A Grid-Enabled Component Container for *CORBA Lightweight Components*[⋆]

Diego Sevilla[1], José M. García[1], Antonio F. Gómez[2]

[1] Department of Computer Engineering
[2] Department of Information and Communications Engineering
University of Murcia, Spain
{*dsevilla, jmgarcia*}*@ditec.um.es, skarmeta@dif.um.es*

1 Introduction and Related Work

Grid technology [3] has emerged as a new paradigm for reusing the computing power available in organizations worldwide. Particularly, Grid toolkits like Globus [6] and frameworks like *Open Grid Services Architecture* (OGSA) [11] help establishing a standard framework for integrating new developments, services, users, organizations, and resources.

Within these frameworks, which offer the foundation for the development of the Grid, distributed component models fit seamlessly to provide a higher level services for integrating and reusing components and applications.

Component models allow developing parts of applications as independent components. These components can be connected together to build applications. Components represent the unit of development, installation, deployment and reuse [15].

Taken together, the benefits of both the Grid and components raise the level of reuse and resource availability, allowing the development of a "component market", in which all the organization offer their components and services.

Traditional component models such as *Enterprise Java Beans* (EJB) and the *CORBA Component Model* (CCM) [10] are not suited for Grid computing because the enterprise services overhead. Thus, other component models oriented towards the Grid have appeared, such as the *Common Component Architecture* (CCA) [1], the work of Rana et al. [7] and Furmento et al. [4]. However, these works do not offer a complete component model, and do not specify packaging or deployment models.

In this abstract we present the *CORBA Lightweight Components* (CORBA–$\mathcal{LC}$) distributed component model and study the design and implementation strategies for its component container.

2 The CORBA–$\mathcal{LC}$ Component Model

CORBA Lightweight Components (CORBA–$\mathcal{LC}$) [13,14] is a lightweight component model based on CORBA, sharing many features with the CORBA Component Model (CCM)[9].

[⋆] Partially supported by Spanish SENECA Foundation, Grant PB/13/FS/99.

The following are the main conceptual blocks of CORBA–$\mathcal{LC}$:

- **Components**. Components are the most important abstraction in CORBA–$\mathcal{LC}$. They are both a *binary package* that can be installed and managed by the system and a *component type*, which defines the characteristics of component instances (interfaces offered and needed, events, etc.) Component characteristics are exposed by the **Reflection Architecture**.
- **Containers and Component Framework**. Component instances are run within a run-time environment called **container**. Containers become the instances view of the world. Instances ask the container for the required services and it in turn informs the instance of its environment (its *context*). Component/container dialog is based on agreed local interfaces, thus conforming a component framework. The design and implementation strategies for the CORBA–$\mathcal{LC}$ containers are described in Section 3.
- **Packaging model**. The packaging allows to build self-contained binary units which can be installed and used independently. Components are packaged in ".ZIP" files containing the component itself and its description as IDL and XML files. The packaging allows storing different binaries of the same component to match different Hardware/Operating System/ORB.
- **Deployment and network model**. The deployment model describes the rules a set of components must follow to be installed and run in a set of network-interconnected machines in order to cooperate to perform a task. CORBA–$\mathcal{LC}$ deployment model is supported by a set of main concepts: **nodes**, the **reflection architecture**, the **network model**, the **distributed registry** and **applications**.
  - **Nodes**. The CORBA–$\mathcal{LC}$ network model can be effectively represented as a set of nodes (hosts) that collaborate in computations. Nodes maintain the logical network connection, encapsulate physical host information and constitute the external view of the internal properties of the host they are running on. Concretely, they offer (Fig. 1):
    - A way of obtaining both node static characteristics (such as CPU and Operating System Type, ORB) and dynamic system information (such as CPU and memory load, available resources, etc.): ***Resource Manager*** interface.
    - A way of obtaining the external view of the local services: the ***Component Registry*** interface reflects the internal ***Component Repository*** and allows performing distributed component queries.
    - Hooks for accepting new components at run-time for local installation, instantiation and running[8] (***Component Acceptor*** interface).
    - Operations supporting the protocol for logical ***Network Cohesion***.
  - **The Reflection Architecture**. Is composed of the meta-data given by the different node services and is used at various stages in CORBA–$\mathcal{LC}$ (Fig. 1):
    - The ***Component Registry*** provides information about (a) the set of installed components, (b) the set of component instances running

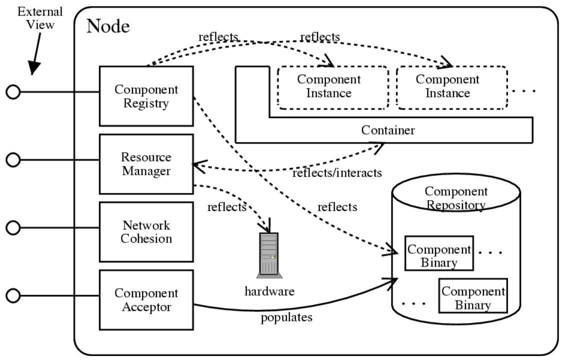

Figure 1. Logical Node Structure.

in the node and the properties of each, and (c) how those instances are connected via ports (assemblies)[12]. This information is used when components, applications or visual builder tools need to obtain information about components.

∗ The ***Resource Manager*** in the node collaborates with the ***Container*** implementing initial placement of instances, migration/load balancing at run-time. Resource Manager also reflects the hardware static characteristics and dynamic resource usage and availability.

- **Network Model and The Distributed Registry**. The CORBA–$\mathcal{LC}$ deployment model is a network-centered model: The complete network is considered as a repository for resolving component requirements. Each host (node) in the system maintain a set of installed components in its ***Component Repository***, which become available to the whole network. When component instances require other components, the network can decide either to fetch the component to be locally installed, instantiated and run or to use it remotely. This network behavior is implemented by the ***Distributed Registry***. It stores information covering the resources available in the network as a whole.
- **Applications and Assembly**. In CORBA–$\mathcal{LC}$, ***applications*** are just special components. They are special because (1) they encapsulate the explicit rules to connect together certain components and their instances (*assembly*), and (2) they are created by users with the help of visual building tools. Thus, they can be considered as ***bootstrap*** components.

3 A Grid-Enabled Container for CORBA–$\mathcal{LC}$

Containers in CORBA–$\mathcal{LC}$ mediate between component instances and the infrastructure (both CORBA–$\mathcal{LC}$ runtime and the Grid middleware). Instances ask the container for needed resources (other components, for instance), and it, in turn, provides them with their *context*. Figure 2 shows the building blocks of a container, as well as its responsibilities within a call by the component client. Concretely, container responsibilities include:

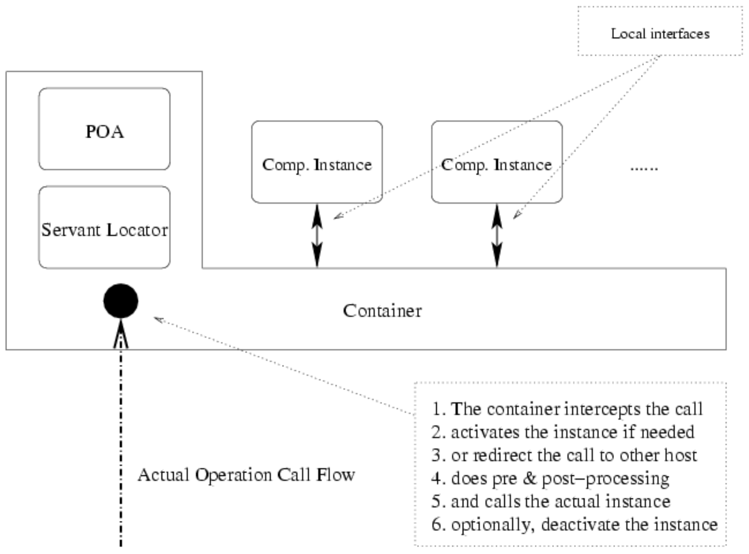
Local interfaces
POA
Comp. Instance
Comp. Instance
......
Servant Locator
Container
Actual Operation Call Flow
1. The container intercepts the call
2. activates the instance if needed
3. or redirect the call to other host
4. does pre & post–processing
5. and calls the actual instance
6. optionally, deactivate the instance

Figure 2. Component instances and interception within a Container.

– **Manage component instances**. It interacts with the component factory to maintain the set of active component instances, as well as instance activation and deactivation.
– **Provide a controlled execution environment for instances**. They obtain all the network resources (their *view* or *context*) from the container, becoming their representative into the network. The communication between container and instances is made through agreed local interfaces.
– **Provide transparent fault tolerance, security, migration, and load balancing**. The container intercepts all the calls made to the component instances it manages. This gives the container the chance to redirect the calls to the correct component or to group component instances to implement fault tolerance. There are two main strategies to implement this behavior:
  • **Interception**. Using CORBA as a foundation, *Portable Interceptors* [10] can be used to intercept every call to the instance. This is more flexible and generic, but inefficient.
  • **Code generation**. With this approach, an utility can take interface definitions and non-functional aspects of the component (described as XML files) and generate a customized container. The container becomes a wrapper for the component. This approach is more efficient because the container is made *for* the component. Moreover, the code generation can also convert the generated container into an *adapter* [5] to offer the component interfaces as Grid Services compliant with the OGSA specification [11].

Containers follow the Aspect-Oriented Programming (AOP) philosophy [2]. Components specify the non-functional requirements (*aspects*) they need from the environment. While traditional component models as EJB and CCM cover the needs of enterprise-oriented applications, we believe they are not suited for dealing with Grid or HPC applications, because of the burden of enterprise-oriented services such as persistence or transactions. Moreover, they have a fixed set of aspects, making them difficult to adapt to those environments.

We have identified a set of aspects that components can specify so that the container can manage them effectively in a Grid environment. This list is not exhaustive, and is a result of our ongoing research on this area:

- **Instance lifetime and service**. Defines the lifetime of instances. Helps the container to manage instances.
- **Integration with grid security**. Specifies the security restrictions for this component. The container must ensure them by leveraging the grid infrastructure.
- **Mobility**. If the component can travel or must be run remotely. The former allows physical component distribution. The latter is semantically equivalent to a OGSA service.
- **Fault Tolerance** and **Replication**. The container must ensure the level of fault tolerance required by the component (number of replicas, etc.)
- **Data Aggregation and distribution**. This is interesting for data-parallel components, which can specify how many *workers* they need for the realization of their work. They also have a protocol to join partial results. The container is in charge of finding the workers, delivering the data and bringing back the results to the instance through agreed local interfaces.

4 Status and future work

Current status of CORBA–$\mathcal{LC}$ allows building components and connect them. We are currently researching in the area of aspects suited for grid computing and the design and implementation of the CORBA–$\mathcal{LC}$ container.

References

1. CCA Forum. *The Common Component Architecture Technical Specification - Version 1.0.* http://z.ca.sandia.gov/~cca-forum/gport-spec/.
2. J. Fabry. Distribution as a set of Cooperating Aspects. In *ECOOP'2000 Workshop on Distributed Objects Programming Paradigms*, June 2000.
3. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1999.
4. N. Furmento, A. Mayer, S. McGough, S. Newhouse, T. Field, and J. Darlington. Optimisation of Component-based Applications within a Grid Environment. In *SuperComputing 2001, Denver*, November 2001.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.
6. *The Globus Project Home Page.* http://www.globus.org/.

7. M. Li, O. F. Rana, M. S. Shields, and D. W. Walker. A Wrapper Generator for Wrapping High Performance Legacy Codes as Java/CORBA Components. In *Supercomputing'2000 Conference*, Dallas, TX, November 2000.
8. R. Marvie, P. Merle, and J-M. Geib. A Dynamic Platform for CORBA Component Based Applications. In *First Intl. Conf. on Software Engineering Applied to Networking and Parallel/Distributed Computing (SNPD'00)*, France, May 2000.
9. Object Management Group. *CORBA Component Model*, 1999. OMG Document ptc/99-10-04.
10. Object Management Group. *CORBA: Common Object Request Broker Architecture Specification, revision 3.0.1*, 2002. OMG Document formal/02-11-01.
11. *Open Grid Services Architecture.* http://www.globus.org/ogsa.
12. N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair. Towards a Reflective Component-based Middleware Architecture. In *ECOOP'2000 Workshop on Reflection and Metalevel Architectures*, 2000.
13. D. Sevilla, J. M. García, and A. Gómez. CORBA Lightweight Components: A Model for Distributed Component-Based Heterogeneous Computation. In *EUROPAR'2001*, Manchester, UK, August 2001.
14. D. Sevilla, J. M. García, and A. Gómez. Design and Implementation Requirements for CORBA Lightweight Components. In *Metacomputing Systems and Applications Workshop (MSA'01)*, Valencia, Spain, September 2001.
15. C. Szyperski. *Component Software: Beyond Object-Oriented Programming.* ACM Press, 1998.