Dynamical Vertical Integration of Distributed Java Components Using an Architecture Model

Alexander Prack (ap@sernet.de)
Ulf Schreier (schreier@fh-furtwangen.de)

Abstract: A key idea of architecture is the description of components and their connections. This information can be extended to define the horizontal and vertical distribution of components. RemoteXParts is a framework that uses such a model and exploits it for dynamic deployment at runtime along the vertical client/server line. Its goal is the support of the POJO (Plain Old Java Object) programmer. It enables him to distribute application layers between different systems without concerning himself with actual distributed programming. He does not have to take care of tasks like executing remote calls, checking for version differences, and updating old classes or transferring new ones where needed. It is a minimal extension to Java, as small as possible. Additionally it does not need the generation of source code like in MDA approaches [OM03], although it is based on the UML2 component concepts.

1 Introduction

When faced with the task of bringing together different applications or data in an enterprise, a programmer today can choose between a multitude of competing — sometimes contradicting — client / server approaches. A distribution model can be based on concepts like web services [W303], Enterprise Java Beans [Mi03] or Microsoft's DCOM or .NET framework [Mi99] amongst others. The common denominator of these techniques is their compulsiveness: either in the regard of committing to certain types of interfaces of one particular solution without the possibility to change later, or because the distribution of the components has to be decided during installation, not dynamically during runtime.

However, ideally it would be possible to make vertical integration happen where the need for it arises. If a client has the capabilities to take over some of the server's tasks, it should be possible to harness these capabilities. Otherwise an increase from for example 15 to 15,000 clients leaves the additional load to be handled exclusively by the server. This should happen only for those clients that do not have the computing power or storage space to assume additional tasks themselves, such as PDAs or programmable cellular phones. For this reason, it is desirable to define vertical distribution of an application based on each individual client, and not generally for the whole client base.

RemoteXParts [Pr04] aims to be a solution for the pure POJO programmer who doesn't want to intertwine his application with a component model that adds its own complexity to the problem. This complexity is most likely the main reason why distributed compo-

nents are not commonly used to address the demands of a distributed architecture. Instead, programmers have turned to server-centric approaches. The server / thin client (browser) model that is being used by the majority of distributed applications with an HTML frontend eases installation and maintainability, but is accompanied by restrictions for the user interface and other disadvantages.

2 The Basic Component Model

An XParts visualisation is very close to a UML2 component diagram [OM02]. Fig. 1 shows an example application. Subsystems exist of other subsystems (or packaged components in the new UML2 terminology) and bound components (or basic components in UML2), implemented directly in Java. Interfaces can be delegated from outer to inner components. Connections between components can be explicitly modelled by references to neighbouring components (this notion will need slight changes in order to support the new UML2 port notation). The actual representation of an XParts application is based on an XML file.

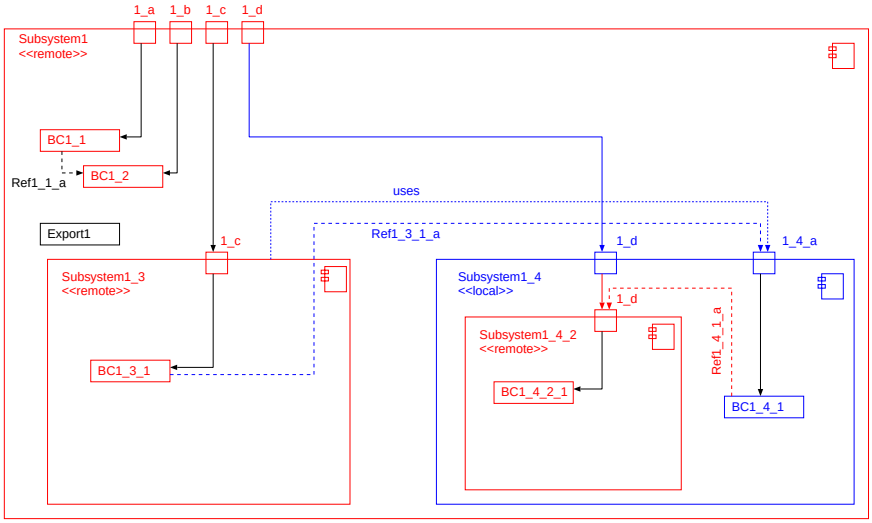

Figure 1: Example of XParts Components

The following code excerpt shows the XML representation of a subsystem.

```
<subsystem name="Subsystem1"
    access="remote" locality="free" dimension="singleton">
    <delegates refInterface="Ifc1_a" type="method"
        refPart="BoundComponent1_1"/>
    <consists-of refPart="BoundComponent1_1" />
    <export name="ExportClass1"/>
```

```
</subsystem>
```

The following code shows how a component instance according to the XML description is created and accessed. A factory takes care of instantiating the necessary objects and creating proxies where necessary to access the component as a whole. A component can be cast to any interface that it implements itself or delegates to a subcomponent:

```
XPartsFactory componentFactory =
new XPartsClientsideFactory(
"./build/classes",
"./build/classes/TestComponents.xml");
Subsystem subsystem1 = componentFactory.createSubsystem("Subsystem1");

Ifc1_a subsystemTyped = (Ifc1_a) subsystem1;
String returnValue = subsystemTyped.testMethod();
```

3 Distribution of Components

*Remote*XParts introduces the ability to define a server for XParts components. Components can be distributed from the server to multiple clients. The server authoritatively defines for each component whether it should run on the client or the server, or if this decision is left to each individual client (Attributes `access` and `locality` in the component definition shown above). Figure 1 shows `Subsystem1_4` (in blue) residing on the client, all others are running on the server. If a client does not have all components for a complete architecture, the server offers support for remote instantiation and invocation of the missing parts or migrating the class code to the client. Delegations from component instances on the server back to the client are possible, even nested components can be distributed between client and server.

RemoteXParts offers the ability to change the whereabouts of a component at any time for each individual client using a high level API. The methods of the `ApplicationState` interface allow querying the current location of a component – since it is otherwise completely transparent to the application programmer. Distributing an application becomes as simple as setting a property:

```
ApplicationState currentState = Architecture.getArchitecture();
currentState.setRemote("Subsystem1");
```

After the change, the subsystem is accessed in exactly the same way as in the previous example but all calls are remotely delegated to the implementing classes running on the server. The `Architecture` class is the runtime representation of the XML architecture description file. Any changes made by the application to its own model description are synchronised back to the XML file on the client. The server saves a master copy of an architecture description that remains untouched.

Although the communication of the XParts framework between client and server is handled using stateless session beans, all user defined XParts components on the server are

stateful. All objects and their fields are preserved during the lifetime of the client. Components are removed as soon as the client no longer holds a reference to them. The maintainability of distributed XParts applications is increased by XPart's ability to identify and resolve version differences between components on different systems. An application can be updated on all clients by simply replacing the affected components on the server.

XParts support can be added to any EJB application server by deploying the XParts EAR. The application programmer does not have to program a single EJB himself, nor does he have to make any RMI calls. The stateless session bean is exclusively used by the framework. All the application developer has to do, is to subclass the `AbstractBasicBoundCo` class. Apart from the constructor, no methods of the superclass have to be rewritten. The extended BoundComponent can implement all interfaces that the developer wishes the component to offer. In executing the interface methods he can use as many other objects derived from plain Java classes as he likes.

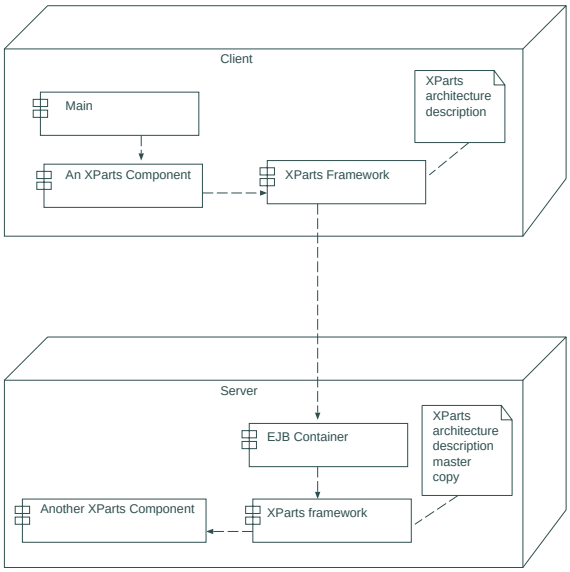

Figure 2: Deployment of XParts on client and server

Figure 2 shows how framework and user components are deployed on client and server. XParts components on the client are used by a regular Java application. If delegation of a call to a remote component residing on the server becomes necessary, the framework contacts a stateless session bean running in an EJB container, which in turn forwards the call to the correct component instance. This happens transparently for the application.

Calls from the server to the client (upcalls to components of a higher layer) are also possible and useful for tasks like notifying a client GUI about model changes on the server. Upcalls are also transparent to the programmer and implemented as a normal component delegation. They are realized in the framework using a separate connection to the server

originating on the client-side, enabling this functionality over any firewall configuration. The upcall mechanism utilises asynchronous events to notify one or more clients, not only about method calls to execute but also about updated classes or interfaces. The event stream can also be accessed by the application programmer directly if he wants to do so, enabling him to register multiple listeners to an event received by the XParts framework.

4 Summary

By using RemoteXParts together with other available free software it becomes possible to integrate existing Java application layers vertically across different systems by turning them into individual components. The goal is to achieve this with a minimum of additional programming, most of the existing Java classes can be reused without any changes. Once this is done, the existing interfaces can be called between different systems using automatically generated proxies that represent the defined application components and handle all necessary remote calls and further issues caused by distribution, such as resolving version differences.

An XParts description can be exported by a UML modelling tool supporting the XMI format. The file is parsed by the XParts framework and a runtime representation of all components and their interactions is created. Calls to a remote component are handled transparently, migrating and running components between systems becomes possible. The migration of already instantiated components during runtime while they are being accessed is being worked on right now and will be finished in the near future.

To support horizontal integration, it would become necessary to have one EJB container for each component server and support the synchronisation of architecture descriptions between them. This will require further implementation to be done on the framework.

References

[Mi99] Microsoft: *The Component Object Model Specification*. http://www.microsoft.com/com/resources/comdocs.asp. 1999.

[Mi03] Microsystems, S.: *Enterprise Java Beans Specification Version 2.1*. http://java.sun.com/products/ejb/docs.html. 2003.

[OM02] OMG: *UML Profile for Enterprise Distributed Object Computing Specification Final Adopted Specification*. OMG. 2002.

[OM03] OMG: *OMG Model Driven Architecture*. http://www.omg.org/mda/. 2003.

[Pr04] Prack, A.: *Dynamisch verteilbare Java Anwendungskomponenten*. http://webuser.fh-furtwangen.de/~schreier/Diplomarbeit_AlexanderPrack.pdf. 2004.

[Sc02] Schreier, U.: *XParts*. http://webuser.fh-furtwangen.de/~schreier/xparts.html. 2002.

[W303] W3C: *Web Services Architecture*. http://www.w3.org/TR/ws-arch/. 2003.