Propagative Deployment of Hierarchical Components in a Dynamic Network

Didier Hoareau, Yves Mahéo

Propagative Deployment of Hierarchical Components in a Dynamic Network

Didier Hoareau and Yves Mahéo

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

Abstract. This paper addresses the distribution and the deployment of hierarchical components on heterogeneous dynamic networks. Such networks may include fixed and mobile resource-constrained devices and are characterized by the volatility of their hosts and connections, which may lead to their fragmentation. We propose a propagative, hierarchically-controlled deployment process for such networks and an ADL extension allowing the specification of this context-aware deployment.

1 Introduction

The component-based approach becomes widely reckoned to be relevant for developing complex distributed applications and many component models and their associated technologies are now available. Some of the proposed models (*e.g.* Koala [11], Darwin [6] or Sofa [9]), known as hierarchical models, wake up the interest of software architects. In such models, a component –that is then called a composite component– can be itself an assembly of components, recursive inclusion ending with primitive components that encapsulate computing code.

Besides, the distributed platforms that are susceptible of being the target of complex distributed applications, have evolved in a few years from homogeneous networks of workstations to networks of heterogeneous hosts that may comprise mobile and resource-constrained devices. Among these platforms, dynamic networks represent common but challenging environments. What we call a dynamic network is a network that is characterised by its heterogeneity (e.g. hosts do not all provide similar hardware and software resources), and its dynamism (e.g. hosts may become unaccessible because of their mobility or their volatility). A major consequence of this dynamism is that the target platform cannot be considered as a fully connected network. It is rather described as a partitioned network, viewed as a collection of independent islands. An island is equivalent to a connected graph of hosts that can communicate together, while no communication is possible between two islands. In addition, the configuration of the islands may change dynamically.

This paper describes a distribution scheme of hierarchical components and its associated deployment process that targets the abovementioned dynamic networks. Because of the very constrained environment in which the application is to be deployed, we can hardly envisage a permanent access to the services offered by the application or an optimal use of the resources. The emphasis is put on finding a distribution scheme and

some deployment mechanisms (focusing on the instantiation and the activation phases) that achieve a minimal availability while taking account of the environment.

2 Distributed Hierarchical Component Model for Dynamic Networks

In order to support network disconnections we propose a distributed hierarchical component model which allows an application to run in a degraded mode, avoiding that the entire application becomes unusable. We introduce the notion of *active interface* to the component model. Our runtime support detects network disconnections and deactivates some components' interfaces accordingly. The underlying distribution scheme of the model is based on the replication of composite components. This replication allows the interfaces of a composite to be easily accessible on a set of hosts. Only the membrane, that contains architectural information, is replicated, thus reducing consistency maintenance problems. Each primitive component is localized on a single host, which reflects the semantics of the architecture descriptor in which each reference to a component corresponds one (possibly statefull) component. Further details about the distribution and the support of this distributed hierarchical component model can be found in [5].

3 Context-Aware Deployment Specification

When considering the deployment of distributed components, the key issue is to build a mapping between the component instances and the hosts of the target platform. This task implies to have some knowledge not only about the identity of the hosts involved in the deployment phase, but about the characteristics of each of them as well. However, at design-time, the designer is unlikely to know where to deploy each component regarding resource availability. This motivates the need to differ this task at runtime. We propose to add a deployment aspect to an existing architecture description language (such as [2, 3]).This will allow the description of the resource properties that must be satisfied by a machine for hosting a specific component.

We follow the approach of [4] to specify the deployment of the hierarchy of components in a constraint-based declarative way (see figure 1). The architecture descriptors of the components are augmented with deployment descriptors in which constraints on the resources required by components and on their possible locations can be specified. It is not mandatory to give explicit names or addresses to target machines: the placement of components are mainly driven by constraints on the resources the target host(s) should satisfy. The choice of the machine that will host a component will be made automatically at runtime (during the deployment).

When the deployment is triggered, all the constraints listed in the deployment descriptor may not be satisfied immediately. The dynamism of the network makes the situation even more difficult as it may occur that the set of hosts that would satisfy globally the deployment constraints are never connected together at the same time, precluding any deployment.

```
<component name="DocumentSearch">
  <component name="DocumentFinder">
    <deploymentcontext>
      <locationconstraint>
        <target varname="x"/>
      </locationconstraint>
    </deploymentcontext>
  </component>
  <component name="DocumentBuffer">
    <deploymentcontext>
      <resourceconstraint>
        <memory free="200" unit="MB"
                operator="min" />
      </resourceconstraint>
      <locationconstraint>
        <target varname="y"/>
      </locationconstraint>
    </deploymentcontext>
  </component>
<deploymentcontext>
      <locationconstraint>
        <operator name="alldiff">
<arg varname="this.DocumentSearch.x"/>
<arg varname="this.DocumentBuffer.y"/>
        </operator
      </locationconstraint>
  <deploymentcontext>
</component>
```

Fig. 1. Deployment descriptor

4 Propagative Deployment

The deployment process we propose is a *propagative* one: it allows an application to be activated progressively, that is, part of its provided services can be put at disposal even if some machines that are required for the "not yet" installed components are not available. As soon as these machines become connected (or accessible) or some required resources appear (or become available), the deployment will go along. Thanks to our distributed hierarchical component model and the dynamic activation of interfaces, the application can run in a degraded manner even if some of its parts are not yet started.

The main issue of such a deployment is to ensure the unicity of the component instantiations imposed by the architecture descriptor. Indeed on one hand, since we cannot predict which machines will be connected at any time, we cannot select one to be responsible for the instantiation decisions of the entire application. On the other hand, if we let each machine make an instantiation decision, we cannot guarantee that in two different islands contradictory instantiations may not be performed.

Ensuring consistent instantiations comes down to establishing a distributed consensus across several islands. We use the results of [8] where the authors identify *conditions* for which there exists an asynchronous protocol that solves the consensus problem despite the occurrence of crashes. It is thus possible to elect a machine responsible for the instantiation of a component within an island composed of a *majority* of machines. When an applicant machine is elected and when an instantiation is made, the deployment descriptor is updated with this information. As in the work described in [10], the scalability of our proposition is ensured by the distributed and hierarchical organisation of the control: each composite component of the hierarchy is represented by a machine.

We propose to alleviate the risk that the consensus algorithm may not terminate (*e.g.* the number of hosts within an island may not be sufficient) by taking advantage of network changes to make the consensus evolve. We detect network changes (*e.g.* a machine is newly connected) and possibly react to these changes (*e.g.* make a newly connected machine participate to the consensus). Moreover, in order to avoid that a machine responsible for a composite component makes instantiation decisions in a non-majority island, a reelection mechanism is triggered after comparing the different versions of the deployment descriptors.

5 Conclusion

This paper has presented a support for deploying and executing an application built with hierarchical components on an heterogeneous and dynamic network. The main contri-

bution of this work is that it attempts to take into account a challenging distributed target platform characterized by the heterogeneity and the volatility of the hosts, volatility that may result in the fragmentation of the network.

The propagative deployment presented in this paper is based on a constraint-based language for the description of the placement of the components according to resource requirements. Our distributed component model has been implemented using Julia, a Java implementation of the Fractal component model [1]. The standard Fractal ADL has been extended thanks to the addition of new modules. We use D-Raje [7], a framework developed in our team, dedicated to the observation of distributed system resources in Java. We can thus detect network changes and exploit them in the deployment process.

The main direction of our future work consists in the extension of our propagative deployment in order to define an autonomic deployment in which decisions about the placement of components could be reconsidered.

References

[1] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. An Open Component Model and its Support in Java. In *Proc. of the Int. Symposium on Component-based Software Engineering*, Edinburgh, Scotland, May 2004.

[2] xAcme: Acme Extensions to xArch. School of Computer Science Web Site: http://www-2.cs.cmu.edu/ acme/pub/xAcme/, 2001.

[3] E. Dashofy, A. van der Hoek, and R. Taylor. An Infrastructure for the Rapid Development of XML-based Architecture Description Languages. In *Proceedings of the Int. Conference on Software Engineering*, pages 266–276, Orlando, Florida, USA, May 2002.

[4] A. Dearle, G. Kirby, and A. McCarthy. A framework for constraint-based deployment and autonomic management of distributed applications. In *Proc. of the Int. Conference on Autonomic Computing*, 2004.

[5] D. Hoareau and Y. Mahéo. Distribution of a Hierarchical Component in a Non-Connected Environment. In *Proc. of the 31th Euromicro Conference - Component-Based Software Engineering Track*, Porto, Portugal, September 2005.

[6] J. Magee, N. Dulay, S. Eisenbach, and J. Kramer. Specifying Distributed Software Architectures. In *Proc. of the 5th European Software Engineering Conference*, Sitges, Spain, September 1995.

[7] Y. Mahéo, F. Guidec, and L. Courtrai. A Java Middleware Platform for Resource-Aware Distributed Applications. In *2nd Int. Symposium on Parallel and Distributed Computing*, pages 96–103, Ljubljana, Slovenia, October 2003.

[8] A. Mostéfaoui, S. Rajsbaum, M. Raynal, and M. Roy. Condition-based consensus solvability: a hierarchy of conditions and efficient protocols. *Distributed Computing*, 17(1), 2004.

[9] F. Plasil, D. Balek, and R. Janecek. SOFA/DCUP: Architecture for Component Trading and Dynamic Updating. In *Proc. of the 4th Int. Conference on Configurable Distributed Systems*, Annapolis, Maryland, US, may 1998.

[10] V. Quéma, R. Balter, L. Bellissard, D. Féliot, A. Freyssinet, and S. Lacourte. Asynchronous, hierarchical and scalable deployment of component-based applications. In *Proc. of the 2nd Int. Working Conference on Component Deployment*, Edinburgh, Scotland, May 2004.

[11] R. C. van Ommering. Koala, a Component Model for Consumer Electronics Product Software. In *Proc. of the ESPRIT ARES Workshop*, Las Palmas de Gran Canaria, Spain, February 1998.