INTEROPERABILITY OF GRID COMPONENT MODELS: GCM AND CCA CASE STUDY *

Maciej Malawski † and Marian Bubak
Institute of Computer Science and ACC CYFRONET
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Krakow, Poland
malawski@agh.edu.pl

Francoise Baude, Denis Caromel, Ludovic Henrio and Matthieu Morel
INRIA Sophia Antipolis – CNRS – Univ. of Nice Sophia Antipolis
2004, Route des Lucioles, BP 93 FR-06902 Sophia Antipolis, France
francoise.baude@inria.fr

Abstract This paper presents a case study in the generic design of Grid component models. It defines a framework allowing two component systems, one running in a CCA environment, and another running in a Fractal environment, to interact as if they were elements of the same system. This work demonstrates the openness of both Fractal and CCA component models. It also gives a very generic and exhaustive overview of the interaction strategies that can be adopted to allow full integration of these two models, like strategies for reusing in Fractal single components from the CCA world and connecting a Fractal system to an already running CCA assembly. Finally, it presents the implementation and results of investigation of interoperability between two given component frameworks: MOCCA and ProActive. In generall, this paper presents the key concepts useful to make any two component models interoperate.

Keywords: Component model, interoperability, CCA, Fractal, GCM, MOCCA, ProActive

1. Introduction

Component model may be considered as one of the most appropriate paradigm for programming Grid applications [7]. It allows to tackle the problem of complexity originating from an application and an infrastructure by providing such

*This work was supported by EU IST CoreGRID project and Polish grant SPUB-M.
†Support from the Foundation for Polish Science is kindly acknowledged.

features as composition by interfaces, support for flexible deployment and reconfiguration mechanisms.

There are few component models which address the Grid applications: the most important ones are Common Component Architecture (CCA) [3], Grid extensions of CCM (CORBA Component Model) [10] and Grid Component Model (GCM) [1] developed recently by CoreGRID project. GCM is based on the Fractal [5] and is being developed as a standard component model for programming the Grid. To achieve this goal, the abilities to interoperate with existing applications and to integrate existing "legacy" components are required. The CCA model which has been developed by the HPC community for several years, now has a number of implementations (frameworks) such as CCAFFEINE [3], XCAT [9] and MOCCA [11], and scientific components are expected to be available soon. Therefore, the problem of interoperability between GCM and CCA becomes an interesting and important issue.

In this paper, we address the problem of interoperability between GCM and CCA component models. We focus on the base component model of GCM, namely Fractal, as it defines the fundamental properties of the components and their interactions. We start with an analysis of both models to identify similarities and differences between them. Next, we discuss integration strategies and propose the solutions to the identified problems, such as issues with typing system. We propose a generic and framework independent solution, which is based on the adapter (wrapper) design pattern. In order to validate the approach, we have developed a prototype, which allows ProActive (a GCM prototype) [4] and MOCCA (a CCA implementation) [11] frameworks to interoperate. The extensions to Fractal introduced in GCM, such as collective interfaces and autonomic controllers are left for the future work.

2. Background

Interoperability can be defined as an ability of two or more entities to communicate and cooperate despite differences in the implementation language, the execution environment, or the model abstraction [14]. Today, a popular solution for interoperability between components is Web Services where standardized protocols based on XML provide the common language for applications to communicate [6]. This has been successfully applied also to high-performance modules like ASSIST modules, wrapped as GRID.it components [2].

Interoperability has been outlined as a requisite for the Grid Component Model: a realistic programming model, in the context of heterogeneous systems, component frameworks and legacy software, must be able to interoperate with existing component frameworks and allow applications to be built from a mixture of various kinds of components. Naturally, the GCM proposes to achieve general component interoperability through standardized web services.

Besides of it there are alternative interoperability approaches: our idea is to introduce mediators and adapters to build up an ad-hoc interoperability layer between selected component frameworks without superimposing on them another general-purpose interoperability framework (like a CORBA bus, or a meta-component model implemented on top of some selected existing component frameworks [13]). This alternative approach is undertaken in the work described in this paper.

3. Overview of CCA and GCM

The CCA[3] specification is defined using the Scientific Interface Description Language (SIDL) [8] which specifies the core entities: components, ports and a framework. Ports are the external interfaces of a component and they must extend the `Port` interface. A component declares both its client and server interfaces called *uses* and *provides* ports respectively. The framework is represented to the component by the `Services` interface, which is used by the component to register its ports. This interface also defines a `getPort()` method which allows a component to obtain a reference to the uses port in order to invoke methods on this client interface. The external interface exposed by the framework to the application developers is called `BuilderService`. It provides methods for creating/destroying component instances and connecting/disconnecting their ports. Besides of these core interfaces, CCA also specifies optional ports, such as component repository, connection event service, service registry and parameter ports, intended to facilitate interoperability between different frameworks.

Fractal is a *hierarchical* component model that provides *introspection* and *intercession*; it is easily *extensible* [5]. There are two kinds of components in Fractal: *primitive* components which are black boxes, and *composite* components that are composed of other components and can be used to build up yet other composites. Fractal enforces a clear separation between functional and non-functional aspects; non-functional features are provided by *controllers*, and encapsulated in a *membrane*. This model provides reconfiguration (adding, removing, binding, and unbinding) of the functional content of composites components, in order to support adaptivity of the component systems.

The GCM is a component model targeted at Grid computing, which focuses on the following extensions to the base Fractal model:

- A *deployment* paradigm based on virtual nodes allowing to specify a logical deployment of a system, and a physical deployment separately.
- Support for *several communication patterns*. First, asynchronous method calls is considered as the default semantics, and other semantics as streaming and event-based communication may be supported. A major contribution of the GCM is to standardize multicast and gathercast interfaces that allow 1-to-n and n-to-1 communications.

- Support for *non-functional adaptivity and autonomicity*. The GCM specifies how to design non-functional aspects in a component way, and thus allow the reconfiguration of the non-functional features of a component system. Finally, a set of autonomic controllers is also standardized and they allow component to adapt themselves in a much hierarchical and autonomous way.

4. Comparison of CCA and Fractal

Both CCA and Fractal component models enforce a separation between interface and their implementation, allow composition of applications by connecting client and server ports of components, and provide some reflective capabilities.

The basic and obvious similarity is that the functional interfaces of components in both models are equivalent, e.g. when considering Java implementation, both Fractal and CCA components are Java classes implementing their functional interfaces and some additional interfaces imposed by the specification. Interaction between components in both models is based on the method invocation on the client interface which is connected to a server counterpart.

The first conceptual difference is the way the components in both models interact with the outside world. In CCA, a component is given an explicit reference to the framework, and the component itself has the "initiative" to actively inform the framework about its internals, i.e. ports (interfaces). On the other hand, the Fractal model assumes that the component has a passive role in the introspection process and can reveal its internals on demand.

The second difference is the way the component interfaces are connected. In CCA the `BuilderService` is responsible for creating the connections and the framework manages them, while the component is only required to invoke `getPort()` method to get a valid reference to the port before using its client interface. In Fractal, the connection is managed by the component, by implementing a `BindingController` interface.

`ContentController` in Fractal does not have its counterpart in CCA because CCA does not support composite components as explicitly as Fractal. Also, there is no standard life cycle controller mechanism.

Although CCA does not distinguish non-functional interfaces (controllers) there are some standard ports, which are optional. One of them is is a `BasicParameterPort` which can be used to read and modify arbitrary properties of a component, analogously to Fractal `Attribute` controller.

The mechanism of component creation is also different in both models. The method for creating instances in CCA is included in the `BuilderService` port, whereas Fractal defines the `Factory` interface for this purpose. In both cases the creation mechanism may be implementation specific, and depends on the actual framework.

Although there is no standard Application Description Language (ADL) for CCA components, the `BuilderService` provides all the required functionality to construct such a description. The Application Factory project defined the XML-based ADL for XCAT [9], whereas CCAFFEINE [3] defines its own scripting language for composing applications.

5. Overcoming Typing and ADL Issues

One of the main issues in this work is to deal with the fact that Fractal (and GCM) components have an immutable type (i.e. a set of exported interfaces cannot evolve dynamically) whereas CCA component can subscribe new ports to be exported at any time. More precisely, in CCA, each component can register a port at any moment, so there is no concept such as a component type. On the contrary, in Fractal, except collection interfaces which can be instantiated several times along the life of the component, the type of a component and the set of its interfaces is fixed upon its instantiation. The "static" typing of Fractal components can be used to verify the correctness of the bindings, according to interface types. We propose the following ways of solving the typing issue:

1 Generate a Fractal component automatically upon instantiation of a CCA component, i.e. to use only the port declared by the `setServices` method. This allows to build a Fractal component automatically without any additional code (no ADL need to be specified) but prevents adding new ports after component initialization.

2 A programmer should specify the ADL for the CCA component. This means more manual effort, but no set of interfaces has to be automatically inferred. One of the main advantages of this approach is that some ports provided during the component lifetime could be specified as Fractal *optional* interfaces.

3 An improvement of the previous approach consists in generating the ADL specification upon a CCA component instantiation (not necessarily the real one) and then reuse the ADL inferred in the scenario 2 above. The user may then modify the ADL generated (to add some of the ports that will be provided during the component lifetime).

4 One can also generate an ADL from available CCA description (e.g. as SIDL [8]). The CCA script language (used by frameworks, but not standardized) may be reused to declare which ports of the CCA component / assembly should be exported.

We have chosen the second approach as it seems the most general, it enables a very good understanding of the differences between CCA and Fractal, and it is

centered on the interaction between the two frameworks. Moreover, it can be automatized later on with solutions 3 and 4.

In the all aforementioned approaches a mapping between exported CCA ports and GCM interfaces is required. More precisely, CCA ports are identified by the component name and port name, and this must be mapped to Fractal interfaces defined in the ADL. In other words, we need to define a bijection between CCA ports (i.e., component name + port name) and Fractal interfaces as it is defined in the ADL.

6. Integration Strategies

We separate CCA integration inside a GCM component system into two approaches: the encapsulation of a single CCA component (Section 6.1) and of a complete CCA system, consisting of several CCA components (Section 6.2).

Along the life time of a CCA-Fractal composition, the integration framework must support: (a) communication from the Fractal component system to the CCA system; (b) communication from the CCA system to Fractal components; (c) plugging or unplugging of Fractal interfaces to the CCA system (both on the client and on the server side); (d) exportation of new CCA ports if this is supported (see Sec. 5).

Additionally, we are looking for solution that are as general as possible, i.e. independent of CCA framework implementation as much as possible.

6.1 Simple Integration

We first focus on a simple case: how to encapsulate a single CCA component into a Fractal one?

The proposed solution enables the creation (instantiation) of a CCA component as a primitive Fractal component in a single address space. It relies on a wrapper that encapsulates a CCA component, and exposes `cca.Services` interface to a CCA component (see Fig. 1). Before instantiation we should know the type of a component in order to define the Fractal type of the component; this might be obtained from a provided ADL description.

In practice, the wrapper stores the references to bound interfaces and pass them to `getPort()` method. All the communication is done by a Fractal framework (no need to have any CCA framework running at all – the wrapper will constitute a mini-framework for that component).

6.2 Real Interoperability

In this case CCA components are created in their own framework and they are connected to Fractal components running in their framework.

Complete interoperability between two frameworks requires instantiation of a whole CCA assembly, and ability to interact with it from a Fractal framework

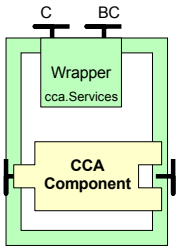

Figure 1. Integration of a single CCA component into a Fractal one

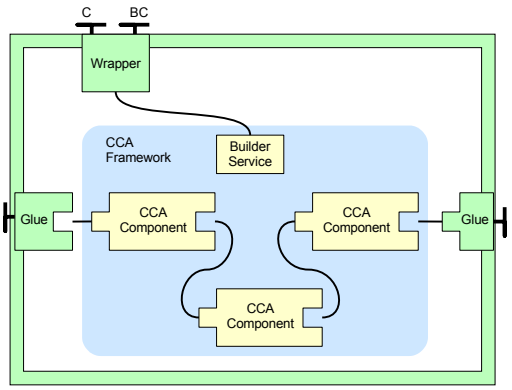

Figure 2. Interoperability between CCA and Fractal components

as if it was a Fractal composite component. In this case, we have a CCA component or a set of CCA components which are created and connected among themselves by a CCA framework (e.g. MOCCA). So, we wrap the component assembly as a Fractal component in such a way that it can be connected to other Fractal components.

The solution we propose is based on a wrapper which adds a Membrane to a CCA assembly. The wrapper should interact with the CCA framework only via `BuilderService` external interface (obtained in framework dependent manner). The wrapper is given the mapping between CCA system ports and external Fractal ports as discussed in Sec. 5 and using this information it creates GluePorts as CCA components (using `BuilderService` for each of the exported ports). The implementation of a GluePort is framework specific, and translates the Fractal invocations to CCA invocations and reversely. The GluePorts expose Fractal interfaces to the outside world, and they can be connected (bound) to other Fractal components using `BindingController` and `Component` interfaces of the wrapper. The wrapper uses the `BuilderService` to connect exported CCA ports to corresponding GluePorts using CCA framework, so the communication between CCA component assembly and GluePorts is handled by the CCA framework.

In other words, the Wrapper component is both a CCA and a Fractal component. Although Fig. 2 shows the CCA system "inside" the wrapper, it is possible also to see the Fractal system from the CCA perspective as "wrapped' one, so the solution is symmetric.

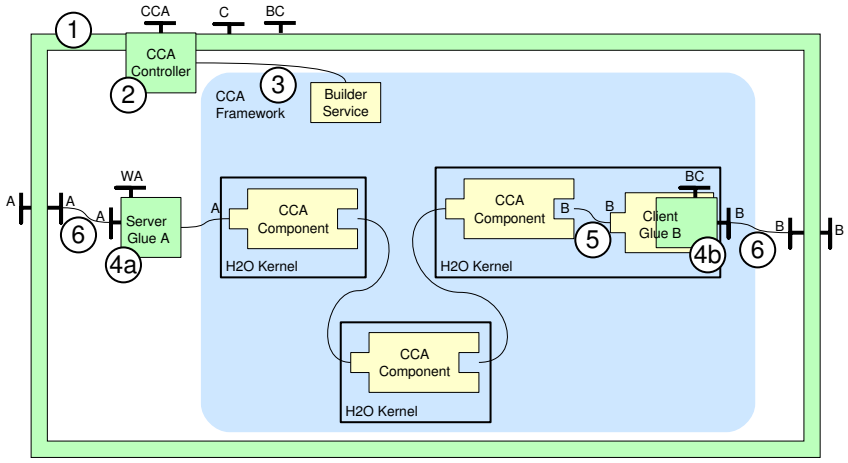

Figure 3. Wrapping an assembly of CCA components running in MOCCA framework as composite Fractal/ProActive component

7. Implementation - ProActive and MOCCA

In order to verify the proposed above solution a prototype using Java-based ProActive and MOCCA implementations was developed.

Integration of a single component was realized as planned in Sec. 6.1. A wrapper which encapsulates a CCA component and which exposes `Services` interface to a CCA component is created by the Fractal framework. The wrapper instantiates the CCA component as a local object and it invokes `setServices (this)` on a CCA component, passing the reference to itself. The CCA component registers its uses and provides ports, and consequently the wrapper can create direct (local) bindings to exported CCA ports.

In the real interoperability scenario we assume that there are CCA components running in a framework and connected using a mechanism specific to this framework (e.g. a script, or Java API), forming the existing CCA assembly. Fig. 3 shows the example of wrapping an assembly of three CCA components which provides one port of type `A` and uses one port of type `B`. The scenario consists of the following steps:

1 The Fractal framework creates a `CompositeWrapper` Component.

2 The wrapper implements a `CCAController` which is used to pass the description of the CCA assembly to the wrapper. This description includes all parameters allowing to connect the external ports of the assembly.

3 The reference to `BuilderService` is returned by a framework-specific bootstrap method. In the case of MOCCA the reference is obtained from the URI to Builder pluglet.

4 The type of Wrapper Component is obtained from an ADL or Fractal API invocations. Provided with the mapping from CCA ports to Fractal interfaces (Sec. 5), the wrapper creates the GluePorts:

   (a) For each Provides port of wrapped CCA assembly one ServerGlue port is created. It is created as a primitive Fractal component with one server interface and it has one attribute controller called WrapperAttributes, which is immediately used to pass the reference to the corresponding CCA provides port (see e.g. `ServerGlue A` on the Fig. 3). The ServerGlue component has a MOCCA client code which delegates the method invocation to the wrapped component.

   (b) For each Uses port of the wrapped system one ClientGlue is created: it is a primitive one, becoming *at the same time* the Fractal and CCA component. It is instantiated in H2O kernel (a container for MOCCA) and upon creation it launches ProActive runtime to expose the BindingController (BC). Consequently, ClientGlue can be connected to CCA components on its server side and to Fractal interfaces on the client side (see `ClientGlue B` on the Fig. 3).

5 The wrapper uses the `BuilderService` to connect the exported CCA uses ports to corresponding GluePorts.

6 CCAController connects all Glue ports to the composite Wrapper using standard Fractal bindings.

7 Fractal BindingController of a composite wrapper may be used to connect exported ports to other interfaces of the Fractal application.

It should be noted that both Client and Server Glue components are conceptually symmetric and their role is to translate invocations from one framework to the other. It was the implementation choice to create a Server Glue as ProActive component which includes the MOCCA code, whereas a Client Glue is created as MOCCA component with an "embedded" ProActive one (Fig. 3).

8. Conclusions and Future Work

The analysis of CCA and GCM component models, shows that despite some differences, it is feasible to integrate components from one model into another framework, as well as to create the glue code which enables inter-framework interoperability. The prototype functionality has been verified with a number of examples, including a non-trivial application (simulation of gold cluster formation[12]) and integrated with the ProActive library.

We observed that if the properties of two different component models can be well understood, then the generation of wrappers and glue code bridging two different component frameworks can be generic and thus automated.

Our approach resembles the one adopted in SciRun2 [13]with Bridge components acting like our GluePort ones. However, we avoid introducing the notion of a new (meta) component model and we allow components running in their native frameworks to interoperate (i.e. not requiring an additional one).

Future work will focus on automatic ADL building, generation of glue at runtime, investigating advanced features by which GCM extends Fractal model and performance tests to measure the overhead introduced by glue layer.

References

[1] CoreGRID Programming Model Virtual Institute. Basic features of the grid component model (assessed), 2006. Deliverable D.PM.04, CoreGRID, `http://www.coregrid.net`.

[2] M. Aldinucci et al. Building interoperable grid-aware ASSIST applications via WebServices. In *PARCO 2005: Parallel Computing*, pages 145–152, Malaga, Spain, 2005.

[3] R. Armstrong et al. The CCA component model for high-performance scientific computing. *Concurr. Comput. : Pract. Exper.*, 18(2):215–229, 2006.

[4] F. Baude et al. From distributed objects to hierarchical grid components. volume 2888 of *LNCS*, pages 1226 – 1242. Springer, 2003.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL component model and its support in Java. *Softw., Pract. Exper.*, 36(11-12):1257–1284, 2006.

[6] I. Foster. Service-oriented science. *Science*, 308(5723):814 – 817, 2005.

[7] V. Getov and T. Kielmann, editors. *Component Models and Systems for Grid Applications.* Springer, 2005.

[8] S. R. Kohn et al. Divorcing Language Dependencies from a Scientific Software Library. In *Proc. of the 10th SIAM Conf. on Parallel Processing for Sci. Comp.*, Portsmouth, USA, Mar. 2001. SIAM.

[9] S. Krishnan and D. Gannon. XCAT3: A Framework for CCA Components as OGSA Services. In *Proc. Int. Workshop on High-Level Parallel Progr. Models and Supportive Environments (HIPS)*, pages 90–97, Santa Fe, New Mexico, USA, Apr. 2004. IEEE.

[10] S. Lacour et al. Deploying CORBA components on a computational grid. volume 3083 of *LNCS*, pages 35 – 49. Springer, 2004.

[11] M. Malawski et al. MOCCA – towards a distributed CCA framework for metacomputing. In *Proceedings of the 10th HIPS Workshop in Conjunction with IPDPS*. IEEE, 2005.

[12] M. Malawski et al. Experiments with distributed component computing across grid boundaries. In *Proceedings of the HPC-GECO/CompFrame workshop in conjunction with HPDC 2006*, Paris, France, 2006.

[13] S. Parker et al. Integrating component-based scientific computing software. In M. A. Heroux et al., editors, *Frontiers of Parallel Processing For Scientific Computing*, chapter 15. SIAM, 2005.

[14] A. Vallecillo et al. Component interoperability. Technical Report ITI-2000-37, Departmento de Lenguajes y Ciencias de la Computacion, University of Malaga., 2000.