Ada Binding to a Shared Object Layer

Johann Blieberger[1], Johann Klasek[1], and Eva Kühn[2]

[1] Institute of Computer-Aided Automation, Technical University Vienna, Treitlstr. 1/1831, A-1040 Vienna, Austria ({blieb,jk}@auto.tuwien.ac.at)
[2] Institute of Computer Languages, Technical University Vienna, Argentinierstr. 8, A-1040 Vienna, Austria (eva@complang.tuwien.ac.at)

Abstract. CORSO, a coordination system for virtual shared memory, allows bindings to different programming languages. Currently C, C++, Java, VisualBasic, and Oracle's Developer2000 are supported. We implement an Ada binding to CORSO, thus opening the area of virtual shared memory to the Ada world. Our Ada CORSO binding enhances Ada with transaction-oriented, fault-tolerant, distributed objects in a straight-forward way without having to extend the Ada language.

1 A Layered Approach

In distributed and heterogeneous environments some technique is desirable to shield the attributes of distributed objects like location, replication, representation and persistency from the programmer. Different approaches exist and the relation between them points to some kind of orthogonality. The most common pattern seem to be the message passing versus virtual shared memory (VSM) paradigm.

VSM neither intends to replace nor to exclude message passing architectures like CORBA or DCOM. In contrast, VSM should be seen as an additional layer providing enhanced mechanisms to the programmer.

Specifically in Ada's case, a binding to a VSM increases functionality and facilitates developing distributed applications, despite the fact that a variety of Ada built-in features and annexes in the Ada standard are available. The following issues are to mention:

- Communication, data sharing, and persistence has not to be implemented by means of standard Ada but can be put under coordination of a VSM system where several other languages and system architectures are glued together.
- The symmetric property of a VSM covers the actual needs of an application well. Particular subtasks can be implemented by the best-suited language, e.g. core development in Ada for safety critical parts and GUI development using Java.

The remaining paper is organised as follows: Section 2 overviews shared object paradigms. Section 3 presents concepts of CORSO, a virtual shared object layer developed at the *Institute of Computer Languages* at the *Technical University Vienna* and now made available commercially by *Tecco Coordination*

M. González Harbour and J.A. de la Puente (Eds.): Ada-Europe'99, LNCS 1622, pp. 263–274, 1999.

Systems, Vienna. Technical aspects of CORSO are revealed in Section 4. Our Ada binding to CORSO is described in Section 5. Pros and cons of our binding can be found in Section 6 where we also compare our binding to other language bindings and conclude the paper.

There have been other implementations of VSM in Ada, namely of the Linda tuple space (cf. [5,6,7]). All these implementations are stand-alone Ada implementations which lack the multi-language support of CORSO. In addition, the features offered by Linda are only a subset of CORSO's functionality.

2 Shared Objects

For the communication and synchronisation of distributed systems there exist two paradigms: *Message Passing* versus a *Virtual Shared Memory.*

2.1 Message Passing

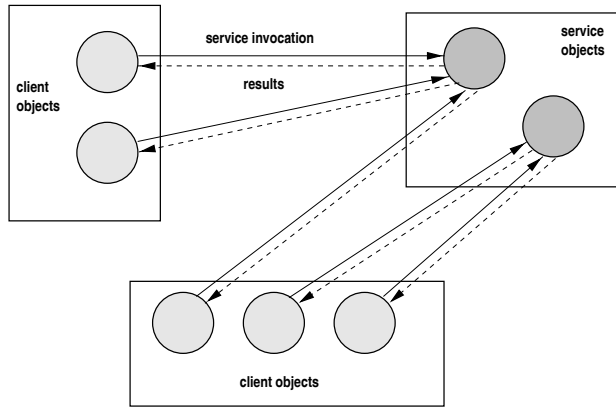

Fig. 1. Message Passing Paradigm

The classical and commercially wide-spread approach is the message passing paradigm. Processes communicate through the explicit (a-)synchronous sending and receiving of messages. The remote procedure call (RPC) is a two-way communication. The highest abstraction of the message passing paradigm are *distributed object* systems like CORBA and DCOM, where object methods can be invoked at remote sites which also results in a two-way communication pattern. Ada's Distributed Systems Annex (see [4]) also favours message passing. Application programs need to be aware of message sending or remote service invocation which means that e.g. fault tolerance has to be implemented explicitely into each distributed application. Adding and/or removing sites from the network of the distributed application has also to be considered explicitely

Moreover, such systems do not cache data fields locally which makes data field access expensive, because it requires an expensive remote procedure call.