CORBA *Lightweight Components*: A Model for Distributed Component-Based Heterogeneous Computation*

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

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

Abstract. In this article we present CORBA *Lightweight Components*, CORBA–$\mathcal{LC}$, a new network-centered reflective component model which allows building distributed applications assembling binary independent components spread on the network. It provides a *peer* network view in which the component dependencies are managed automatically to perform an intelligent application run-time deployment, leading to better utilization of resources. We show the validity of the CORBA–$\mathcal{LC}$ approach in dealing with CSCW and Grid Computing applications.

1 Introduction

Component-Based Development[21] has emerged as the natural successor of the Object-Oriented paradigm. Components allow (1) to develop independent binary units that can be packaged and distributed independently, and (2) to build modular applications based on the assembly of those binary units. Binary interchangeability leads to the maximum reuse as components can be added to (and removed from) a system even without the need of recompiling, provided that the components state what they require and what they offer to the system.

When component technology is applied to distributed applications, programmers can develop independent components that can interact transparently with other components residing in remote machines. However, while programmers would expect the component infrastructure to utilize all the computing power and resources available for running their components, traditional component models force programmers deciding the hosts in which their components are going to be run and build a "static" description of the application (*assembly*).

This article describes CORBA *Lightweight Components* (CORBA–$\mathcal{LC}$), a new component model based on CORBA[13]. CORBA–$\mathcal{LC}$ offers the traditional component models advantages (modular applications development connecting binary interchangeable units) allowing automatic placement of components in network

* Partially supported by Spanish SENECA Foundation, Grant PB/13/FS/99.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 845–854, 2001.

nodes, intelligent component migration and load balancing, leading to maximum network resource utilization. Thus, it introduces a more *peer* network-centered model in which all node resources, computing power and components can be used at run-time to automatically satisfy applications dependencies.

This paper is organized as follows. Section 2 describes the CORBA–$\mathcal{LC}$ Component Model. Section 3 outlines other distributed component models and how CORBA–$\mathcal{LC}$ is related to them. Section 4 describes the principal application domains we are targeting in this research: Computer-Supported Cooperative Work (CSCW) and Grid Computing. Finally, Section 5 presents the conclusions, current development status and future work.

2 CORBA *Lightweight Components*

CORBA–$\mathcal{LC}$ is a lightweight component model based on CORBA, sharing many features with the CORBA Component Model (CCM)[12].

2.1 Components

Components are the most important abstraction in CORBA–$\mathcal{LC}$. They can be seen, at least, under two different dimensions:

1. **Binary package**: Components are independent binary units that can be used to compose applications. Thus, components include information which allow them (a) to be installed, (b) to be managed as a binary package, (c) to be dynamically (un)loaded as a Dynamic Link Library (DLL), and (d) to be instantiated. This information includes static hardware and software (such as Operating System, *Object Request Broker*) dependencies (***Static*** dimension).
2. **Component Type**: Component instances are run-time incarnations of the behavior stored in a component. Thus, components are also a description of run-time properties and requirements of their instances (***dynamic*** dimension). These properties and requirements can be either internal or external.

Internal properties describe requirements and properties that instances expose to the framework in which they are immerse. This framework is described in the following subsections.

External properties are those that component instances expose to their clients (other components or applications). These external communication points are collectively called "ports". Ports allow components to be connected together to accomplish the required task. A set of IDL and XML files is used to establish the *minimal* set of ports a component needs from and offers to other components. Those files are included within the component binary package (§2.3).

CORBA–$\mathcal{LC}$ does not limit the different port kinds that a component can expose. However, there are two basic kinds of ports: interfaces and events.

A component can indicate that its instances (***provide***) or ***use*** some interfaces[1] for their internal work. Interfaces represent agreed synchronous communication points between components.

Events can be used as asynchronous communication means for components. They can also specify that they produce or consume some kind of event in a publish/subscribe fashion. For each event kind produced by a component, the framework opens a push event channel. Components can subscribe to this channel to express its interest in the event kind produced by the component.

The set of external properties of a component is not fixed and may change at run-time. This is supported by the **Reflection Architecture** described in §2.4.

Finally, factory interfaces[5] are needed in CORBA–$\mathcal{LC}$ to manage the set of instances of a component. Clients can search for a factory of the required component and ask it for the creation of a component instance. Factory code can be automatically generated depending on component requirements.

2.2 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*). As in CCM and Enterprise Java Beans (EJB)[20], the component/container dialog is based on agreed local interfaces, thus conforming a component framework. Containers leverage the component implementation of dealing with the non-functional aspects of the component[3], such as instance activation/de-activation, resource discovery and allocation, component migration and replication, load balancing[14] and fault tolerance among others. Containers also act as component instance representatives into the network, performing distributed resource queries in behalf of their managed component instances.

2.3 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. This is similar to the CCM Packaging Model[12]. This information can be used by each node in the system to know how to install and instantiate the component. The packaging allows storing different binaries of the same component to match different Hardware/Operating System/ORB.

2.4 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

[1] We use "interface" here in the same sense that it is used in CORBA.

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 that collaborate in computations. ***Nodes*** are the entities maintaining the logical network behavior. Each host participating must have running a server implementing the *Node* service. 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 queries.
- Hooks for accepting new components at run-time for local installation, instantiation and running[10] (***Component Acceptor*** interface).
- Operations supporting the protocol for logical ***Network Cohesion***.

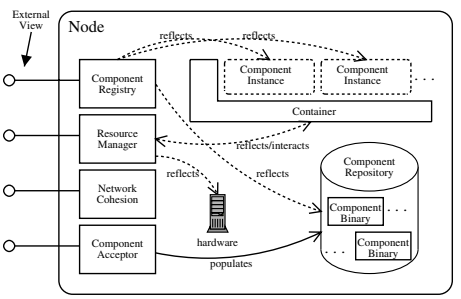

Fig.1. Logical Node Structure

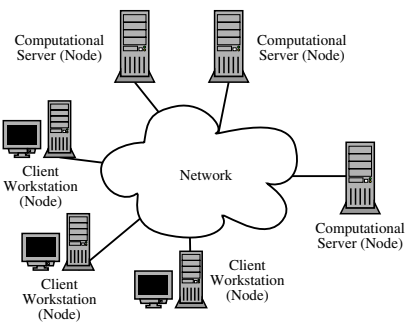

Fig.2. Network-centered architecture

The Reflection Architecture 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 in the node and the properties of each, and (c) how those instances are connected via ports (assemblies)[15]. 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.

With the help of the reflection architecture, new components (or new version of existing components) can be aggregated to the system at any time, and become instantly available to be used by other components.

In contrast to CCM, the set of external properties of a component is not fixed and may change at run-time. Thus, component instances can adapt to the changing environment requesting new services or offering new ones. CORBA–$\mathcal{LC}$ offers operations which allow modifying the set of ports a component exposes[17].

Network Model and The Distributed Registry The CORBA–$\mathcal{LC}$ deployment model is a network-centered model (Fig. 2): 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 issues the corresponding distributed queries to each node's ***ComponentRegistry*** in order to find the component which match better with the stated QoS requirements. Once selected, the network can decide either to fetch the component to be locally installed, instantiated and run or to use it remotely (a component decoding a MPEG video stream would work much faster if installed locally).

This network behavior is implemented by the ***Distributed Registry***. It stores information covering the resources available in the network as a whole, and is responsible of managing these. Component Registries, Resource Managers and the Network Cohesion interface of each node support the Distributed Registry behavior. Component Registries collaborate to resolve distributed component queries and reflect the internal Component Repository of each node.

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, and (2) they are created by users with the help of visual building tools.

With the given definition, applications can be considered as ***bootstrap*** components: when applications start running, they expose their explicit dependencies, requiring instances of other components and connecting them following the user stated pattern for that particular application. This is similar to what in CCM is called an ***assembly***. Conversely, in CORBA–$\mathcal{LC}$ the matching between component required instances and network-running instances is performed at run-time: the exact node in which every instance is going to be run is decided when the application requests it, and this decision may change to reflect changes in the load of either the nodes or the network. The deployment of the application, instead of being fixed at deployment-design time, is intelligently performed

at run-time, which allows intelligent run-time scheduling, migration and load balancing schemes. Fixed *versus* run-time deployment can be compared with static *versus* dynamic linking of Operating Systems libraries, but augmented to the distributed heterogeneous case.

3 Related Work

To date, several component models have been developed. Although CORBA–$\mathcal{LC}$ shares some features with them, it also has some key differences.

Java Beans[19] is a framework for implementing Java-based desktop applications. It is limited to both Java and the client side of the application. In contrast, CORBA–$\mathcal{LC}$ is not limited to Java and allows components to be distributed among different hosts, still allowing seamless integration of local GUI components.

Microsoft's Component Object Model (COM)[11] offers a component model in which all desktop applications are integrated. Its main disadvantages are that, (1) it does not fit well the distributed case (DCOM), and (2) its support is rather limited to Windows. Moreover, COM components do not expose their requirements (other required components)[15,8]. CORBA–$\mathcal{LC}$ inherits from CORBA its Operating System, programming language and location transparency, thus effectively adapting to heterogeneous environments. Moreover, it is designed from the beginning to automatically exploit the computing power and components installed in the network using its Reflection Architecture.

In the server side, SUN's Enterprise Java Beans[20] and the new Object Management Group's CORBA Component Model (CCM)[12,18] offer a server programming framework in which server components can be installed, instantiated and run. These models are fairly similar, but EJB is a Java-only system, and CCM continues the CORBA heterogeneous philosophy. Both models are designed towards supporting enterprise applications, thus offering a container architecture with convenient support for transactions, persistence, security, etc.[17] They also offer the notion of components as binary units which can be installed and executed (following a fixed assembly) in Application Servers.

Although CORBA–$\mathcal{LC}$ shares many features with them, it presents a more dynamic model in which the deployment is performed at run-time using the dynamic system data offered by the Reflection Architecture. It also allows adding new components and modifying component instances properties and connections at run-time and reflecting those changes to visual building tools. It is a *lightweight* model: the main goal is the optimal network resource utilization instead of supporting the overhead of enterprise-oriented services. This complexity is one of the main reasons why the CCM specification is still not finished.

In general, component models have been designed to be either client-side ***or*** server-side. This forces programmers to follow different models for different layers of applications. CORBA–$\mathcal{LC}$ offers a more *peer* approach in which applications can utilize all the computing power available, including the more and more powerful user workstations and high-end servers. Application components

can be developed using a single component model and spread into the network. They will be intelligently migrated into the required hosts. Thus, a *homogeneous* component model can be used to develop all the tiers (GUI, application logic) of distributed multi-tiered applications.

In[8], a dynamic configuration management system is described. This work provides us with valuable ideas for our research. However, it is centered in the process of automatic component configuration and does not offer a complete component model.

4 Application Domains

The CORBA–$\mathcal{LC}$ model represents a very convenient infrastructure for developing applications in a wide range of domains. It can be seen as a general purpose infrastructure. However, we are specially interested in dealing with Computer-Supported Cooperative Work (CSCW) and Grid Computing.

4.1 Computer Supported Cooperative Work (CSCW) Domain

Collaborative work applications allow a group of users to share and manipulate a set of data (usually multi-media) in a synchronous or asynchronous way regardless of user location[22]. We are interested in the development and deployment of ***synchronous*** CSCW applications, including video-conferencing, shared whiteboard and workspaces, workflow and co-authoring systems. CORBA–$\mathcal{LC}$ represents an optimal environment for various reasons:

- It offers a peer distributed model, which matches the inherently peer distributed nature of these applications.
- GUI components can be considered within the whole application design, allowing the presentation layer to evolve smoothly.
- It allows bandwidth-limited, multimedia components (such as video stream decoding) to be migrated and installed locally to minimize network load.
- It allows Personal Digital Assistants (PDAs) to be used as normal nodes with limited capabilities: they can use all components remotely.

Figure 3 depicts the relationships between a CSCW application and other components, including GUI ones. The latter can be either local or remote, painting in their portion of the window using the local *Display* component which provides painting functions. Applications can change how the data is shown by replacing the GUI components at run-time. Can be all remote, enabling thin clients such as PDA.

4.2 Grid Computing Domain

Our view of Grid Computation targets scalable and intelligent resource and CPU usage within a distributed system, using techniques such as *IDLE*

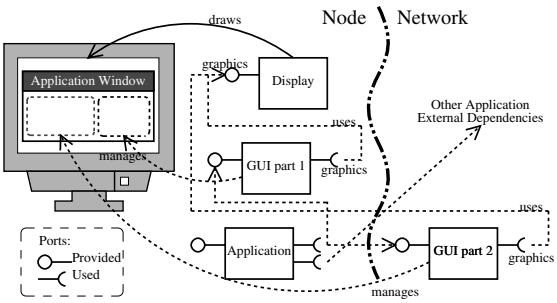

Fig.3. CSCW application model

computation[6] and *volunteer computing*[16]. These techniques fit seamlessly within the CORBA–$\mathcal{LC}$ model to suit Grid Computation needs.

Other component-based alternatives such as the Common Component Architecture (CCA)[1] have appeared in the High-Performance Computing (HPC) community. These models introduce components kinds which reflect the special characteristics of the field (for example, components whose instances must be split to perform a highly-parallel task). While we find this approach very interesting, those models usually become only a minimum wrapper[9] for reusing legacy scientific code and do not offer a complete component model. A similar approach was presented also by Walker *et al.*[23]. Their interest is in Problem-Solving Environments (PSEs) using an XML-based component model to wrap legacy scientific components.

FOCALE[2] offers a component model for grid computation. It uses CORBA and Java (although it supports legacy applications). It provides a system view at different levels: federation, server, factories, instances and connections.

Developments in the Grid Computing field include Globus[4] and Legion[7]. They are systems which offer services for applications to access to the computational grid. However, they are huge systems, difficult to manage and configure, somewhat failing in its primary intentions. Moreover, they do not address very well the interoperability and code reuse through component technology.

5 Conclusions and Future Work

In this article we have described the CORBA–$\mathcal{LC}$ Component Model. Also, we have stated the validity of the design to target the CSCW and Grid Computing domains. Current CORBA–$\mathcal{LC}$ implementation allows building components with the stated external characteristics and packaging. However, the implementation is still incomplete, so we have some future work to do: Explore strategies to maintain the described Reflection Architecture and the network-awareness of both nodes and the Distributed Registry[8], also introducing fault-tolerance techniques; Implement visual building tools allowing users to build applications based on all available network components; Further identify CSCW and Grid-

based application needs enhancing CORBA–$\mathcal{LC}$ to better support them; And study the integration of this model with future CCM implementations.

Finally, we plan to continue enhancing CORBA–$\mathcal{LC}$ as a general computing platform, to offer programmers both the advantages of the Component-Based Development and Distributed Computing.

References

1. R. Bramley, K. Chiu, S. Diwan, D. Gannon, M. Govindaraju, N. Mukhi, B. Temko, and M. Yechuri. A Component Based Services Architecture for Building Distributed Applications. In *Proceedings of the High Performance Distributed Computing Conference*, 2000. 852
2. G. S. di Apollonia, C. Gransart, and J-M. Geib. FOCALE: Towards a Grid View of Large-Scale Computation Components. In *Grid'2000 Workshop, 7th Int. Conf. on High Performance Computing*, Bangalore, India, Dec. 2000. 852
3. J. Fabry. Distribution as a set of Cooperating Aspects. In *ECOOP'2000 Workshop on Distributed Objects Programming Paradigms*, June 2000. 847
4. I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishing, 1999. 852
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995. 847
6. D. Gelernter and D. Kaminsky. Supercomputing out of Recycled Garbage: Preliminary Experience with Piranha. In *Sixth ACM International Conference on Supercomputing*, pages 417–427, July 1992. 852
7. A. S. Grimshaw and Wm. A. Wulf. The Legion Vision of a Worldwide Virtual Computer. *Communications of the ACM*, 40(1), January 1997. 852
8. F. Kon, T. Yamane, C. Hess, R. Campbell, and M.D. Mickunas. Dynamic Resource Management and Automatic Configuration of Distributed Component Systems. In *Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001)*, San Antonio, Texas, February 2001. 850, 851, 852
9. 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. 852
10. 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. 848
11. Microsoft. *Component Object Model (COM)*, 1995. http://www.microsoft.com/com. 850
12. Object Management Group. *CORBA Component Model*, 1999. OMG Document ptc/99-10-04. 846, 847, 850
13. Object Management Group. *CORBA: Common Object Request Broker Architecture Specification, revision 2.4.1*, 2000. OMG Document formal/00-11-03. 845
14. O. Othman, C. O'Ryan, and D. Schmidt. The Design and Performance of an Adaptative CORBA Load Balancing Service. *Distributed Systems Engineering Journal*, 2001. 847
15. 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. 848, 850

16. L. F. G. Sarmenta. Bayanihan: Web-Based Volunteer Computing Using Java. In *2nd International Conference on World-Wide Computing and its Applications (WWCA'98)*, March 1998. 852
17. D. Sevilla. CORBA & Components. Technical Report TR-12/2000, University of Extremadura, Spain, 2000. 849, 850
18. D. Sevilla. *The CORBA & CORBA Component Model (CCM) Page*, 2001. http://www.ditec.um.es/~dsevilla/ccm/, visited April, 2001. 850
19. SUN Microsystems. *Java Beans specification*, 1.0.1 edition, July 1997. http://java.sun.com/beans. 850
20. SUN Microsystems. *Enterprise Java Beans specification*, 1.1 edition, December 1999. http://java.sun.com/products/ejb/index.html. 847, 850
21. C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. ACM Press, 1998. 845
22. G. Henri ter Hofte. *Working Apart Toguether. Foundation for Component Groupware*. PhD thesis, Telematica Institut, The Netherlands, 1998. 851
23. D. Walker, O. F. Rana, M. Li, M. S. Shields, and Y. Huang. The Software Architecture of a Distributed Problem-Solving Environment. *Concurrency: Practice & Experience*, 12(15):1455–1480, December 2000. 852