Assemblies of Local and Remote Components in the CORBA Component Model

Egon Teiniker[1,2], Stefan Mitterdorfer[1,2], Leif M. Johnson[2],
Christian Kreiner[1,2], Zsolt Kovács[2], Reinhold Weiss[1]

[1]Institute for Technical Informatics
Graz University of Technology, Austria
Inffeldgasse 16, A-8010 Graz
egon.teiniker@tugraz.at

[2]Salomon Automation GmbH
Friesachstrasse 15
Friesach bei Graz, Austria

The CORBA component model (CCM) defines a component architecture and a container framework in which the component life cycle takes place [2]. The CCM specification describes only remote components where all ports are accessible from CORBA clients [4].

In our Local Component Adapter Concept (LCAC), we separate application code from the implementation of the CORBA component logic [3]. For every IDL definition of a CORBA interface or component, we define a corresponding interface in the native implementation language. Adapter classes provide CORBA mappings, and link the implementation of business logic to the CCM component. By taking advantage of the adapter concept, we can implement local components without a CORBA shell. Using a local path for connecting components significantly reduces the communication overhead. Note that the decision between using the local or remote adapters does not affect the implementation of the business logic; in other words we can choose the remote accessibility of a component in a port by port manner at deployment time.

An important aspect of the LCAC is the separation between the component model and the implementation of components. The CCM component model is defined by two metamodels, *BaseIDL* and *ComponentIDL*. Dividing component models from component realizations leads to more flexibility in describing (IDL, UML, etc.) and using (local vs. remote) components. The business logic developer uses the local component logic and adapters convert the calls to the underlying remote technology in a transparent manner. So we can choose the best remote technology for the particular domain at deployment time.

Local components increase runtime performance, but they are not accessible from other processes. To overcome this problem, we can build assemblies of local and remote components using the *Session Facade* pattern [1]. If an instance of a session facade component is created, all connected local and remote components will automatically be instantiated and connected. From the client's point of view, the session facade is a fat remote component. In fact, it contains an assembly instance graph consisting of thin remote and local components that ensure easy reuse of business code.

To verify our concept, we implemented an application framework that uses local and remote Java components on the client side, while the server side is built from remote component assemblies implemented in C++. Our experiments show that the communication overhead of local Java and C++ components is not significant compared to plain class method calls. As expected, the costs of remote communication are significantly higher than for local access. These results support our hypothesis that the use of assemblies of local and remote CCM components allows us to effectively implement modular applications that can be accessed from remote clients.

References

[1] Floyd Marinescu. *EJB Design Patterns: Advanced Patterns, Processes and Idioms.* Wiley Computer Publishing, 2002.

[2] OMG. CORBA Components. Technical Report 02-06-65, Object Management Group, June, 2002.

[3] Egon Teiniker, Stefan Mitterdorfer, Christian Kreiner, Zsolt Kovács, and Reinhold Weiss. Local Components and Reuse of Legacy Code in the CORBA Component Model. In *EUROMICRO 2002, Dortmund, Germany, Sept. 4-6, 2002*, pages 4–9, 2002.

[4] N. Wang, D. Schmidt, and D. Levine. Optimizing the CORBA Component Model for High-performance and Real-time Applications, 2000.