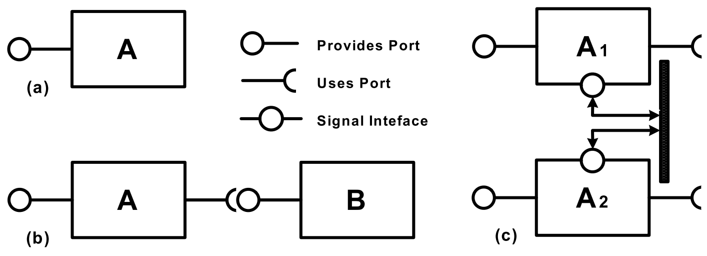

Figure 1. a) Independent Web Service b) RPC-based Component Interaction c) Event-based Application Coordination

3.2 Handling State and Composition

Figure 1 (a) depicts a Web service viewed as an encapsulated piece of software providing service through a typed interface (*provides port*). The software component may expose one or more interfaces, each defining a contract containing a set of operations together with binding information used by clients to invoke the service over a network. VGE-CCA extends this model by applying the concept of port dependencies allowing a Web service to express dependencies on services provided by other components based on defined interfaces called *uses ports*. A connection between two components, drawn by a client application developer, results in placing a handle to the selected service port into the connection table of the component requesting a remote port.

If a service maintains state, it is essential to establish a context between the requestor and the actual resource represented by a component. The way the component framework handles component instantiation is therefore an important aspect. In the context of Grid and Web services, instantiation can be realized by providing an application factory service as pointed out by Gannon et al. [10]. In VGE, we pursue a slightly different approach by maintaining a conversational identifier that is mapped to the respective application instance created and managed by the application service. In our model, stateless components may provide services (e.g. security) to other components but usually do not exhibit dependencies.

3.3 Types of Composition

VGE services encapsulate parallel applications and provide generic interfaces for controlling the execution of a component (scheduling, executing, monitoring) as well as operations for handling the data-flow between components (upload, download, data push). VGE services are stateful and multi-threaded creating a client context by maintaining a conversational identifier stored within the SOAP message header using WS-Addressing. The VGE-CCA framework provides libraries and services that extend the application services with the required mechanisms for component-based composition. Moreover, the framework provides a plugin mechanism that supports the development and deployment of individual *application component libraries* (clibs) (Section 4.1.1) encapsulating application specific logic, ports and dependencies. The current VGE-CCA implementation supports different types of composition which are explained in the following paragraphs:

Sequential Data-Flow: VGE components support data-flow by port connections allowing to directly stage i/o data between services (Figure 1 (b)). In such workflow scenario, the output of a computation typically serves as input for the following ones, for example an image reconstruction that is followed by a visualization. Data connections are explicitly controlled by the user and invoked through a corresponding *push* operation. A component may have data connections to multiple services which can be monitored and executed concurrently.

Coupled Parallel Applications: The *clibs* plugin mechanism provides the required functionality to transform applications running on different HPC computing resources into actively interacting components which can then be launched by clients as one composite application. The application components may be coordinated through asynchronous message exchange using the signal interface (Figure 1 (c)). An example using distributed Ant colony optimization is described in section 4.1.1. The event mechanism is currently implemented as a one-to-many CCA port connection. For future versions, we plan to incorporate a notification-based system like WS-Notification. For co-scheduling concurrently running component instances, we utilize scheduling and advance resource reservation as provided by the VGE QoS module.

Stateless Service Dependencies: Within VGE Grids, stateless services are typically “supporting infrastructure elements” providing services like security or data management. Accessing such services usually does not require a client to use any session or scheduling mechanisms. A dependency on an infrastructure service may be explicitly visible to a client or implicitly handled by the component and configured descriptively at service deployment time (e.g. auditing, security).

4. The VGE-CCA Component Framework

VGE-CCA implements a distributed component framework on top of a Web service based Grid of HPC application services as well as general infrastructure services such as security and information. A key design goal of VGE-CCA was the preservation of the service-oriented architecture and the provision of component extensions, without conflicting the Web services model. VGE-CCA provides a set of libraries that can be used to extend Web/Grid services as well as a set of infrastructure elements providing services to components and client runtimes. The software design allows to optionally install the VGE-CCA distribution without requiring to change code of existing services and thereby preserving the original interfaces and functionality. On the client-side, VGE-CCA provides support for component based application construction as well as workflow steering and execution.

4.1 Coupling Co-Scheduled Application Components

4.1.1 Pluggable Component Libraries (CLIBS). VGE-CCA provides a mechanism that allows to create individual software libraries that are specifically tailored to an underlying application. The component libraries (clips) can be plugged into VGE application services and are automatically deployed with the service. By default, VGE-CCA components provide interfaces for application, data, and QoS management (cf. VGE) as well as a *Builder Service*

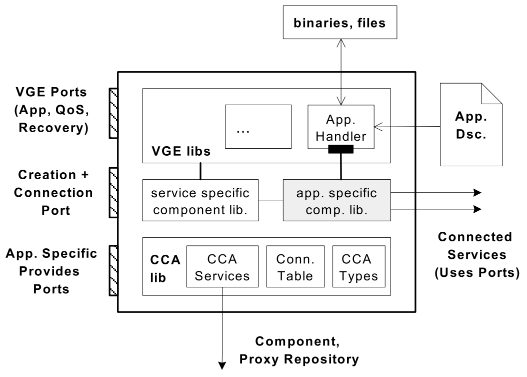
binaries, files
VGE Ports (App, QoS, Recovery)
...
App. Handler
VGE libs
App. Dsc.
Creation + Connection Port
service specific component lib.
app. specific comp. lib.
Connected Services (Uses Ports)
App. Specific Provides Ports
CCA lib
CCA Services
Conn. Table
CCA Types
Component, Proxy Repository

Figure 2. Design of a VGE-CCA component

for component creation, and connection (cf. CCA). *Clibs* are used to add individual ports, dependencies, or application specific logic to services running VGE-CCA. Moreover, the plugin mechanism allows to extend the behavior of existing services at defined entrance points, for example to trigger an activity right before/after a certain file is uploaded to the service. Application libraries are developed by subclassing a predefined component class that provides the mechanisms and handles required to augment the service and inject the desired behavior. The individual component libraries are descriptively specified and automatically loaded into the container at deployment time. Figure 2 shows a schematic design of a VGE-CCA component including VGE and CCA libraries as well as application and service specific *clibs*. Coordination among co-scheduled VGE components is distributed and currently handled using a simple signaling mechanism. The current implementation therefore extends the port connection mechanism towards supporting connections from one *uses* to n *provides* ports. Message generation and distribution is handled transparently by the framework.

4.1.2 Example: Ant Colony Optimization. Consider an application using a savings-based ant colony optimization (ACO) algorithm to solve a vehicle routing problem [7]. The application implements a multi-colony approach where several colonies of ants cooperate in finding good solutions. On the fine-grained level, each colony of ants is partitioned into n (number of processors) subcolonies that share the same pheromone matrix. The goal of parallelizing the ACO algorithm is twofold: to speed up the execution and to improve the solution quality. In order to aggregate multiple computing clusters, the application has been distributed using a custom ant component library. The VGE component was extended in order to start a daemon that keeps track of the current local optimum, written to a file by the application. If a colony calculates a better solution than the global optimum the current solutions and parts of the pheromone information are multicasted to connected components using the *signal()* interface. The coupling between the individual ant colonies is loose allowing colonies to be added or removed during runtime.

The VGE-CCA client API targets to provide useful abstractions that allow component-based application construction by hiding the complexity of the distributed system. Components are co-scheduled using QoS constraints at creation time resulting in an advance resource reservation as provided by the VGE QoS module. Connections within composites are peer-based and interaction driven, which reduces complexity at the workflow level. In the case of ACO, the client developer constructs an application by interconnecting multiple distributed ant colony components. The experiment can then be run based on a single composite entity. The code snippet in LISTING 1 shows how the client API is used to create a (simple) ACO composite. Operations for runtime steering and monitoring provided by VGE application services (e.g. *start()*, *getStatus()*) can be used likewise with the application composite.

```
//Listing 1: Ant client snippet
VgeComponent ant1 = ComponentFactory.create(coid1); //...
VgeComponentGroup antComposite =
  new VgeComponentGroup(ant1, ant2, ant3);
antComposite.upload(vrp_infile);
antComposite.start(); //...
```

4.2 The Software Distribution

In the following, we provide a short description comprising the basic building blocks of the VGE-CCA distribution. For a detailed description of implemented CCA mechanisms the reader is referred to [16].

A library package implementing the **service-side CCA framework** (Figure 2) is used to equip the application service with additional interfaces for remote component registration (component interface), creation and connection (builder interface). Additionally, the Web service is provided with a local CCA *services* library, a connection table, as well as the component plugin mechanism used to create and insert individual application components. The CCA libraries are in general used by the application component but may also be used by the individual service implementation to locate and directly connect to infrastructure services, e.g. auditing, or certificate revocation list retrieval.

A **component registry** realized as Web service implements the remote portion of the CCA *services* interface. Components register the ports they provide as well as dependencies on other components by descriptors containing the required information for discovering and utilizing the component (e.g. interface descriptors, proxy class, associated properties). Moreover, a *provides* port may also be associated with a proxy implementation that can be uploaded to the proxy registry and dynamically retrieved by components or clients. The registry service allows for dynamic service discovery and delivers the information required for component introspection (e.g. supported ports, underlying application, QoS attributes) and for component interaction (component handle, binding information) back to the requestor.

The programming environment is provided as a versatile Java **client API** that supports the creation and execution of distributed applications. Components may be described and created based on an unique identifier or an abstract component description. Applicable services are located and selected at runtime using the registry service. Client assemblies are created by interconnecting pairs of compliant *uses* and *provides* ports, which results in the establishment of peer connections between the services. The API is extensible and has a layered design supporting messaging and security, general programming constructs such as basic CCA types and mechanisms, as well as specialized application components and composites.

A **negotiation broker** service is utilized during the component creation phase to locate and create components that meet a certain Quality of Service level. The broker service utilizes the capabilities provided by the VGE QoS module to negotiate with multiple services on the various QoS guarantees. The VGE-CCA client environment integrates QoS support by providing means for qualitatively describing a VGE component. The negotiation and selection of an appropriate component is transparently delegated to the negotiation broker. A successful QoS negotiation goes along with an advance reservation of the required resource, i.e. the number of nodes on a cluster within a certain time frame, which is an essential mechanism used by the framework to co-schedule coupled application components.

5. Conclusion and Future Work

VGE-CCA serves as a framework for constructing Grid applications from native application components provided by HPC application services. We introduced a plugin mechanism for application specific component libraries allowing service providers to specifically tailor VGE services to the underlying application. A port-based connection mechanism and distributed coordination allow to couple co-scheduled application components which are represented as single composite entities on the client side. The current VGE-CCA distribution relies on Java and Web services technology. All Web service interfaces and types are described using XML schema which allows bindings to clients and components in other programming languages, such as C++ or Microsoft .Net. For future work, we plan to work on interoperability with other distributed CCA frameworks, such as XCAT-C++ [12] or LegionCCA.

References

[1] The AneurIST Project. `www.aneurist.org/`.

[2] F. Baude, D. Caromel, and M. Morel. From Distributed Objects to Hierarchical Grid Components. *International Symposium on Distributed Objects and Applications (DOA)*, Catania, Italy, 2003.

[3] S. Benkner, G. Berti, G. Engelbrecht, J. Fingberg, G. Kohring, S. Middleton, and R. Schmidt. GEMSS: Grid Infrastructure for Medical Service Provision. *Journal of Methods of Information in Medicine*, 44, 2005.

[4] S. Benkner, I. Brandic, G. Engelbrecht, and R. Schmidt. VGE - A Service-Oriented Grid Environment for On-Demand Supercomputing. In *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, November Pittsburgh, PA, USA, 2004.

[5] D. E. Bernholdt et al. A Component Architecture for High-Performance Scientific Computing. *Intl. J. High-Perf. Computing Appl.*, 2006.

[6] The Common Component Architecture Forum. `http://www.cca-forum.org`.

[7] K. Doerner, R. Hartl, S. Benkner, M. Lucka. *Cooperative Savings based Ant Colony Optimization - Multiple Search and Decomposition Approaches*, Parallel Processing Letters, 2005.

[8] I. Foster, A. Savva, D. Berry, A. Djaoui, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. V. Reich. The Open Grid Services Architecture, Version 1.0. GGF OGSA Working Group (OGSA-WG), 2005.

[9] N. Furmento, J. Hau, W. Lee, S. Newhouse, and J. Darlington. Implementations of a Service-Oriented Architecture on Top of Jini, JXTA and OGSI. In *Second Across Grids Conference*, 2004.

[10] 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.

[11] D. Gannon, R. Bramley, G. Fox, S. Smallen, A. Rossi, R. Ananthakrishnan, F. Bertrand, K. Chiu, M. Farrellee, M. Govindaraju, S. Krishnan, L. Ramakrishnan, Y. Simmhan, A. Slominski, Y. Ma, C. Olariu, and N. Rey-Cenvaz. Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. *J. Cluster Computing*, 5(3):325–336, 2002.

[12] M. Govindaraju, M. R. Head, and K. Chiu. XCAT-C++: Design and Performance of a Distributed CCA Framework. *The 12th Annual IEEE International Conference on High Performance Computing (HiPC) 2005*, Goa, India, December 18-21.

[13] S. Krishnan and D. Gannon. XCAT3: A Framework for CCA Components as OGSA Services. In *Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004)*. IEEE , 2004.

[14] M. Malawski, D. Kurzyniec, and V. Sunderam. Mocca - Towards a Distributed CCA Framework for Metacomputing. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, page 174.1, 2005.

[15] C. Prez, T. Priol, A. Ribes. Paco++: A Parallel Object Model for High Performance Distributed Systems. *37th Hawaii Intern. Conf. on System Sciences (HICSS-37)*, 2004.

[16] R. Schmidt, M. R. Head, M. Govindaraju, M. J. Lewis, and S. Benkner. Design and Implementation Choices for Implementing Distributed CCA Frameworks. *in GECO-COMPFRAME06: Workshop HPC Grid programming Environments and COmponents and Component and Framework Technology in High-Performance and Scientific Computing (at HPDC-15)*, Paris, France, June 2006.