Ubiquitous Fractal Components

Didier Hoareau, Yves Mahéo

Ubiquitous Fractal Components

Didier Hoareau and Yves Mahéo

VALORIA Laboratory
University of South Brittany, France
{Didier.Hoareau|Yves.Maheo}@univ-ubs.fr

1 Introduction

The relationship between computers and computer users has evolved from *several users for one computer* to *several computers for one user*. Interaction is now possible with pervasive devices ranging from PDA and mobile phones to powerful workstations. However, unlike the hardware environment the user can be faced with, current software elements cannot be qualified as ubiquitous yet. An application should be said ubiquitous if its services are available everywhere (or at least where it makes sense). However traditional solutions consisting in installing the whole application on each device or relying systematically on a client-server approach are hardly applicable to the kind of environment considered. Many resource-constrained devices cannot host the entire application and the absence of permanent connectivity between the devices makes the use of distant servers hazardous.

The overall objective of our work in project Cubik [1] is to provide middleware support for the deployment and the execution of component-based ubiquitous applications. In our approach, for a given set M of machines composed of mobile and resource constrained devices, we want to put at disposal all the functionalities of a component-based application, functionalities that can be accessed from every machine of M. Moreover the devices that compose the network are characterised by their heterogeneity, their mobility and their volatility.

In this article we will focus on one part of project Cubik, through the presentation of a distribution scheme for Fractal-based applications that is namely based on a user-transparent extension to the Fractal component model. This scheme allows any Fractal application to expose it provided services in an ubiquitous way on several machines. The mechanisms implemented support network disconnections and let the application run in a degraded mode if necessary (e.g. parts of the application are still available whereas others parts are unaccessible). With our extension, Fractal components are adaptive with respect to network disconnections and reconnections and take advantage of the available connectivity of the network in order to maximise the number of provided interfaces that can be used.

2 Distribution of a Fractal Component: Towards Ubiquitous Components

In our approach, making a Fractal component ubiquitous is to make its provided interfaces available from several hosts. We propose a distribution scheme that is

based on the replication of composite components. Indeed, we allow a composite to be duplicated on a set of hosts, although each primitive component is localised on a single host. We do not elaborate here on the process that decides on the placement of each component (see [3] for more details). The general principle of our distribution model is the following: a composite component c is distributed over a set of hosts H if it exists on every host of H an instance of c. All the instances of c are created according to the directives found in the architecture descriptor (specified in an augmented FractalADL). At execution time, each instance of c maintains locally the configuration of its subcomponents. The provided and required interfaces of c are accessible on all the hosts of H. These interfaces are those defined in the architecture descriptor.

A primitive component is not replicated. So the instances of a composite containing a primitive component hold a distant reference to the single instance of this primitive (except for the composite instance located on the same host as the primitive instance, for which this reference is local). A composite c distributed over H may contain a composite subcomponent c' distributed over H' (H' must then be a subset of H). In this case, on a host that is in H but not in H', the local instance of c holds a distant reference to any one distant instance of c'.

The different replicas of a composite component are created during the deployment process. The addition of a new controller, named `cubik-controller`, to any Fractal component allows us to identify these replicas. When a composite component is instantiated locally, its `cubik-controller` is set with the placement information of its direct subcomponents. Our current implementation is based on Julia/ ASM2.0 and remote communication between components is achieved thanks to FractalRMI. Each proxy component that represents a remote composite or primitive component is a Fractal component that is created on the fly (thanks to the ASM library) and is bound to other local components as a normal Fractal component. Distribution is thus transparent.

3 Management of Network Disconnections

3.1 Active interfaces

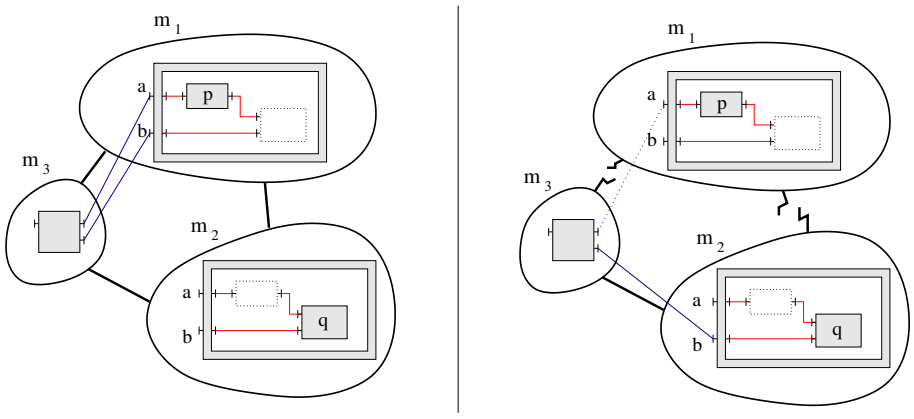
Because of the dynamism of the network a machine may become unaccessible together with the primitive components which were running on this machine. Since all provided interfaces of a replicated composite component are still accessible according to our distribution scheme, a method invocation on an interface that is bound to tan unaccessible remote primitive component will throw a network exception. In order to prevent such a situation we introduce the notion of *active interfaces*. If a remote component becomes (un)accessible, all interfaces that lead to the provided interfaces of this component are (de)activated. For example on Figure 1, if m_1 becomes unaccessible from m_2, interface a on m_2 is deactivated and method invocations on this interface are blocked. On m_2, only the provided interfaces of primitive component q and interface b are still active.

3.2 Automatic activation/deactivation of interfaces

Active interfaces are implemented within the `cubik-controller` thanks to the *MetaCodeGenerator* interceptor of Julia: method calls on each interfaces are reified and, according to the state of the interface, a specific adaptation policy is applied (e.g. wait until the interface becomes active again). When the state of an interface changes, the `cubik-controller` of the corresponding component propagates this information to the `cubik-controller` of its super component and by ricochet to all the components, following the dependencies between required and provided interfaces. Programmers can use the `cubik-controller` in order to discover the state of any interface so as to implement specific behaviours that take into account changes in interfaces' states. An obvious adaptation is to avoid invoking a method of an inactive interface.

We provide also a generic framework in order to give more possibilities in writing adaptation code (e.g. when a required interface becomes inactive and instead of directly propagate this state through the hierarchy, we may want to cache the method call during a parameterisable amount of time). Indeed, the proxy component that is generated on the fly is a composite component which encloses the proxy object and an adapter component. The adapter has the same provided interfaces than the remote component and can be used in order to write pre and post adaptation code (cf `SimpleCodeGenerator` class in Julia).

Network disconnections have been taken into account thanks to D-Raje (Distributed Resource-Aware Java Environment) [4], an extensible Java-based middleware developed in our team. D-Raje makes it possible to model and to monitor hardware resources (e.g. processor, memory) and software resources (e.g. process, socket, thread) in a distributed environment. D-Raje has been extended by adding two new types of resources: *RemoteBinding* and *NetworkLink*. A *NetworkLink* re-



(a) Fully connected network

(b) Network with two disconnections

Figure 1. Reconfiguration of bindings according to the state of the interfaces

sources models the physical link between two hosts and reifies the state of this connection. A *RemoteBinding* resource listens to a *NetworkLink* in order to maintain the state of a binding between two remote components: when a disconnection occurs, the corresponding *RemoteBinding* resources are notified and the Fractal component interfaces are deactivated using the `cubik-controller`.

3.3 Automatic reconfiguration of bindings

The use of active interfaces allow Fractal applications to perform in a degraded mode even if some parts of the application are unaccessible. When this happens, in order to maximize the number of active interfaces, it is possible to define an automatic reconfiguration of the bindings that takes advantage of the network connectivity. Indeed, in our approach, a composite component is replicated on several machines in order to increase the availability of the application. When a component is bound to a composite component on a specific host, it is possible to choose any other replica of this component according to its accessibility.

For example Figure 1 shows how bindings between remote components are reconfigured according to the state of the remote interfaces: on part (a), components p and q are available from m_1 (that is, the proxy to q on m_1 is a valid reference to m_2) and on part (b), m_2 is no longer accessible from m_1. We can notice that the composite component on m_2 performs in a degraded mode: only a subset of its subcomponents is running.

4 Conclusion

This paper has presented a method to make Fractal-based application ubiquitous, that is, to allow the services implemented by the application to be invoked on more than one host. This is mainly achieved thanks to a distribution scheme of composite components that can be duplicated on several hosts. The introduction of *active* interfaces to the model allows the application to perform in a degraded mode when disconnections occur. Some simple mechanisms have also been described to show how ubiquitous Fractal components can be tuned regarding network disconnections and how bindings are automatically reconfigured when network failures occur.

References

1. Project Cubik web pages. http://www-valoria.univ-ubs.fr/CASA/Cubik.
2. Didier Hoareau and Yves Mahéo. Distribution of a Hierarchical Component in a Non-Connected Environment. In *31st Euromicro Conference - Component-Based Software Engineering Track*, pages 143–150, Porto, Portugal, September 2005. IEEE CS.
3. Didier Hoareau and Yves Mahéo. Constraint-Based Deployment of Distributed Components in a Dynamic Network. In *Architecture of Computing Systems (ARCS 2006)*, volume 3864 of *LNCS*, pages 450–464, Frankfurt/Main, Germany, March 2006. Springer Verlag.
4. Yves Mahéo, Frédéric Guidec, and Luc Courtrai. A Java Middleware Platform for Resource-Aware Distributed Applications. In *Proceedings of the 2nd International Symposium on Parallel and Distributed Computing (ISPDC'2003)*, pages 96–103, Ljubljana, Slovénie, October 2003. IEEE CS.