A Distributed Objects Framework using Virtual Machines

Stefano Gobbo

Dipartimento di Elettronica e Informazione
Politecnico di Milano - Italy
gobbo@elet.polimi.it

October 7, 1994

Abstract

The growing demand for the exploitment of the increasing computing power led to the design of multiple solutions, covering different levels of abstraction (operating system[11], virtual embedding[10], programming interface[9], object oriented computing[1], graphic application[8, 4]). The idea proposed here consists in choosing the appropriate solution for the various levels and joining them in a unique framework supporting both the Parallel and the Object Oriented paradigms.

1 Introduction

In the last years, there has been a proliferation of software architectures designed to support parallel programming in heterogeneous network systems. These instruments vary from distributed programming libraries to intrinsic parallel languages [5, 12] and parallel extensions of traditional sequential languages [7, 3].

Furthermore, the research seems now oriented in the development of higher level visual environments [8, 4], whose aim would be to alleviate the designer task from many of the problems which have to be considered when designing parallel applications.

However, each of these solutions poses its attention on the approach to a particular class of problems. Moreover, the scalability of these environments is bounded by their self-contained designs. Each environment defines its own semantic, without leaving any door open to the dialogue with other environments. Even the paradigm they support is often different. HPF-like languages adopt the data parallel computation model, which is well suitable for SPMD techniques (i.e. domain decomposition algorithms), while low level programming libraries and high level graphic applications usually adopt message passing and data/control flow mechanisms.

On the other hand, the proliferation of such environments makes prohibitive any effort oriented to the definition of a standard that developers should follow in the design of their tools. Therefore, we are facing a babel of languages and environments for the development and the design of distributed memory parallel applications, each one speaking its own language and well suited for the solution of a limited (even though wide) class of problems.

Our proposal is the development of a distributed system in which standalone parallel applications designed using different tools or languages and following different paradigms can communicate.

2 The Framework

The scenario is a cluster of hosts composed by a heterogeneous network of workstations and parallel machines, on which top runs a communication system that emulates a generalized distributed memory multiprocessor (henceforth referred to as *Virtual Machine*). Here, message passing parallel programs can be ported to a distributed environment, and new distributed applications can be designed without the need to bother of communication and synchronization details.

Having a single support in charge for the execution of every parallel application running on the same virtual machine is important, since each component of each application is started, controlled and terminated following the same basic steps. When we decide to run a particular application we specify to the run-time support the coordinates of every component of the application, along with the input parameters we want to pass it. The run-time support locates these components and runs them, passing input values to the apposite elements (the so-called *starting nodes* of the application). Upon completion, the *terminal nodes* of the application pass back the results to the run-time support, which provides to notify them to the user. All these steps are independent from both the tool and the paradigm involved in the development of the application, the same source-independence we witness during the execution of sequential processes on SISD architectures (i.e., UNIX processes).

If every instrument rely on the same execution environment for running and controlling the computation of their parallel applications, there is a chance that different applications designed using different development systems can cooperate. The problem is moved to the definition of an agreement base which applications developed using different environments must follow to communicate.

Object Oriented characterization could be a possible solution to this problem. The idea here proposed is to assign each application an interface, independent from both the language the application is implemented in and the environment used to design it. Through this interface, parallel applications are therefore converted into *Distributed Objects* (DO) that export *methods* to any other application enabled to access their interfaces, allowing them to *use* existing DOs or to *exploit* them to build new ones.

3 Structure

The framework here proposed wants to provide a methodology for attaching interfaces to applications and for allowing distributed environments access these interfaces. Figure 1 shows the structure of the framework. It is layered on different level of abstraction, each managing a different kind of requests:

- **Physical level:** the hardware computing platform, consists of a heterogeneous network of workstations and parallel machines.
- **Virtual level:** the communication system, used to cluster the underlying network, provides to the higher levels a general distributed memory parallel machine programming interface.
- **Object support level:** the distributed development systems which provide the communication libraries on which to build Distributed Objects.
- **Object definition level:** higher level interfaces to the distributed system. They can be graphic interfaces and/or dedicated languages interpreters/compilers.
- **Circuit level:** DOs request are managed at this level and translated in references to active Objects.

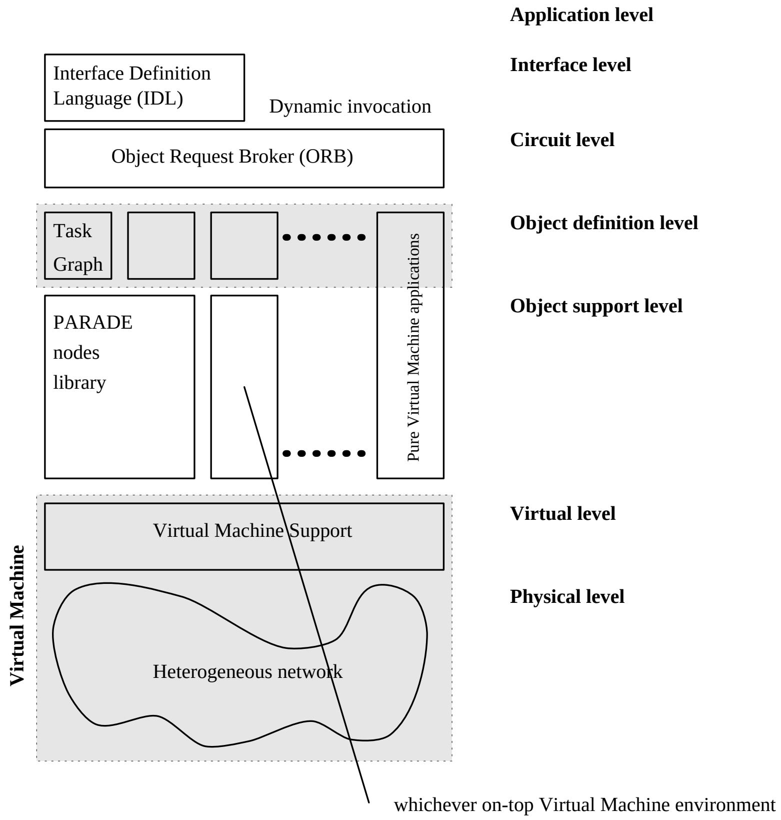

Figure 1: The Distributed Object Framework and the Virtual Machine.

- **Interface level:** Provides applications static/dynamic methodologies for invoking DOs.
- **Application level:** Parallel applications running in this framework.

Here follows a brief description of each of the previous level:

3.1 Physical-Virtual levels

The lower level of the system consists of the Virtual Machine, a heterogeneous network of workstations and parallel machines clustered as a single virtual parallel environment.

3.2 Object support level

The *Object support level*, layered on the top of the Virtual level, is composed by a set of distributed development systems which rely on the Virtual machine to provide the support for implementing distributed applications.

3.3 Object definition level

Object support level systems are often sided by higher level interfaces designed to support the user in his/her task of developing distributed applications. These interfaces may be either distributed oriented languages [3, 7] or graphic interfaces [4, 8] or both.

3.4 Circuit level

Object Oriented support is accomplished at a higher level introducing an Object Request Broker (ORB), whose task is to resolve requests to Distributed Objects implemented on any node of the Virtual Machine. Distributed applications become DO at this level, registering their application/interface association in an *Implementation Repository* where the ORB looks during requests resolution. Any application is allowed to make a request to an existing object, leaving the ORB the task to locate its implementation, run it (if necessary) and resolve the request by returning to the demanding application a reference to the DO.

Figure 2 shows a request being sent by a client to an object implementation. The client is the entity that wishes to perform an operation on the object and the Object Implementation consists of the code and data of the parallel application that actually implements the object. The ORB is responsible for all of the mechanism required to find the object implementation for the request, to prepare the object implementation to receive the request, and to communicate the data making up the request.

3.5 Interface level

Applications can request DOs both statically and dynamically. Static requests are resolved compile-time, linking the application to the proper interface for any object requested. These interfaces are written in C-like fashion using an Interface Definition Language (IDL). Dynamic requests are resolved run-time, asking the ORB for the appropriate references via Dynamic Invocation interface, a single interface independent of the interface of the target

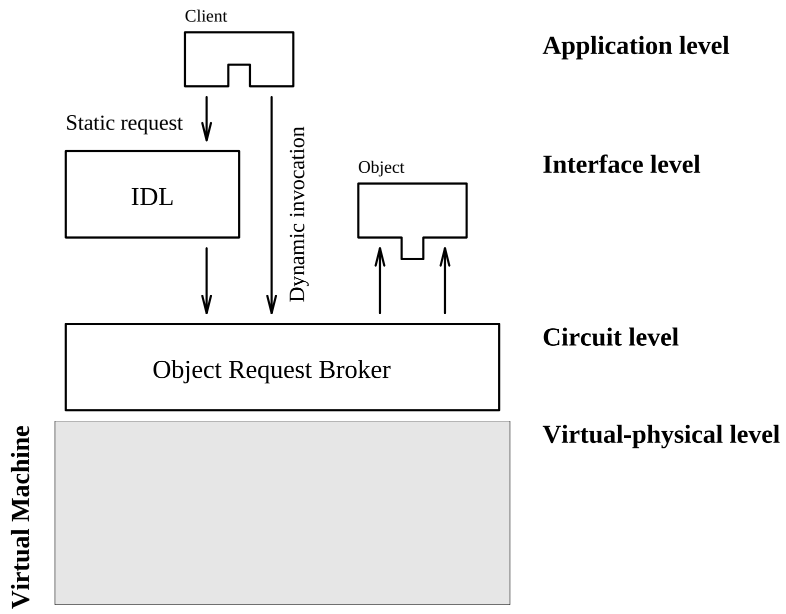

Figure 2: Distributed Object requests through the ORB.

object. The interface the client sees is completely independent of where the object is located, what programming language it is implemented in, or any other aspect which is non reflected in the object's interface.

4 Providing Interfacing Methodologies to Parallel Applications

Once we described the functionality of each level of the framework, there is still a basic question to answer: how do we assign interfaces to parallel applications?

Recall that there is a single run-time support responsible for the execution and the control of every parallel application of our system. To solve our purposes, this support must integrate the two following capabilities:

- To act as a *distributed kernel*, loading applications components and establishing a communication between user and application for passing parameters and returning results.

- To implement the ORB features, providing a mean for locating applications via IDL interfaces and returning references to the Distributed Objects they represents.

Thus, an application (the client) is free to ask a particular feature to the ORB section of the run-time, which looks for the apposite application/interface associations in the implementation repository. The ORB returns to the demanding client the reference to a DO (the server application) attached to the requested interface. At this point, the client invokes the DO using that reference. The kernel section of the run-time recognizes the reference access and provides to load the requested DO, passing to its starting nodes the data specified by the client at the invocation, and to return the results produced by the DO to the client. In fact, the ORB acts as a filter which enables parallel applications executions to be interfaced with another parallel application running over the same virtual machine (see Figure 3).

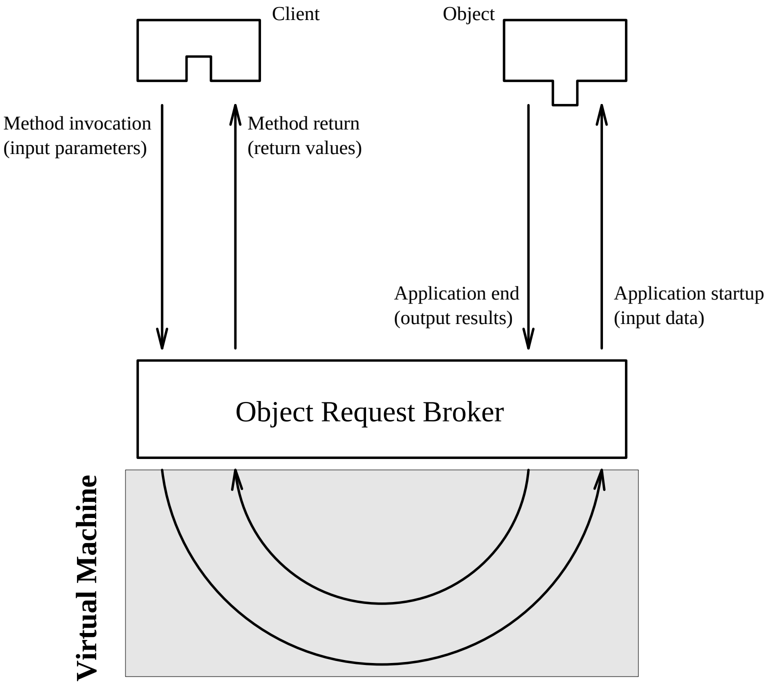

Figure 3: Input/Output data filtering through the ORB.

5 Choices

Most of the level are implemented *reusing* existing projects.

The virtual machine is implemented using **PVM**. PVM is a software framework for heterogeneous concurrent computing in networked environments. PVM provides a straightforward and general interface that permits the exploitment of a collection of heterogeneous computing elements interconnected by one or more networks. The underlying infrastructure permits the execution of applications on a virtual computing environment that supports multiple computation models. At the moment, PVM seems oriented to become the standard *de-facto* of the non-dedicated distributed communication systems.

In our proposal, ORB and IDL specifications follow the *Common Object Request Broker Architecture* (**CORBA**) [6] definition, a self-contained response to the Request For Proposal issued by the Object Management Group (OMG), a task force of many vendors (including DEC, HP, Hyper Desk, NCR, Object Design and SunS oft) oriented to the study of object definition in distributed environments.

CORBA is structured to allow integration of a wide variety of object systems, providing the mechanism by which objects transparently make requests and receive responses. The ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems.

Even the ORB mechanisms are implemented with PVM, resulting in a whole run-time support capable both to reference and to execute interfaced applications.

5.1 An Example: Exporting PARADE Applications

PARADE (PARallel And Distributed Environment) is a user-friendly environment which provides a easy-to-use visual interface for design, task mapping, execution control, monitoring and debugging of graphically expressed parallel applications. The programming paradigm adopted by PARADE refers to the notion of coarse-grain *data flow*[2], traduced here in a directed acyclic graph (DAG) where the nodes are either computation nodes or control nodes, and the arcs represent the flow of data and the synchronization between data flows.

Each node executes a sequential program, activated by a simple procedure call performed by the node controlling code. Figure 4b) shows the basic principle of a task: task execution is triggered by the arrival of data x, on which the function *f* is applied in order to produce the result *f(x)*.

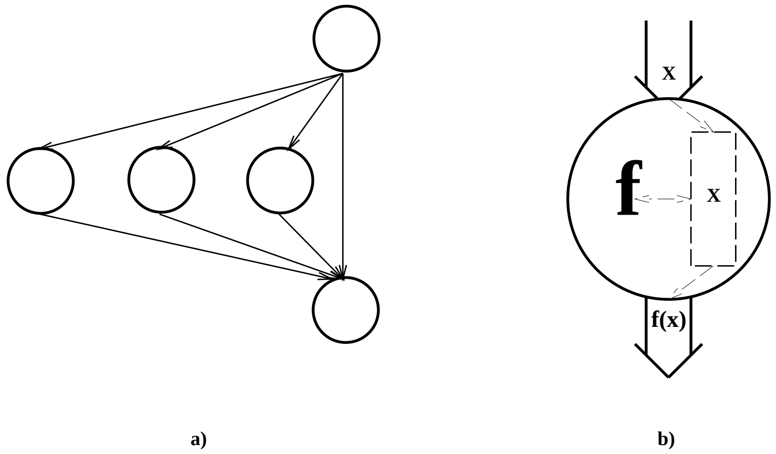

Figure 4: a) A simple task graph b) A task graph node.

Applications consist of a pool of nodes, structured in directed acyclic graph, called *task graph* (Figure 4a) Since task graphs defines both the data flow and the control synchronization, it's easy to locate *starting* and *terminal* nodes in every application. The activation of a PARADE application consists in spawning every node of the task graph, activating the starting ones passing them optional input parameters. These data activate the nodes and are passed for processing to the included functions. Before termination, terminal nodes return execution control and results to the execution environment that runs the application.

In its latest form, PARADE has been ported over PVM, gaining access to every virtual machine supported by PVM. Porting PARADE on PVM have meant leaving to PVM the total handling of application control, since PARADE applications are initiated by this kernel through a `pvm_spawn` specifying all the task composing the application. This situation makes easy the exporting of PARADE application as DOs, since every PVM-based application is allowed (via the kernel-ORB PVM-based support) to reference PARADE task graphs, passing the triggering data x to the starting nodes of PARADE applications and obtaining back the *f(x)* values returned by terminal nodes (Figure 5)

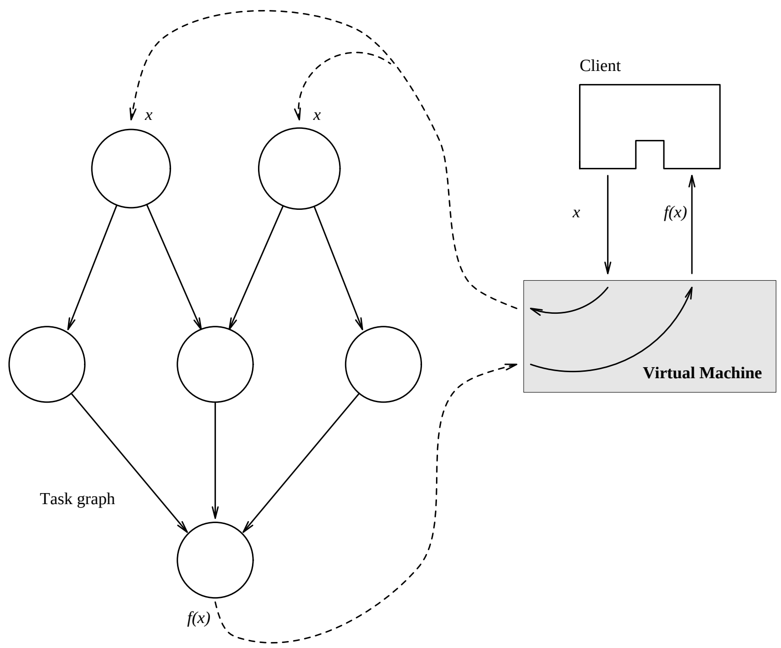

Figure 5: Virtual Machine calling a Task graph.

6 System scalability

Another capability of the proposed framework is its scalability. This system is scalable since any programming environment built on the top of the PVM communication library can be added without influencing the other components.

Moreover, the system is scalable at the application level, since the Distributed Objects definition does not depend on the environment used to build the application. So, any development environment based on the PVM communication system can be used to write both client applications and Distributed Objects.

In summary, the system here proposed is scalable under two different points of view:

- **Object support:** the Object support level is composed by a set of distributed development systems which rely on both the Virtual machine and the ORB to provide the support for implementing distributed objects. The system is scalable at this level because here can be

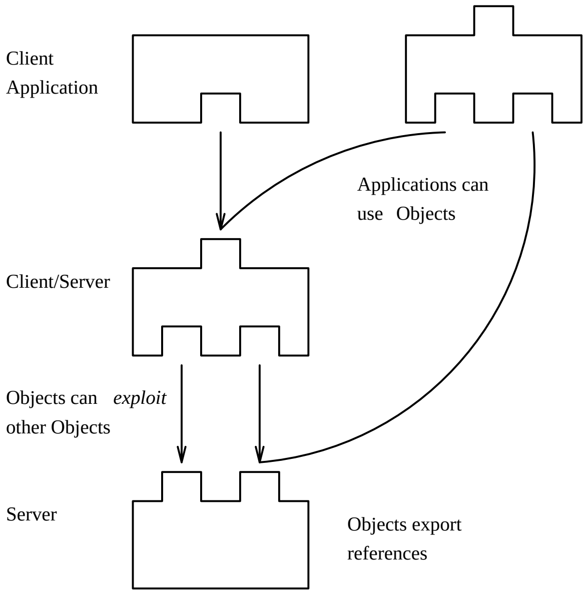

Figure 6: Scalability in the Object implementation.

inserted any development environment based on the PVM communication system. For example, we are developing a system in which the PVM virtual machine is expanded with the porting on PVM of the PARADE nodes library [9] (see Section 5.1).

- **Object implementation:** the DO library, the pool of Objects implementation managed by the ORB, can be easily expanded. In fact, any application (a client of the existing DOs) can be transformed in a DO (thus becoming a server itself, see Figure 6) simply designing an IDL interface containing the description of the references exported to the system by the application.

7 Concluding Remarks

This paper tries to describe a common famework in which different parallels environments, supporting different parallel programming paradigms, can cohexist and cooperate.

The ideas proposed here offer to the users of potential virtual machines the following advantages:

- the exploitment at best of the strongnesses of every parallel/distributed environment residing over the virtual machine.

- the possibility to add a low level but more flexible control to standalone distributed parallel applications.

- the instruments to create a vast library of distributed applications, which can be used both by specific clients and in the design of more complex modular general-purpose distributed applications.

References

[1] S. Vinoski, "Distributed Object Computing With CORBA", *C++ Report*, July/August, 1993.

[2] W. Ackerman, "Data Flow Languages", *IEEE Computer*, pp. 15-22, February 1982.

[3] F. De Paoli and M. Jazayeri, "FLAME: a language for distributed programming", *Proceeding International Conference on Computer Languages*, pp. 97-78, March 1990.

[4] A. L. Beguelin, J. J. Dongarra, G. A. Geist and R. Manchek, "HeNCE: A Heterogeneous Network Computing Environment", August 27, 1993.

[5] High Performance Fortran Forum, "High Performance Fortran Language Specification", *Thecnical Report Version 1.0*, Rice University, May 1993.

[6] The Common Object Request Broker: Architecture and Specification, OMG TC Document Number 91.12.1, Revision 1.1, December 6, 1991.

[7] H. E. Bal, M. F. Kaashoek and A. S. Tanenbaum, "Orca: A Language For Parallel Programming of Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. 18, NO. 3, pp. 190-205, March 1992.

[8] D. Bruschi, S. Gobbo, P. Lenzi, E. Pozzetti and G. Serazzi, "A User-friendly Environment for Parallel Programming", *Proceeding Euromicro Workshop on Parallel and Distributed Programming*, pp. 451-456, January 1993.

[9] E. Pozzetti and V. Vetland , "Parallel Programming with PARADE", *Workshop Elaborazione Parallela: Ricerca ed Applicazioni*, pp. 1867-1882, September 1994.

[10] V. Sunderan, "PVM: a framework for parallel distributed computing", *Concurrency: Practice and Experience*, Vol. 2, December 1990.

[11] J. G. Mitchell et al., "An Overview of the Spring System", Sun Microsystems Inc., 93.

[12] H. P. Zima and B. M. Chapman, "Compiling for Distributed-Memory Systems", *Research Report APCP/TR 92-17* Austrian Center for Parallel Computation, University of Vienna, November 1992.