GARLIC : Generic Ada Reusable Library for Interpartition Communication

Yvon Kermarrec
Télécom Bretagne
Département Informatique
Technopôle de l'Iroise
29 285 Brest
France
yvon@enstb.enst-bretagne.fr

Laurent Pautet
Télécom Paris
Département Informatique
46, Rue Barrault
75 013 Paris
France
pautet@inf.enst.fr

Samuel Tardieu
Télécom Paris
Département Informatique
46, Rue Barrault
75 013 Paris
France
sam@inf.enst.fr

28 juillet 1995

ABSTRACT

This paper presents an implementation of the distributed programming features of Ada 95 within the GNAT system. The work we describe is the result of an international collaboration whose goal is to produce a high level environment for distributed system programming. This paper focuses on issues of interprocessor communication, since this is the core element of our software architecture. We describe the design and implementation of GARLIC, an interface between the network and the application. GARLIC is an extension of the predefined interface specified by System.RPC.

1 INTRODUCTION

A distributed system comprises a network of computers and the software applications that execute on them. These architectures are traditionally used to improve the performances, the reliability and the reusability of complex applications. The key difference between them and a centralized system is the absence of a shared memory. Therefore, message-passing is the sole facility for transferring information and making the various components of the distributed application cooperate. In fact, messages are the basic communication unit at lower levels as well (e.g., at the network level).

In the context of the GNAT project, our team is in charge of implementing the distributed features of Ada 95. Among other things, we have specified the compiler extensions needed to compile distributed programming constructs, and the mechanisms that must be made available at execution time [11]. The aims of this document is to present our approach for communication i.e., the way in which facilities for cooperation are made available to distributed applications. The Generic Ada Reusable Library for Interpartition Communication (GARLIC in what follows) is a high level communication tool that constitutes the interface between the Ada communication layer and the network level. We have followed the guidelines and the principles of the Message Passing Interface [12] to insure the ongoing compatibility of our software developments.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise or republish, requires a fee and/or specific permission.

©1995 ACM 0-89791-705-7/95/0011--0263 3.50

In the first section, we present the design criteria that guide our software developments. The second section is devoted to the specific needs of Ada 95 in the context of distributed systems. Next, we present the services that are required in our implementation to reach the goals we have selected. Then we describe GARLIC and its integration into the GNAT system. The last sections present tentative conclusions and our ongoing research work.

2 DESIGN PRINCIPLES

2.1 A BRIEF PRESENTATION OF THE ADA MODEL FOR DISTRIBUTION

The distribution of an Ada program over a network of processors is achieved through the Ada partition model. An Ada 83 program corresponds to an Ada 95 active partition. An Ada 95 program consists in a set of one or more partitions. A partition comprises one or more library units. Each partition may be elaborated and executed independently of other partitions in the program. Partitions may be active or passive.

The interactions between partitions can be of two kinds:

- by means of remote calls. These can be either statically or dynamically bound. Units that accept remote calls must be categorized with the pragma Remote_Call_Interface (RCI).
- by means of shared variables. Units that are categorized with the pragma Shared_Passive are assigned to a passive partition that can be accessed by other (active) ones. Variables in such units are shared variables.

In addition, the distributed programming Annex specifies the interface between the compiler and the Partition Communication Subsystem (PCS), defined in the predefined package System.RPC.

2.2 DESIGN OVERVIEW

The message-passing paradigm continues to dominate multiprocessor applications where portability and improved performance are both required. Recent work in developing standard message passing interfaces such as Message Passing Interface (MPI) [12], and POSIX 1003.12/1003.21 are indicative of the interest in increasing application portability without compromising performance. For example, MPI unifies various schemes and defines both the syntax and semantics of a message passing library. For those applications where performance is less critical, approaches such as the Parallel

Virtual Machine (PVM) [1] have become attractive alternatives.

The Remote Procedure Call (RPC) paradigm[2] is a well-known approach, It has been embodied, among others, in the OSF Distributed Communication Environment architecture[13]. DCE provides a collection of services similar to RPCs that are to be integrated directly in the OS. This approach is a significant step towards facilitating the programming of distributed systems. From our point of view, its drawback is that it still presents services as separate from the host programming language.

In contrast to these two approaches, the distributed object paradigm provides a more object-oriented approach to programming distributed systems. A distributed object is an extension of the notion of abstract data type, that permits the services provided by the type interface to be called independently of where the actual service is executed. When combined with object-oriented features such as inheritance and polymorphism, distributed objects promote a more dynamic and structured computational environment for distributed applications. The OMG Common Object Broker Request Architecture (CORBA) [5] is an industrial-sponsored effort to standardize the distributed object paradigm via the CORBA Interface Definition Language (IDL). Mappings between the IDL and different programming languages allow for the paradigm to be expressed directly in the chosen programming language[1]. Currently, several implementations of CORBA implementations exist that map the IDL to C++.

2.3 OUR IMPLEMENTATION CRITERIA

The implementation of the Annex within the GNAT system has been done according to the following criteria:

- Compatibility with existing distributed systems. In the previous section, we have contrasted the Ada approach with that of other models of distribution. These models are widespread and cannot be ignored. Moreover, interoperability is now viewed as a must for the construction of complex systems. This forces us to be compatible (or try to be) with these systems. Finally, our software developments benefit from these experiences because their models can help us find an answer and prevent us from falling into pitfalls. GARLIC also benefits from our previous software developments [8].
- Modularity. We have not mixed low-level communication features with the high-level communication interface, so that our system can be adapted to different distributed system architectures. Our approach makes use of the development of reusable software components [9].
- Portability. GNAT targets a wide range of architectures and operating systems thanks to the retargettable gcc back-end. Therefore, we must deal with various network interfaces and operating systems. Therefore, our implementation must be able to incorporate easily new protocols. Moreover, we may have to deal with more than one protocol for a given application and several protocols may have to co-exist.
- Efficiency. For obvious performance purposes, we want to make direct calls to the run-time low-level libraries and to implement efficient algorithms. Moreover, we use protected types and requeue to reduce the cost of race-free data synchronization. Another critical point is buffer management, and we have optimized their use.
- Availability. We want to make limited modifications to the GNAT system so that it remains readily available on a variety of targets. Therefore, we have chosen an approach where code transformation (carried out solely by the expander phase of the compiler) is the major issue. This approach makes it possible for other compilers to be readily adapted to make our implementation available.
- Evolution. The implementation is intended to be used in production-quality applications and to fulfill the needs of distributed system architects, designers, and programmers. What we have implemented is a compliant open implementation that may be extended to meet future requirements.

1. The mapping specification is available from the OMG as document 95-5-16 and may be found at http://conf4.darpa.mil/corba-ada

3 SERVICES PROVIDED BY GARLIC

GARLIC implements communication between partitions but also offers software services.

3.1 INITIALIZING THE DISTRIBUTED SYSTEM

When running, a distributed system is just another sophisticated application. It is sophisticated because it is composed of various cooperating software components. Taken alone each of these elements is simple. A name server (or a domain server) is nothing else but a data base and ancillary services provided to clients. The complexity of a distributed system comes from the difficulty of coordinating the activities of these components over time.

GARLIC initialization involves several dynamic configurations. In order to simplify the distribution of the application, several configuration parameters are determined at run-time. Another solution would be to determine these parameters at configuration time but some of these have to be validated dynamically in any case. Moreover, our approach allows future extensions, especially in the context of fault-tolerant applications. Among the parameters to check at run time, two of them need discussion: Version_ID and Partition_ID.

A Version_ID is a value of type String that identifies the version of the compilation unit that contains the declaration of a given program unit. According to the LRM, calling stubs and receiving stubs must check their Version_ID in order to ensure that both clients and server use the same RCI interface. A Partition_ID is a value of type Universal_Integer, that identifies the partition (that is to say, the machine) on which a given unit was elaborated. Two partitions are not allowed to have the same partition id, and there is only one receiving stub per RCI package. This must be checked at run time.

3.2 PARTITION ID SERVER

Partition_ID is an integer type and instances of this type play a major role since they make it possible for the application to access any partition in a unique way. Thus, these id's must be made unique. Computing a unique number is a well-known problem in distributed systems, and we have at least two approaches for it. The identity can be computed from system-dependent information (e.g., the process number and the IP machine number) or it can be a logical number.

We prefer to use a logical number since the programmer should not be able to interpret the embedded information and access partitions directly. Moreover, the logical number ensures independence from the underlying OS and its services. This approach has other useful consequences, e.g. if we integrate a fault tolerance mechanism like partition migration. In GARLIC, the partition identity is given by the value of an incremented central counter.

In our model, partition id's are dynamically determined. A partition identity server allocates unique id's to the application partitions. Thus, the computation of a unique identity is required when a partition elaborates itself, and partitions may need extra information to communicate with a partition once a given partition id is known. Moreover, the partitions need to localize this server in some way. The information on how to access the partition id server needs to be present during the elaboration of any partition. For this purpose, we introduce the notion of invocation key. This parameter provides all the needed information to reach the desired Partition_Id server: the protocol to be used, the machine number, as well as a port number in the case of TCP/IP. For reasons of flexibility, the invocation key is an environment variable. The partition id server and the main procedure are both located on the 'main' partition.

3.3 ELABORATION

In our system, we must ensure that the different protocol packages used by GARLIC are the first ones to be elaborated. As mentioned in section 3.2, the partition id server provides an unique identity to each new partition. Therefore, when a partition elaborates (e.g., when it *boots*), it starts by obtaining an identity from the partition id server with the help of an internal RPC. It uses the facilities of the protocol known thanks to the invocation key.

To avoid the full recompilation of the PCS when one adds a new protocol package, this communication package is declared in the body of System.RPC.GARLIC.General which elaborates all the protocol methods. Moreover, this ensures that the appropriate protocol is always available to communicate with the partition id server. When a protocol package elaborates, it registers with GARLIC.

Before elaborating System.RPC, we elaborate System.-RPC.GARLIC.General and the package which declares the partition id server. Thus, the partition may query its id or may become the partition id server if it is on the main partition.

3.4 ESTABLISHING A COMMUNICATION CHANNEL

The communication channel does not appear directly in GARLIC but in one of its child packages. To communicate with another partition, GARLIC needs a partition identity: the lower level communication services are then in charge of interpreting this partition identity and using the appropriate physical communication layer.

For example in GARLIC.TCP, communication is supported by sockets but all the related information is kept hidden from the user and is stored in a local repository (each partition owns and manages such a repository). Our current approach is to establish the communication links only when needed. The alternative is to open all of them at elaboration time. The expected benefit is to prevent a partition from wasting an unexpected time setting up the channel and dealing with connection protocol. This also avoids a number of deadlock problems.

3.5 OTHER SERVICES

GARLIC has been designed to be extensible and to simplify the incorporation of additional services. The notion of internal services makes it possible to extend the system functionalities. The programmer of a distributed application faces a daunting task since he has to coordinate several threads of control. His application code can be very complex since it needs to incorporate control of the distributed application itself as well as trace information to monitor the distributed execution. In this context, we have isolated a few services that are worth integrating at the system level instead of the application level. For instance mutual exclusion and clock synchronization algorithms based on the message passing paradigm are provided at the system level.

Termination and deadlock detection are of interest for the programmer but the algorithms are often quite hard to implement. The algorithm we have selected has been proposed by J.M. Helary et al. [6] and is based on the examination of messages to detect the stability and the passivity of the distributed system. GARLIC can monitor the messages and detect the occurrence of a partial / global deadlock or the termination of the application.

Another service that is of interest for the application level, integrates time related facilities. In a distributed system, we have no physical clock but applications may need such a time reference. We also need exchanges of messages to implement this service: clock resynchronization or drift prevention, for example.

We can also incorporate in GARLIC other paradigms for distributed systems like Linda [3]. The recovery block mechanism can also be of interest for fault tolerance issues and can be the base for transactions [7].

4 COMMUNICATION IN GARLIC

4.1 THE NOTION OF PROTOCOL

At the system level, the key element is the communication protocol. Traditionally, a protocol is used to refer to a set of precisely-defined rules and conventions used for communication between similar software modules running on the different computers in a network [4]. In our context, partitions run on computing elements of the distributed system and each partition includes all the run-time services it needs. GARLIC is the communication system that provides the interface between the application level and the network layer. As mentioned earlier, the model is more complex since several network interfaces can co-exist and since several network protocols can be used. Figure 1 illustrates the way GARLIC and network protocols are related.

The notion of protocol that we use in GARLIC is very close to the definition proposed by G. Coulouris. A protocol is an instance of a tagged type whose primitive operations are related to the access rules of the communication layer. The type is declared as abstract, and so are several of its primitive operations. The programmer needs to specify the abstract

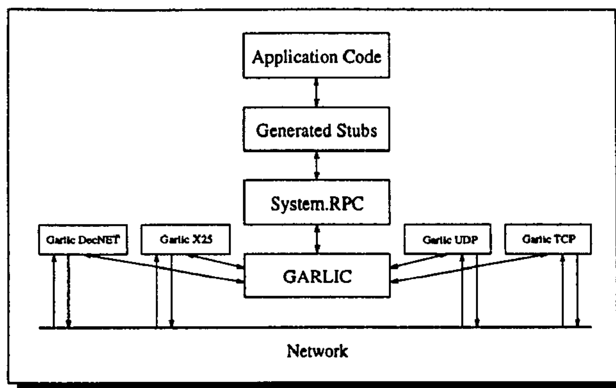

FIG. 1 - *GARLIC and the network protocols*

operations (i.e., the way to access to the communication facilities) every time a new protocol is introduced in GARLIC (i.e., every time the protocol type is extended). The protocol type is also limited because only one instance of every protocol can exist in the system at the same time. We indicate below the definition of this type and some of the primitive operations.

Initiate_Send procedure has to be called first. It locks in write mode the communication channel associated with the partition. This procedure blocks if the communication channel is already locked by another Send request. It also returns a Protocol object used to actually communicate with the partition. Send (Protocol.all, Partition, Data) performs the Send operation. The communication channel must be unlocked after all the messages are sent, by using Complete_Send (Protocol.all, Partition). The internal service type is given by the Operation field.

```
type Opcode is
  (Remote_Call,      -- Normal remote call
   Shared_Memory,    -- Shared memory message
   Message_Passing,  -- Free message passing
   ...);

type Protocol_Type is
   abstract tagged limited private;

procedure Initiate_Send
  (Partition : in  Partition_ID;
   Length    : in  Stream_Element_Count;
   Protocol  : out Protocol_Access;
   Operation : in  Opcode);

procedure Send
  (Protocol  : in Protocol_Type'Class;
   Partition : in Partition_ID;
   Stream    : access Params_Stream_Type);

procedure Complete_Send
  (Protocol  : in Protocol_Type;
   Partition : in Partition_ID) is abstract;

procedure Initiate_Receive
  (Partition : out Partition_ID;
   Length    : out Stream_Element_Count;
   Protocol  : out Protocol_Access;
   Operation : in  Opcode);

procedure Receive
  (Protocol  : in Protocol_Type'Class;
   Partition : in Partition_ID;
   Stream    : access Params_Stream_Type);

procedure Complete_Receive
  (Protocol  : in Protocol_Type;
   Partition : in Partition_ID) is abstract;
```

4.2 ADDING A NEW PROTOCOL IN GARLIC

GARLIC is designed to simplify the integration of new protocols. Adding a new protocol is done by providing GARLIC with a child package in which the programmer provides a type extension to Protocol_Type, corresponding to his new protocol. He also must indicate the concrete primitives that override the abstract ones.

A protocol has to register itself to GARLIC and this operation can be performed at any time. GARLIC knows about the new protocol in an indirect way since its primitives are stored in a descriptor which holds the dispatch table (see 2). In this way, GARLIC knows how to get access to the set of primitives set of a given protocol.

4.3 ENTRY POINTS IN GARLIC

GARLIC is intended to be used at two distinct levels. GARLIC supports the communication between the distributed partitions and Ada 95 through RPC and shared variables. Moreover, GARLIC offers communication facilities at the application level. At the administration level, GARLIC is responsible for dealing with internals of the distributed application: partition identities, localizing the partition id server, etc..

Several interactions do exist between partitions and they all use the communication facilities. We introduce here the notion of *service*. Every message in the network is prefixed with a service value which is used to execute the appropriate action. GARLIC knows the following services:

Remote_Call This is the basis for RPC implementation. On the receiver side, GARLIC unblocks a receiver task (an agent task that is waiting for an incoming request to execute a service for a client), informing it that there is a Data of length Length to read and thus a service has to be executed for a remote client. GARLIC will also transmit the Partition_ID of the client/sender, as well as information on a Protocol_Type'Class which will be used to retrieve Data from the communication channel;

Ask_For_New_ID This call may arrive only on the distinguished server, otherwise Program_Error is raised. GARLIC reads the Data field which contains the Network_Address of the calling partition. It then allocates the first unused Partition_ID, stores the network address corresponding to it for future use, registers the partition on the right Protocol_Type'Class object and returns the newly allocated Partition_ID with a Set_Your_Partition_ID message;

Set_Your_Partition_ID This call may not arrive on the distinguished server (since it is the one that allocates the new Partition_ID), otherwise Program_Error is raised. GARLIC reads the Data field which contains th

newly allocated Partition_ID and unblocks tasks which were waiting for it.

Query_Network_Address This call may arrive only on the server, otherwise Program_Error is raised. GARLIC reads the Data field which contains the Partition_ID of the requested Partition. GARLIC then looks up in its tables and returns a Set_Partition_ID message with the corresponding Network_Address (prepended by the Partition_ID) if it was found; it raises Program_Error otherwise.

Set_Partition_ID This call may not arrive on the server, otherwise Program_Error is raised. GARLIC reads the Data field which contains the Partition_ID and the Network_Address of the Partition to register in the tables. It then adds this information in its internal tables, registers this partition on the right Protocol_-Type'Class object and unblocks tasks which were waiting for informations on this Partition_ID.

Message_Passing This call may be used by the application to do message passing between partitions without using remote calls.

No_Operation This call is used to connect a partition to another one at boot time without performing any remote operation.

Shared_Memory This call is reserved for a future implementation of shared memory.

4.4 INTERNALS OF GARLIC

The GARLIC system can be considered as a medium-level communication protocol since it is used by System.RPC to exchange data between packages, but it does not deal directly with communication channels (see figure 1). Low-level transport agents are implemented in child packages and thus can be changed at any time without need of recompiling GARLIC (although relinking may be necessary) and user transport protocols may be added at any time by derivation of the Protocol_Type tagged type. A child package implementing TCP communications is provided under the name System.RPC.GARLIC.TCP.

System.RPC.GARLIC has no active thread of control of its own. Its services are either called by the upper layer (e.g., by the application level through System.RPC) when the application uses a communication channel, or by a lower level (e.g., System.RPC.GARLIC.TCP acts an interface to TCP/IP) to inform GARLIC that data need to be handled.

GARLIC is built around a Synchronizer that is a protected object. This synchronizer ensures that concurrent accesses from the upper or lower layers will not corrupt internal tables. Moreover, the entries implement event synchronization in a rather efficient way.

4.5 PRIMITIVE OPERATIONS OF GARLIC FOR THE APPLICATION LEVEL

GARLIC is a communication medium and its operations are quite similar to what is available in other existing systems. Communication through the network involves buffer and system level routines. The purpose of GARLIC is to provide the programmer with an unified network interface. In this sense, GARLIC is generic and interacts with the network in an ideal way.

Sending a message to another partition requires three steps:

- Preparing a stream and initializing it: GARLIC assumes that the stream (see LRM) has already been allocated. GARLIC automatically inserts the nature of the service (e.g., RPC call, pure message-passing) into the stream.
- Marshalling data in the buffer: data now has to be included in the stream. This operation is complex since we have reduced the number of buffer copies and allocations: this issue is critical because poor buffer management has a disastrous impact on efficiency. At this stage, the data can be encoded to fit heterogeneous machine characteristics by using either XDR or ASN-1.
- Once data are inserted and encoded, the buffer can be physically sent to its destination. GARLIC passes the buffer to the lower level, which is in charge of sending it according to the characteristics of the required protocol.

Message reception involves three complement operations that we do not detail in this document. The key element we must point out is that GARLIC plays a role of active interface and all the network details are completely hidden from the user code.

4.6 GLOBAL VIEW OF GARLIC

In figure 2, we indicate the global architecture of GARLIC with its entry points and services.

4.7 PARTITION ID MANAGEMENT

ID CREATION

When a partition begins its elaboration, the first action it performs is to declare itself to the partition id server (1). The partition sends a registration request that provides routing information, such as the host name if the protocol in use is TCP. When the partition id server receives such a request, it allocates an id and caches the routing information. It sends to the newly registered partition its partition id (2).

ID QUERY

When a partition wants to send a message to another partition with a given id, it has to query the partition id server for routing information in order to communicate with the desired partition (3). The partition id server blocks the request if the information is not available (ie. if the desired partition has not registered itself yet). This service provided by the partition id server informs any partition on the routing data needed to connect to another partition (4). For instance, in a TCP context, this information consists in the host name and the port used by the remote partition. When network addresses are known, the two partitions can communicate (5 and 6).

5 CONCLUSION AND WORK IN PROGRESS

The design and implementation of GARLIC are complete and these elements will be integrated within the GNAT system. In this way we hope to promote further contributions and refinements to our implementation so that Ada can become a dominant language for distributed systems in both

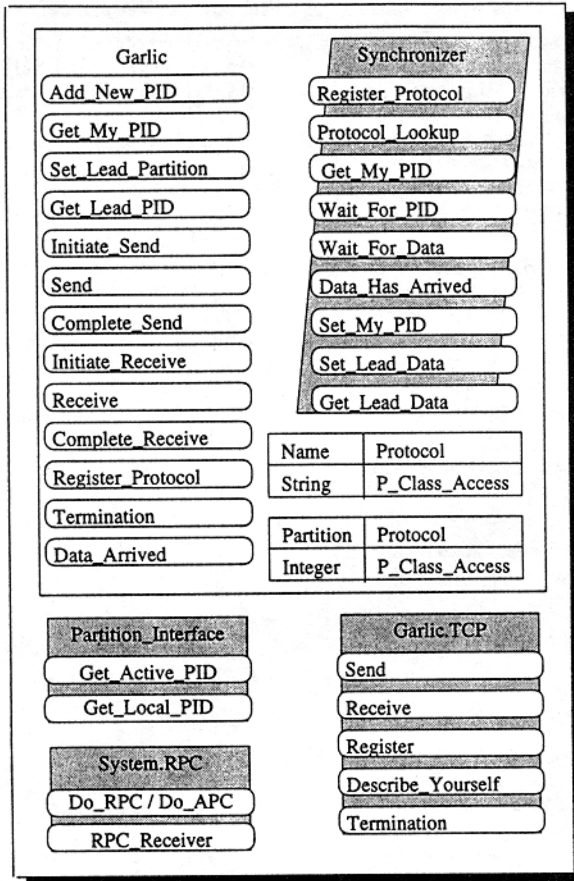

FIG. 2 - *Software architecture of GARLIC*

academia and industrial applications. In our implementation, we have tried to integrate various perspectives and to consider existing approaches; this should make our system attractive to the non-Ada community.

GARLIC is a generic software component that has been designed in order to accommodate several network protocols and communication systems. The implementation makes intensive use of the new facilities of Ada 95 because they reduce the complexity of the software (tagged types and protected objects, for example). Our previous developments were aiming at similar goals [8] but the implementation with Ada 83 was more tedious to produce and more cumbersome.

We hope to continue the project and to extend GARLIC to incorporate facilities for the management of shared memory, among others. The algorithm [10] has already been validated, and it fits the requirements for embedded systems.

Another target of future developments is to adapt the repository of software components that we have produced for distributed systems and Ada 83 [9]. Our aims are to integrate reusability and software engineering concepts for distributed system programming, which is too often considered as network or system programming. Some of these services can be integrated into GARLIC directly. The programmer could then

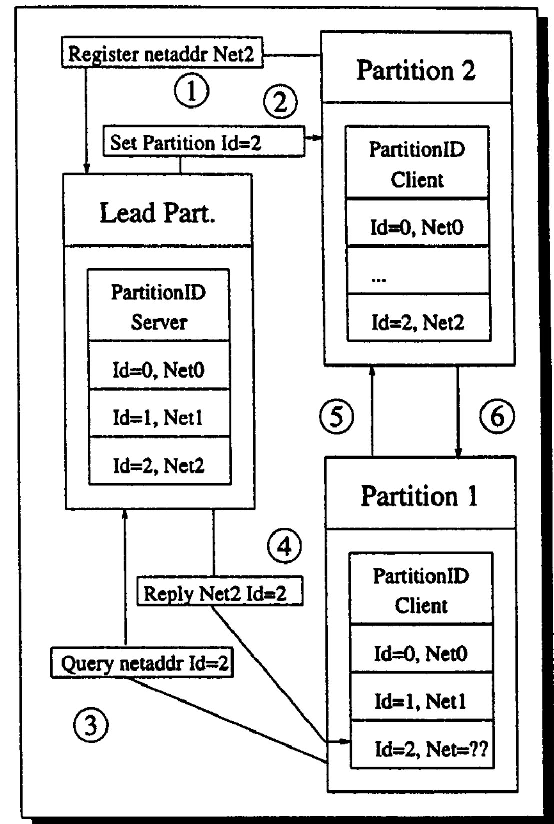

FIG. 3 - *How to communicate with another partition*

benefit of these services and be guaranteed of an efficient implementation. The services that can be provided include time related facilities, deadlock and termination detection, fault tolerance mechanisms.

Acknowledgment

The authors are very grateful to the GNAT team at New York University for having invited them to the exiting GNAT project. The authors are also specially grateful to the anonymous referees for their helpful comments. Pr. Ed Schonberg and Anthony Gargaro have motivated intensive brain storming on the issues of distribution.

REFERENCES

[1] Al Geist et al. *PVM: Parallel virtual Machine.* The MIT Press, 1994.

[2] A. Birrell and B. Nelson. Implementing remote procedure calls. *ACM Trans. Computer Systems*, 2(1):39–59, February 1984.

[3] N. Carriero and D. Gelernter. How to write parallel programs : a guide to the perplexed. *ACM Computing Surveys*, 21(3):323–358, September 1989.

[4] G.F. Coulouris and J. Dolimore. *Distributed systems: concepts and design.* Addison Wesley, 1988.

[5] DEC, HP, and et al. The common object request broker: architecture and specification. Technical Report OMG 91-12-1, Object Management Group and X Open, December 1991.

[6] JM Helary, C. Jard, N. Plouzeau, and M. Raynal. Detection of stable properties in distributed applications. In ACM, editor, *Sixth ACM symposium on principles of distributed computing*, pages 125–136, Vancouver, August 1987.

[7] Y. Kermarrec, L. Nana, and L. Pautet. Implementing an efficient fault tolerance mechanism in Ada 9X: an early experiment with GNAT. In *Ada Belgium conference*, Bruxells, Belgium, November 1994. ACM and Université Libre de Bruxelles.

[8] Y. Kermarrec and L. Pautet. Ada communication components for distributed and real time applications. In *Proceedings of the TRI Ada 92 conference*, pages 530–536, Orlando, Florida, November 1992. ACM SigAda.

[9] 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.

[10] Y. Kermarrec and L. Pautet. Integrating page replacement in a distributed shared virtual memory. In *Proceedings of the 14th international conference on distributed computing systems*, Poznan, Poland, June 1994. IEEE.

[11] Y. Kermarrec, L. Pautet, and E. Schonberg. Design document for the implementation of distributed system annex of ada 9x in gnat. Technical report, New York University, Courant Institute, 715 Broadway, New York NY 10012, March 1995. (to be published).

[12] Message Passing Interface Forum. Mpi: a message passing interface standard. Technical Report 230, CS Department, University of Tenessee, Knoxville, April 1994.

[13] Ward Rosenberry, David Kenney, and Gerry Fisher. *Understanding DCE*. O'Reilly and associates, inc, 1993.

Yvon Kermarrec got a PhD degree in computer science from IRISA at Rennes University in 1988. The title of his dissertation is "An approach for distributed system simulation : software components in Ada". He joined the Ada-Ed group at New York University as a visiting researcher. He worked with Ed Schonberg and Robert Dewar on the NYU Ada Compiler. In 1990, he joined the faculty at Ecole Nationale Supérieure des Télécommunications in Paris as assistant professor. He has just arrived at Ecole Nationale Supérieure des Télécommunications in Brest, France. He teaches Ada, software engineering and distributed algorithms. His research interests are: distributed systems, Ada and programming languages.

Laurent Pautet received the Diplôme d'Ingénieur from the Ecole Nationale Supérieure des Télécommunications Paris (ENST-Paris), Paris, France in 1989. He worked with E. Schonberg and H. Operowsky on parallel garbage collectors. He received a PhD degree in computer science from the ENST in Paris in 1994. He concurrently joined Dassault Electronique (hard real-time for avionic embedded systems) Paris, France as a research engineer in Dec. 1990. He joined the ENST Paris University as professor assistant. His research interests include software engineering, distributed systems, and real-time systems. He is currently participating to the implementation of the Ada9X Distributed Systems Annex in the realm of the GNAT project. He is also involved in the design and the development of a hardware/software environment, called SPIF, offering a genuine prototyping environment for embedded real-time systems.

Samuel Tardieu received the Diplôme d'Ingénieur from the École Nationale Supérieure des Télécommunications de Paris (ENST-Paris), Paris, France in 1994. He is participating to several GNU projects, including the distributed annex of the GNAT compiler. He is currently working as an engineer for the French army until August 1996 and will be candidate for a PhD degree in computer science starting September 1996.