Implementing the distributed features of Ada 9X with PVM

Yvon Kermarrec[1] and Laurent Pautet[2]

[1] Télécom Bretagne
Département Informatique
BP 832
F 29 285 Brest Cedex - France
Yvon.Kermarrec@enst-bretagne.fr
[2] Télécom Paris
Département Informatique - C220
46, rue Barrault
F 75 634 Paris cedex 13 - France
pautet@inf.enst.fr

Abstract. This extended abstract describes our experience in implementing the distributed features of Ada 9X in the GNAT compiler [7], a free Ada9X compiler which is part of the GCC compiler system. As members of the GNAT team, we have been working on the semantics of the distribution model for the language and have proposed a general implementation model. We have used PVM as a prototyping tool to validate our proposals as rapidly as possible [5, 6].

1 Introduction

The key issue with distributed systems is programming a distributed application. Bal et al. survey various approaches and solutions [2]. The original design of Ada [1] did not address distributed features, and as a result numerous non-standard language extensions have appeared over the years, leading to an inevitable loss of portability. Ada 9X [4], the new revision of the language, proposes a scheme for distributed system programming based on the "remote procedure call" paradigm and the shared variables paradigm. A limited number of new features has been introduced to support distribution, but the design preserves the strong typing model of Ada even when distributed code sections (partitions) communicate. The distribution of a program over a network of nodes is itelf not specified in the language, and is left to environment tools, so there is no language syntax that is specific to distribution.

The first section of the paper summarizes the model of Ada 9X for distributed systems. The second section presents a few PVM primitives that we have used in our implementation. We conclude with a discussion of some difficulties we have faced in mapping the semantics of Ada partitions into the PVM model.

2 The Ada 9X model

An Ada 9X program, whether distributed or not, consists of a set of compilation units : packages and subprograms. If a compilation unit is intended to play a role in a distributed system, it is marked by means of categorization pragma, which forces the compiler to generate additional communication code for that unit. Only two categorization pragmas are relevant to our discussion (the language specifies several others):

- **Shared_Passive** which supports the shared variable paradigm, whose programming advantages are well known. A shared passive package is intended to be configured on a storage element that is accessible from other partitions.
- **Remote_Control_Interface** (RCI) which supports the remote procedure call paradigm. Such a package provides services to other partitions by specifying subprograms that can be called remotely.

Distribution of a program is a post-compilation activity: some tool must be available, that provides ways of associating units with nodes (real or virtual) in a network of processors. All details of communication are invisible to the Ada programmer.
Nevertheless, Ada 9X defines a language defined interface (an Ada package, System.RPC) to the communication facilities. This approach increases portability and adaptability as this interface must be used by any conforming implementation.

3 GNAT and PVM

3.1 Why use PVM as a foundation

J. Dongarra in [3] wrote : "PVM can be used on its own or as foundation upon which other heterogeneous network software can be built".

We have selected PVM for several reasons. We indicate here only the major ones, in not particular order:

- PVM is mature technology and has been widely adopted well beyond the boundaries of the computer science community. It benefits from a great simplicity of concepts, an efficient realization and a useful set of abstractions.
- All the low-level details of communication can be completely ignored by the user : e.g., communication protocol, group handling, Unix process management.
- Debugging facilities do exist and we know their importance to the programmer. In particular, PVM can generate extremely useful traces of program execution.
- Tools exist (e.g., Xpvm and Xab) that can monitor the status of processing elements and give useful feedback to the programmer during trace analysis.

3.2 Configuration of the virtual machine

Among the host of PVM primitives, **Pvm_Addhosts** and **Pvm_Delhosts** are of particular interest in our context because they are used to describe the distributed system. Note that PVM offers facilities that are beyond our requirements, such as allowing the mapping of heterogeneous machines into the same virtual machine. It is important to note that these primitives are called from a running program and therefore support the programming of fault tolerance and dynamic reconfiguration in an active application.

3.3 Launching a new Ada 9x partition

The pvm_spawn primitive is used to start the execution of partitions on specified processing elements of the virtual machine. We assume that for each partition there is an associated executable file. This primitive provides a natural means of expressing the mater /worker paradigm, and also provides a way of introducing some measure of dynamic reconfiguration

3.4 Communication

The communication primitives have been used to implement the RPC model of Ada 9X. As we shall consider later, we have used only the non blocking communication operations.

3.5 Group management

Group primitives are used in an unusual way in our implementation. The idea is to use the name server and discard the notion of group itself. In fact, the groups we manage are composed of only one element. For each RCI package, the partition where the package body resides joins a new group and the name of the group is set to be the name of the RCI package (which is unique in the distributed Ada application). The same partition can therefore belongs to many groups.

When it needs to make a remote call, the client uses the broadcast facility and thus knows that its message will reach the right server. This feature presents obvious benefits because no localization of the servers is needed. In fact, this approach delegates the work fully to the group deamon of PVM.

4 Architecture of our system

The implementation of the distributed features of Ada 9X requires software developments at two distinct levels :

- The front-end level of GCC-GNAT which verifies the semantics rules linked to the distribution. This code is being programmed by the NYU team.

- The communication level and that is the object of this paper.

Our design criteria for the communication issues are clear : modularity and minimization of the impact to the existing GNAT system. Therefore, we have selected a software engineering approach based on layered software components and clear interfaces. Figure 1 illustrates the architecture of the complete system in terms of software components. GNULLI and GNARLI form the Ada run time and they also conform with modularity and adaptability ; System.RPC is the application -level interface to the inter-partitions communication facilities ; and GARLIC (GNU Ada Reusable Library for Interpartition Communication) which is our low level interface to the network.

GARLIC is being specified and we have not yet completed its design.

As the reader can notice, the application has direct access to the PVM primitives. This presents a major interest as the PVM programmer can also benefit of the software engineering features of Ada. In this context, Ada 9x presents a powerful alternative to the C language. Also, the distributed Ada 9X application accesses PVM to get services that are beyond the scope of the Ada 9X language itself and are implementation dependent. Among these services, we can cite : configuring and reconfiguring the distributed machine and loading/starting distributed code.

5 Troubles at the semantic interface

We indicate here a few points which complicate the integration of the distribution annex of Ada 9X in GNAT with PVM.

- PVM is strongly biased toward C programs, a language that has no notion of tasks nor concurrency in any form. The Ada programmer must be aware that only one buffer is allocated to one PVM task, and therefore concurrent operation on the buffer (e.g., two concurrent tasks want to send data) is troublesome and can lead to abnormal behavior (message mixing, for example). The programmer has to implement some mutual exclusion mechanism here and elsewhere because the system does not.
- Both GNAT and PVM use signals, of course in an inconsistent manner. In GNAT signals are used to map interrupts into language-defined exceptions, and also as part of the task synchronization primitives. Therefore, signals intended by one (e.g. PVM) may be intercepted and lost to the other. As a specific consequence, it is not possible to use synchronous PVM communication nor the group synchronization operation. One way to provide group synchronization is to introduce busy waiting but we know well enough the drawbacks of this approach.
- PVM supports only the transmission of basic C types when preparing a message or extracting data from a buffer. By contrast, Ada9X has very rich family of type construction mechanisms, and the semantics of distribution

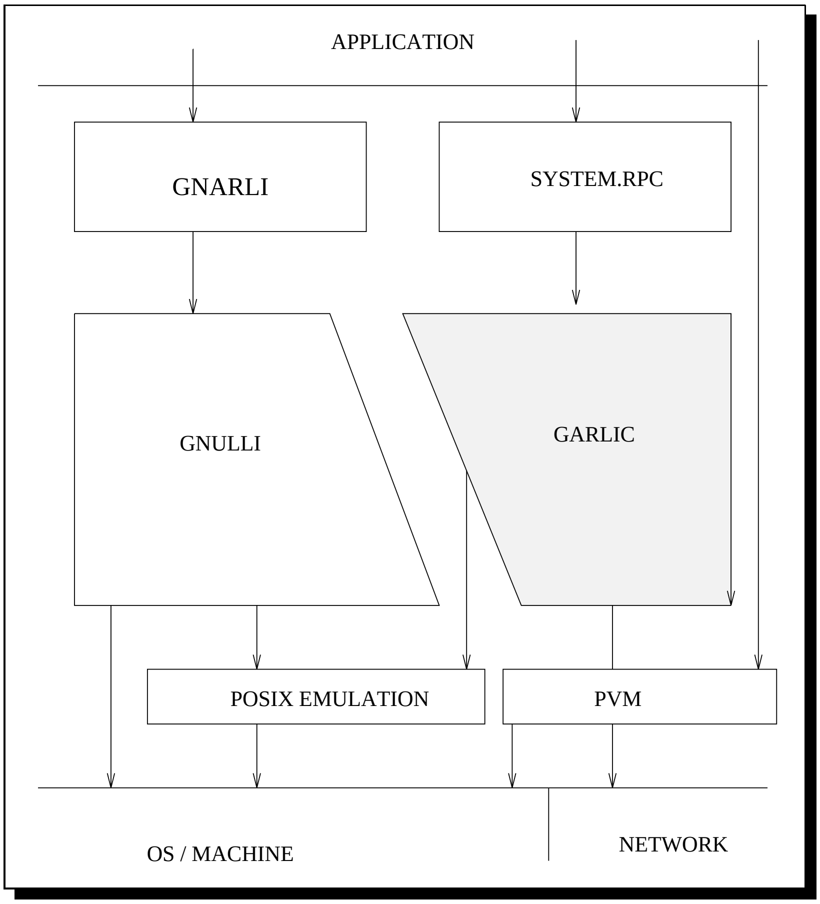

Fig. 1. Our system architecture

requires that all types be communicable across partitions (with the exception of local access types). The Ada9X distribution requires the presence of type specific-attributes that convert any type to and from a byte stream. The semantics of distribution requires that a client marshall the actuals of the call into such a stream, and that the server unmarshall them to create the local call to the server subprogram. If the stream attributes are implemented, it is a simple matter to transmit them between partitions using PVM primitives. In their absence, large amounts of code must be written to describe, e.g. the transmission of a record in terms of the transmission of individual components. As of this writing (October 1994) GNAT does not yet support stream I/O attributes, which has limited the scope of our implementation efforts.

6 Conclusions

Thanks to PVM, we have been able to experiment with the functionality of some important features of the Ada 9X model of distribution. PVM goes a long way towards supporting the Ada9X model, but as we saw there are difficulties of which the programmer must be aware. In particular, the absence of reentrancy in some of the PVM code is a menace in the presence of Ada tasks (several of which can reside on the same partition). Nevertheless we consider our prototyping experiment with PVM and GNAT to be successful, because it allows us to identify the required functionalities for a full implementation of the distributed systems facility of Ada 9X.

Our experiment and our software components illustrate the potential benefits of mixing the software engineering features of Ada 9X with the distributed aspects of PVM.

Our software components which offer an interface to PVM have been used at two distinct level. We access PVM from our internal communication layer as we use PVM services for inter-partition communication and for server localization. The application level also needs PVM to manage the distributed machine and the partitions. Numerous investigations can be done (i.e., reconfiguration) with the help of PVM.

Acknowledgment

The authors are very grateful to Prof. Ed Schonberg, New York University, for having reviewed an earlier version of this paper. They thank him also for having welcomed them in the GNAT team.

References

1. Alsys. *Reference manual for the Ada programming language.* Alsys, La Celle St Cloud, France, 1983.
2. H. Bal, J. Steiner, and A. Tanenbaum. Programming languages for distributed computing systems. *ACM computing surveys*, 21(3):260–322, september 1989.
3. J Dongarra, G. Geist, R. Manchek, and V Sunderam. Integrated pvm framework supports heterogeneous network computing, January 1993. available with ftp from netlib.
4. Intermetrics. *Ada9X reference manual - draft - version 5.0.* Intermetrics, Inc, Cambridge, Massachussetts, June 1994.
5. Y. Kermarrec and L. Pautet. Ada reusable software components for education in distributed systems and applications. In J.L. Diaz-Herrera, editor, *Proceedings of the 7th SEI conference on Software Engineering Education*, number 750 in Lectures Notes in Computer Science, pages 77–96, San Antonio, Texas, January 1994. ACM IEEE, Springer Verlag.

6. Yvon Kermarrec and Laurent Pautet. Evaluation of the distributed systems annex of Ada 9X and its implementation in GNAT. Technical report, Courant Institute of Mathematical Sciences, New York University, 715 Broadway, New York NY 10012, 1994. to be published.
7. E. Schonberg and et al. GNAT: the GNU - NYU Ada translator, a compiler for everyone. In *Proceedings of the TRI Ada conference*, Baltimore, Maryland, november 1994. ACM.