Towards Resource-Aware Parallel Components

Yves Mahéo, Frédéric Guidec, Luc Courtrai

Towards Resource-Aware Parallel Java Components

Yves MAHÉO, Frédéric GUIDEC, Luc COURTRAI
VALORIA Laboratory, Université de Bretagne-Sud
BP 573, 56017 Vannes Cedex, France
Email: {Yves.Maheo|Frederic.Guidec|Luc.Courtrai}@univ-ubs.fr

Keywords : Parallel components, resource-awareness, distributed applications, monitoring, Java

Abstract

This paper reports the development of the Concerto platform, which is dedicated to supporting the deployment of resource-aware parallel Java components on heterogeneous distributed platforms, such as pools of workstations in labs or offices. We propose a basic model of a parallel Java component and present some tools that facilitate the management and the deployment of such a component on a distributed architecture. The Concerto platform also defines and implements an open and extensible framework dedicated to the observation of the distributed environment on which a component is deployed. Support is provided for modeling the execution environment as a set of resources and for monitoring these resources in a distributed way.

1 Introduction

Building distributed applications is known to be a difficult task, more especially as the targeted architectures become more and more heterogeneous. Indeed, there is a growing demand for distributed applications that can be ported on a wide variety of platforms. Such a platform is for example a cluster obtained by assembling regular workstations in a lab, and may include devices linked through wireless connections. The component-oriented approach can help significantly in developing complex applications. So-called "software components" can serve as deployment units, and further be assembled while developing other components, or full-featured applications. Although the advantages of this approach are now widely admitted, little effort has been invested so far on the development of components specifically designed for distributed platforms.

We present in this paper a middleware platform for "parallel components", that is, components that are meant to be deployed on a cluster[1] and that each encapsulate a parallel and distributed code. Moreover, we provide these components with means to perceive their execution environment, so they can adapt dynamically to the changes they can observe in runtime conditions. By doing so, we try to foster the development of *adaptive* parallel components.

Although component adaptability is our prime objective, this paper does not focus on this topic per se. Instead it presents a Java-based middleware platform we designed in order to provide support for such components. The platform Concerto thus implements a number

[1] In this paper, we use the term *cluster* to refer to the distributed platform targeted by the deployment of a component. It is not restricted to a high-performance LAN but can be any heterogeneous network of computers.

of facilities for deploying, launching, and controlling the execution of parallel components on a distributed system. Concerto relies on facilities we have implemented in D-RAJE *(Distributed Resource-Aware Java Environment),* an open and extensible framework that makes it possible to monitor the state of the many resources offered in a cluster.

2 Support for resource modeling and monitoring

One of the main objectives of project Concerto is to provide parallel software components with means to perceive their runtime environment, so that they can adapt to variations observed in its characteristics. Environment modeling and monitoring in distributed systems has already been addressed in numerous projects pertaining for example to Grid computing or to network computing (such as [5, 10, 9, 13]). In many of these projects, though, resource modeling is performed at a rather coarse grain (*eg* computing node, global free memory, network link), and information about resources is often limited to system resources, and collected using *ad hoc* methods. An important part of Concerto is made of a software framework called D-RAJE. This platform was designed so as to offer an extensible model in which any kind of resource in a distributed environment can be reified as a Java object. It also provides various facilities for monitoring these resources in a homogeneous way. A brief description of D-RAJE is given below (see [8] for a more detailed presentation).

2.1 Resource modeling

With D-RAJE a distributed system can be modeled and monitored using Java objects that reify the various resources offered in this system. As a general rule, we qualify as "resource" any hardware or software entity a software component is liable to use during its execution. To date the types of resources considered in D-RAJE –and thus in the Concerto platform– include system resources (CPU, system memory, swap, network interfaces, etc.) that chiefly characterize the underlying hardware platform, as well as so-called "conceptual" resources (sockets, processes, threads, directories, files, RMI servers, etc.) that rather pertain to the applicative environment in which a component is running. An extensible hierarchy of classes models these resources. Any resource object (that is, any instance of one of these classes) is capable of producing on demand an observation report, which provides a summary of its state.

From the viewpoint of an application programmer, resource objects can be created explicitely, or one can rely on a resource discovery monitor that is able to automatically instantiate (and destroy) resource objects pertaining to some classes of resources.

At creation time, resource objects are added to a local resource register. A fully distributed resource manager, accessible on each node of the system, offers the view of a global register and thus makes it possible to access to information about resources in the same way, regardless of where the resources considered are actually located in the system.

D-RAJE somehow compares to the Common Information Model [7] as far as resource modeling is concerned. However, D-RAJE goes beyond pure modeling as it additionally implements facilities for collecting and exploiting information about the state of any system or conceptual resource.

2.2 Resource selection and observation

The resource manager can be consulted when looking for a specific kind of resource in the system, or in order to request information about the status of any resource in the system. Since the population of resource objects can be quite large, a selection can be applied based on their location (*e.g.* on a specific node, globally or in the neighbourhood), their types, or any other criterion. Resource filtering is obtained by using so-called resource patterns. Any programmer can define new patterns, although many patterns are already available as predefined patterns in D-Raje. With these patterns, one can for example identify all objects modeling CPUs, network interfaces or TCP sockets. These objects can additionally be searched for on a specific cluster node, or in the whole cluster. If required one can even perform a more selective search, and look for example for TCP or UDP sockets that have been bound to a local port whose number is in a specific range (say, values between 0 than 1024), and that maintain connections with remote hosts whose IP addresses conform to a specific pattern (say, all adresses in domain 195.83.160/22). Resource identification can thus be achieved at quite a fine grain if needed.

Once a resource has been identified, information on its status can be obtained through any one of two observation mechanisms. The first of these mechanisms allows direct observation of the resource considered: the resource manager can be asked to collect and return an observation report concerning a specific resource, wherever this resource is located in the system. The second obervation scheme relies on an event-based model: local or distant monitors are automatically created when needed. These monitors can observe the resource considered periodically, and notify the user when user-specified events occur. [8] provides a thorough description of these topics.

3 Parallel Components

The Concerto platform is dedicated to supporting adaptive parallel components. Although the concept of parallel component is central in our project, our aim is not to propose a new component model, but to provide an infrastructure that enforces the adaptability of components. We propose below a definition of what we mean by parallel component in Concerto.

A programmer who wishes to develop a parallel component for Concerto must design his component as a set of cooperating threads. He must also define the business part of the component (name, interface, implementation). The platform offers facilities for managing non-functional aspects of the component.

3.1 Component interface

Any parallel component hosted by the Concerto platform must defined the following interfaces.

- *Business interface*: No constraint is put on the type of the business interface. The programmer of a component may for example propose an interface based on Java RMI. The component is then an object implementing the *Remote* interface, whose methods will be called remotely by the clients of the component. The component may also behave as a server listening to a specific TCP or UDP port, with which a client program can communicate. Alternatively, one may design a distributed interface (that

is to say, an interface associated with several objects that each implement a part of the interface) in order to be connected in parallel with another parallel component.

- *Life cycle interface*: Through this interface, the different steps of the life of the component can be controlled. To date, this covers deploying the component on the cluster, launching all the activities (*ie* starting the threads) of the components and stopping these activities. In the future we may include persistence services in this interface.

- *Resource interface*: The component exhibits a resource interface through which the user can access information related to the resources used by the component. Actually, the component itself is considered as a resource. Hence any component in Concerto must implement a method *observe()* that returns an observation report. By default, the observation report generated by a component simply aggregates the observation reports of all the resources it itself uses. For security reasons, the component's programmer may change this default implementation, for example by restricting the amount of information disclosed to the component's client.

3.2 Internal Structure

When building a parallel component, the programmer develops a set of Java threads that cooperate to perform the functions specified in the component's business interface.Threads are gathered into placement (or distribution) entities called fragments. A fragment is a set of threads that belong to a single component, and that must run within the same virtual machine on the same cluster node. The threads of a fragment will thus be able to share a common object space. Communication and synchronization between such threads are performed just like in any other multi-threaded Java program. On the other hand, threads that belong to different fragments must rely on external communication and synchronization mechanisms, such as sockets and RMI.

It must be noticed that information on the component's structure is available through facilities implemented in D-Raje, since threads, fragments and the component itself are modeled as resources in D-Raje. As already mentioned, some of this information is also available to other components through each component's resource interface.

3.3 Deployment

The deployment of a distributed application on a parallel platform is known to be a tedious and error-prone task. In order to alleviate the deployment of parallel components on a cluster of workstations, we have developed an XML dialect that permits a component programmer to describe the component's structure, placement directives for its fragments, and constraints imposed for the deployment to be feasible (availability of a specific version of the JVM, of a RMI registry, etc.)

The Concerto platform includes an administration tool with which a cluster of workstations can be managed, using either a command line or a graphical user interface. With this administration tool workstations can be added to or removed from the cluster at any time. A parallel component can be loaded in the platform by providing a *.jar* file that aggregates Java code for the component's threads, an XML deployment descriptor, and possibly additional documentation and data associated with to the component. During this loading operation, a new *Component* resource object is created in D-RAJE, and automatically registered with the resource manager. Once a component has been loaded, it can be considered

as a new resource in the cluster. Another component can therefore trigger its deployment and activation on the cluster. Alternatively, these operations can be triggered by any user of the platform through the abovementioned administration tool.

4 Implementation and experimentation

The Concerto platform is still under development. The overall architecture of the Concerto middleware on one host is depicted in figure 1. The main part of Concerto consists in portable Java code. A small part of D-Raje requires platform-specific native code to implement system resource monitoring. Presently, only Linux is supported but thanks to the design of D-Raje, porting to other systems should not be a difficult task. In the current prototype, the support for monitoring of some conceptual resources (*eg* threads, sockets) relies on a modified version of the Kaffe virtual machine and instrumented standard Java classes.

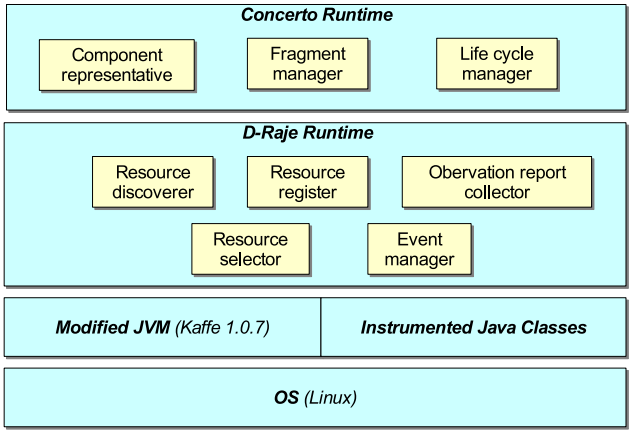

Figure 1: Overview of the Concerto platform (on one host)

Several experiments on distributed applications are currently carried out. We are developing for example a Concerto component that implements a parallel active router for wide area networks. This component can dynamically apply an appropriate service (filtering, compression, encryption,...) to each of the streams that passes through the router. The parallel router-component is deployed on a high-performance cluster in order to accept several concurrent streams and to sustain acceptable bandwidth. In this case, the use of Concerto eases developing multi-criteria adaptation strategies to perform load-balancing according to resource availability.

5 Related Work

The originality of the Concerto approach lies in the fact that it combines a parallel component model together with some support for distributed resource observation. Component-based application development is already possible through the use of ”standard” component technologies, such as Microsoft COM [11] or Sun’s Enterprise Java Beans. The OMG (Object Management Group) also developed its own solution with the CORBA Component

Model [12]. However these technologies were not specifically designed to support parallel components, that is, components that involve parallel activities. Few works have been carried out on models or platforms that target parallel and distributed components. One of them is the Common Component Architecture [1], which defines a component model suited for parallel scientific applications. The main objective in CCA is to allow the efficient interoperability of pre-existing scientific codes. Like in Concerto, few constraints are put on the way the functional part of a parallel component is implemented. However, no support is provided for component adaptation. Project Padico [6] bears similarities with CCA as it is dedicated to coupling scientific codes. It extends the Corba Component Model, and relies on a specific communication toolkit in order to provide efficient communication between several SPMD parallel codes encapsulated in components. Some form of adaptation is provided in the communication layer that allows for several commonly used communication libraries.

The work described in [2] is also focused on parallel and distributed components: an implementation of the Fractal [3] hierarchical component model is proposed in order to provide so-called Grid components. This implementation is based on the Pro-Active library [4]. It provides facilities for asynchronous communication, migration of activities, deployment and debugging. Like in Concerto, a parallel component may be implemented according to the MIMD model. Introspection on both the components structure and placement is possible, though both kinds of introspection rely on disctinct mechanisms.

None of the abovementioned works proposes a homogeneous way to get precise information about the environment of a component (including the system), and about its subcomponents.

6 Conclusion

In this paper we have presented the Concerto platform, which aims at allowing the deployment and the support of parallel components on clusters of workstations. Ongoing work focuses on proposing a basic parallel component model, as well as tools for the deployment of such components. In a first stage, our objective is to adopt as simple and flexible a model as possible.

The clusters targeted by the Concerto platform are primarily those composed by enlisting non-dedicated, heterogeneous, and possibly unstable workstations. The runtime environment offered to parallel components is therefore liable to exhibit much heterogeneity and dynamicity. For this reason, the Concerto platform implements facilities for discovering how resources are distributed in the cluster, and for monitoring these resources at runtime.

The development of the Concerto platform is still in progress. Moreover a number of parallel components are also being developed. In the near future we plan to use the platform for deploying instances of these components on different kinds of cluster platforms.

Aknowledgments

This work is supported by the French Ministry of Research in the framework of the ACI GRID programme.

References

[1] R. Amstrong, D. Gannon, A. Geist, K. Keahey, S. Kohn, L. McInnes, S. Parker, and B. Smolinski. Towards a Common Component Architecture for High-Performance Scientific Computing. In *8th Int. Symposium on High-Performance Computing*, Redondo Beach, California, August 1999.

[2] F. Baude, D. Caromel, and M. Morel. From Distributed Objects to Hierarchical Grid Components. In *Proc. of International Symposium on Distributed Objects and Applications (DOA'2003)*, LNCS, Catania, Sicily, November 2003.

[3] E. Bruneton, T. Coupaye, and J.-B. Stefani. Recursive and dynamic software composition with sharing. In *Proc. of the 7th ECOOP International Workshop on Component-Oriented Programming (WCOP'02)*, Malaga, Spain, June 2002.

[4] D. Caromel, W. Klauser, and J. Vayssiere. Towards Seamless Computing and Metacomputing in Java. *Concurrency Practice and Experience*, 10(11–13), November 1998.

[5] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *10th IEEE Int. Symposium on High-Performance Distributed Computing*. IEEE Press, August 2001.

[6] A. Denis, C. Pérez, T. Priol, and A. Ribes. Padico: A Component-Based Software Infrastructure for Grid Computing. In *17th International Parallel and Distributed Processing Symposium (IPDPS'2003)*, Nice, France, April 2003. IEEE Computer Society.

[7] DMTF. CIM specification v2.2. Technical Report DSP0004, Data Management Task Force, http://www.dmtf.org 1999.

[8] F. Guidec, Y. Mahéo, and L. Courtrai. A Java Middleware Platform for Resource-Aware Distributed Applications. In *2nd Int. Symposium on Parallel and Distributed Computing*, Ljubljana, Slovenia, October 2003.

[9] F. Kon, R. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *9th IEEE Int. Symposium on High Performance Distributed Computing*, Pittsburgh, USA, August 2000.

[10] K. Krauter, R. Buyya, and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Sofware – Practice and Experience*, 32(2):135–164, February 2002.

[11] Microsoft. The Component Object Model Specification. Technical report, Microsoft Corporation, October 1995.

[12] OMG. CORBA Components. TC Document OMG-orbos-99-07-01, OMG, July 1999.

[13] F. Sacerdoti, M. Katz, M. Massie, and D. Culler. Wide Area Cluster Monitoring with Ganglia. In *Proc. of Int. Conference on Cluster Computing*, Hong Kong, December 2003.