Daniel Wengelin
Swedish Defence Research Institute
S-102 54 STOCKHOLM, Sweden

Mats Carlsson Göthe, Lars Asplund
Uppsala Unversity
S-751 21 UPPSALA, Sweden

Abstract Using a source code transformation approach to Ada in a distributed environment will give some implementation difficulties. This paper presents an all Ada, portable, solution to the problem of suspending a caller on one node during a call to a remote node. The solution is based on two sets of tasks on each node, making it possible for a caller to hang on an entry during the call. Algorithms are presented in pseudo-Ada.

1 Introduction

During recent years, much effort has been put into the area of Ada on distributed targets. Several papers [Cor84, AMN88, CWA89, BAP87] focus on the possibility to use standard compilers. This can be accomplished by means of a preprocessor, that translates one Ada program into a set of Ada programs. The transformation is controlled by some partitioning information.

One difficulty observed [EJK89] in the transformation is how to avoid the use of busy waiting during calls to a remote node.

2 A system structure to avoid busy wait

Consider the following example.

The hardware consists of a two node network. The program to be run is basically a task on one node calling a procedure on the other node. The task is to be suspended during the execution of the procedure.

The transformation of the source code will include the adding of code to handle the distribution and interface the network. We assume that there is a package DISTRIBUTED_MAIL dealing with the interface. The package will declare a task type, MESSAGE_HANDLER (M_H), and a resource pool to hold objects of the task type. The package will also declare a RECEIVE procedure and a SEND procedure. A generic package, declaring a RECEPTOR task type, will be used. A pool of RECEPTORs is also held on each node.

The MESSAGE_HANDLER algorithm is

```
loop
    – Get a call from application
    accept FORWARD (inparameters, addressee, mypointer)
    – Send message over the network
    SEND (inparameters, addressee)
    – Wait for a RECEPTOR to call on reply message to me
    accept REPLY (outparameters)
    – Accept final call from application
    accept READ_REPLY (outparameters)
end loop
```

The RECEPTOR is implemented as

```
loop
    – Queue on network port for (request) message
    RECEIVE (inparameters, addressee, frompointer)
    – Perform actual call
    FORWARD_TO (inparameters, addressee, outparameters)
    – Put reply message back on network
    SEND (outparameters, frompointer)
end loop
```

The RECEIVE procedure gets a message from the network. It is implemented to return control to the RECEPTOR only when a request from another node arrives. If the message received is a reply to some earlier call from a message handler on the local node, this message handler will be called. The following statements are found in the RECEIVE procedure.

```
GET_REQUEST : loop
     GET_FROM_NETWORK (message)
     if message.IS_A_REPLYthen
          message.FROM_HANDLER.REPLY (message.OUTPARAMETERS)
     else
          exit GET_REQUEST
     end if
end loop GET_REQUEST
```

The transformation of the original program will include substituting the procedure body on the calling node. The stub will perform the following algorithm.

```
GET_A_MESSAGE_HANDLER (apointer)
PACK_INPARAMETERS (..., inparameters)
apointer.FORWARD (inparameters, addressee, apointer)
apointer.READ_REPLY (outparameters)
UNPACK_OUTPARAMETERS (outparameters,...)
```

The RECEPTOR task will be used as follows. On each node, a package DISTRIBUTE is added during the transformation. The package specification is empty, but the body includes several vital components. First, a procedure TO, which will take a call, identify the addressee, unpack the inparameters, perform the call, and pack and return the outparameters. Second, an instantiation of the generic RECEPTOR package, using the TO procedure as the generic actual to the FORWARD_TO procedure. Finally, the package includes a resource pool for objects of the RECEPTOR task type.

The structure of a remote procedure call can be seen in fig. 1. The numbers denote the data flow sequence . Also in the figure are numbers indicating the order in which the M_H accepts rendezvous and the RECEPTOR makes its calls.

During a call, the following happens; the task will make an ordinary procedure call(1), executing the substituted procedure body. A M_H task will be obtained, and the actual parameters of the call will be transferred by a rendezvous(2). The calling task will then suspend itself, by means of a call to the READ_REPLY entry denoted "3". Meanwhile, the M_H passes the call onto the other node(3,4), where the call is caught by a waiting receptor(5,6). The receptor recognizes the call and performs the actual call through the FORWARD_TO procedure(7). At return(8), a reply message will be created and sent(9) back to the requesting node. There, another receptor will pick up the message through a call to RECEIVE. In RECEIVE, the message is recognized as a reply(11). Hence, the M_H pointer is extracted and the waiting M_H is called(12). This will cause the release of the M_H, the acceptance of the READ_REPLY entry(13), and hence, the release of the application.

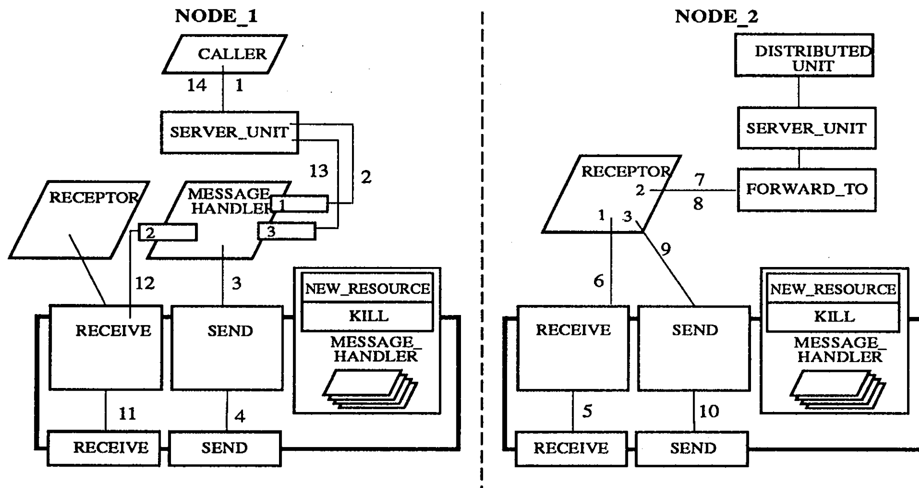

Figure 1. Data and control structure in a remote call in DARTS

3 Conclusions

There is a not very complex, all Ada, portable, way of having callers on one node suspended while their request is processed on a remote node. It has been tested and proven feasible, and is now undergoing further refinement and testing.

4 References

[Cor84] D Cornhill
Four approaches to partitioning of Ada programs for execution on distributed targets
IEEE Computer Society Conference on Ada Applications and Environments

[BAP87] J M Bishop, S R Adams, D J Pritchard
Distributing Concurrent Ada Programs by Source Code Translation
Software- Practice and Experience, Vol 17(12), 859-884 (Dec-87)

[AMN88] C Atkinsson, T Moreton, A Natali
Ada for Distributed Environments
Cambridge University Press, 1988

[CWA89] M Carlsson, D Wengelin, L Asplund
The Distributed Ada Run-Time System, DARTS
Uppsala University Institute of Physics Report, UUIP-1213
Uppsala University, Institute of Physics, P.O.Box 530
S-751 21 Uppsala, Sweden

[EJK89] G Eisenhauer, R Jha, J M Kamrad II
Targeting a Traditional Compiler to a Distributed Environment
Ada Letters, Vol IX(2), 45-51 (Mar/Apr-89)