Adaptability in Component-Based Peer-to-Peer Applications

Sascha Alda
Institute for Applied Computer Science, University of Bonn
53117 Bonn, Germany
alda@bonn.edu, http://www.cs.uni-bonn.de/III

1. Introduction

One great challenge in the field of software engineering is to develop *reusable, adaptable* and *scalable* software systems. To address this goal, a multiplicity of approaches have been proposed. One trend one could ascertain has been the drift away from complex, monolithic applications towards to *distributed systems*. The predominant model for building distributed systems has been the *client-server paradigm* separating the system in one central server and possibly multiple clients, communicating with a server over a common network. Recently, a new abstraction has become popular in the area of distributed computing, indicated as the *peer-to-peer* paradigm. One crucial criterion for peer-to-peer is that, instead of strictly decomposing the system into clients (which consume services) and servers (which provide them), peers can elect to provide services as well as consume them. This *decentralised* approach seems in general more flexible to enhance new application and interaction scenarios with multiple servers.

Another way to build flexible systems is nowadays realised by *software components* [6]. Components can be seen as self-contained units of compositions, with contractually specified interfaces and explicit context dependencies. The deployment of components is handled by *component architectures*. A specification of how to construct a component is termed *component model*. So far, less efforts has been conducted to develop component models and appropriate component architectures for component-based peer-to-peer applications.

This paper presents our work towards the realisation of *a component model* as well as of an architecture serving as a *runtime environment for component-based, distributed peer-to-peer applications*. We further explain additional concepts for *adaptability* in peer-to-peer applications.

2. Component Model

We consider the FlexiBeans component model [4] as our approach of a model for component-based peer-to-peer applications. The FlexiBeans model is an extension of the JavaBeans model. Most notably, we have explicitly incorporated the RMI technology in the interfaces of the component skeletons, enabling them to interact across machine boundaries. For interaction between components we provide two primitives: *events*, allowing an unidirectional way of interaction and *shared objects*, providing bidirectional interaction between components.

3. Component Architecture EVOLVE

The EVOLVE architecture [4] has originally been developed to serve as a runtime environment for component-based, distributed client-server applications. These applications correspond to compositions consisting of the above presented FlexiBeans components. EVOLVE consists of a server runtime environment for server-sided components as well as (multiple) client runtime environments for client-sided components. All compositions are described in a language called CAT (Component Architecture for Tailorability) [3]. CAT defines which components of a given set of components constitute an application and which of them actually interact among each other. There is a CAT file for the server composition as well as for the client-sided composition. Another CAT file (DCAT) declares the remote interaction between client and server compositions.

Although components are yet capable to interact remotely over the network, clients are restricted to interact with the server only, in order to meet the client-server paradigm, which is imposed by the original platform. In our current work we are about to advance EVOLVE by introducing a more sophisticated environment, which we denote as the *peer runtime environment* [1]. This environment is able to deploy service-consuming as well as service-providing components at the same time. Besides, a peer runtime environment offers the following properties:

- Interaction with *more than one* peer.
- Dynamic *switch* to other peers during runtime.
- *Adaptability* of client and server composition
- *Definition of dependencies* between client and server compositions. Dependencies can be weighted, denoting the *relevance* of a certain dependency.
- *Advertisement* of component services
- *Accountability* and *billing* of component services
- Compatibility to client-server environment of EVOLVE (hybrid architecture)

Proceedings of the Second International Conference on Peer-to-Peer Computing (P2P'02)
0-7695-1810-9/02 $17.00 © 2002 **IEEE**
IEEE COMPUTER SOCIETY

A *hybrid* scenario of the EVOLVE platform encompassing all presented environments is depicted in fig. 1. In the centre of fig. 1, three peers are illustrated interacting among each other. Each peer is equipped with client as well as server components. A black line between client and server components indicates a dependency. Additionally, peer A serves as a server for the client depicted in the lower left corner. Peer B consumes services belonging to the EVOLVE server in the lower right corner. In turn, peer B is allowed to re-offer all data it has gained from the EVOLVE server to all peers, consuming services from peer B.

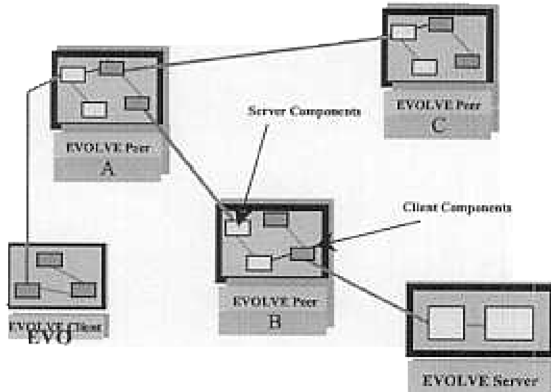

Fig. 1: Hybrid scenario in EVOLVE encompassing the new peer environment.

Currently, we are engaged to move our design principles of a peer environment towards to a concrete implementation. For our purposes, we use the JXTA [5] platform from Sun as the fundamental framework for a peer-to-peer environment.

A Tailoring API allows the adaptability of a given component composition during runtime. However, the process to adapt (i.e. to replace, to change or to delete) a component in a peer-to-peer application is not always a trivial case. As one can see in fig. 1, components can have *direct* or *transitive dependencies* on other components or on other parts of the underlying system. Consequently, before adapting a component, one must obey that no dependencies will be violated. The consideration and management of dependencies between components residing on a local machine has already been explored in some papers (e.g. [2]). For our intentions, a couple of additional aspects must be taken into consideration:

- The distribution of components
- Transitive dependencies of components
- Dynamic change of the peer combination over time
- Sophisticated strategies for adapting a component

In an extreme case, one can imagine two strategies for the process of adaptability: in a *conservative* case, a component is not allowed to be adapted until all dependencies have been resolved. Alternatively, in a *liberal* way, a component can be adapted instantly without considering the dependent components. Apparently, none of these procedures appear effective: the first option might lead to an indefinitely block while the latter one will obviously result in program crashes.

We therefore recommend alternative strategies to handle the adaptability process more accurately. In our approach, we propose a *pre-analysis* of the underlying peer-to-peer network. Such an analysis reveals all direct and transitive dependencies starting from a component belonging to a certain peer. All remote and local dependencies between components can be obtained through a Dependencies API. Based on such an analysis, the owner of a component is capable to make a decision, whether it is still beneficial to follow up the adaptability intentions of a certain component. One could also think of the possibility to impose an *adaptability policy* within the network. Such a policy could prescribe when the owner of a component is allowed to adapt a component that has dependencies to other components. Criteria for such a policy could be the number of depended components or the total relevance of all dependencies starting from a component.

A conceivable algorithm could be based on *graph-based theories*. By scanning a peer network starting from a specific peer, a *graph* is build consisting of *nodes* (denoting components) and *edges* (denoting dependencies). Edges describe both remote interactions between components and dependencies between components on the same peer. The latter edge consists of attributes describing the relevance of an dependency. All strategies to analyse a peer network can be mapped on basic graph operations. Apparently, the algorithm must take account of the dynamic composition of a peer network, since peers can join and leave a peer network.

References

[1] Alda, S., Radetzki, U., Bergmann, A., Cremers, A.B., "A component-based and adaptable Platform for networked Cooperations in Civil and Building Engineering", in: *Proceedings of the 9th Int'l Conf. on Computing in Civil and Building Engineering (ICCCBE-IX)*. Taipei, Taiwan, 2002.

[2] Kon, F., Campbell, R.H., "Dependence Management in Component-Based Distributed Systems", in: IEEE Concurrency January-March 2000, No. 1, Vol. 8.

[3] Magee, J., Dulay, N., Eisenbach, S., Kramer, J., "Specifying Distributed Software Architectures", in: *Proceedings of the 5th European Software Engineering Conference*, Barcelona, LNCS 989 (Springer-Verlag), pp. 137-153, 1995.

[4] Stiemerling, O., Hinken, R., und Cremers, A.B.: "The EVOLVE Tailoring Platform: Supporting the Evolution of Component-Based Groupware", in: *Proceedings of EDOC '99 Mannheim, Germany, IEEE Press*, pp. 106-115, 1999.

[5] Sun Microsystems: *JXTA v1.0 Protocols Specification, Revision 1.1.1, June 12th, 2001.* http://spec.jxta.org/v1.0/

[6] Szyperski, C.: *Component Software - Beyond object oriented programming*, Addison-Wesley, 1997.

0-7695-1810-9/02 $17.00 © 2002 IEEE

IEEE COMPUTER SOCIETY