

CORBA vs. Ada 95 DSA

A programmer's view

Yvon Kermarrec
ENST Bretagne
Département IASC
Technopole de l'Iroise
29285 Brest Cedex
France

`Yvon.Kermarrec@enst-bretagne.fr`

Abstract

Recent evolutions in computer networks have made it possible to consider distributed platforms as possible architectures for numerous applications. The classes of distributed applications are numerous and go from time consuming application (performance quest) to geographically distributed applications through safety and critical softwares. As beforehand, hardware evolutions have been fast paced and much earlier than software techniques to build applications on these distributed targets. In this paper, we shall investigate and compare three different approaches: the language approach of Ada 95 for distributed systems vs. two distributed platforms. And for this evaluation, we shall take the developer's view, that is to go beyond the technical response but to take into account the developer of the distributed application.

1 Context of the study

This paper analyses and presents discrepancies between the Ada 95 model for distribution and CORBA. The aim of this paper is not to make a complete evaluation of both approaches but takes the perspective of the programmer: that is how to build a distributed program from the existing tool box.

Many people perceive CORBA [DEC et al., 1995a] [DEC et al., 1995b] [J Siegel et al., 1996] as **the solution** for programming distributed systems. The entire community is focused on the OMG solution which is proposed by key players in the computer industry and research. Moreover, the commercial side of CORBA and the enormous promotion through books, articles and products has transformed the OMG specification into an unavoidable solution.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. SIGAda'99 10/99 Redondo Beach, CA, USA ©1999 ACM 1-58113-127-5/99/0010...\$5.00

On the other side, the Ada 95 model for programming distributed systems Distributed System Annex or DSA [Intermetrics, 1995] was published after the initial CORBA proposal and did not receive any publicity. Worse, the Ada community itself was a little bit reluctant (and not too many people knew it) and the annex was about to be thrown away from the last release of the reference manual. This situation is rather unfortunate for the promotion of the annex and is also illustrated by the lack of implementations. Up to now, only one Ada environment implements the annex completely and this illustrates once more the relative and poor interest for the complete Ada solution.

We believe that DSA [Kermarrec et al., 1995] [Gargaro et al., 1995a] has a better approach of distributed systems than CORBA and other middleware because Ada is a programming language. The distributed application is one entity and the execution semantics is well defined as the interaction with the other features of the Ada (e.g., asynchronous transfer of control, tasking, memory control). DSA can also appear as an interesting vector to attract people to Ada: Ada has unique capabilities among other programming languages. Its approach for distributed systems with various paradigms (communication with shared memory, by message passing, the client/server model and distributed objects) enables it to tackle various classes of problems.

This paper is organized as follows. In the first section, we describe the initial contact between CORBA and DSA ; the first sight of any approach plays a major role since the reader can keep investigating or discard right away. In the second section, we present the issues which have to be addressed when proposing a framework for distributed systems developments. The third section investigates software engineering issues which play a major role if the application has a long life cycle or if safety/critical systems is involved. In the forth section, we investigate how the commercial promotion around CORBA could be used in the context of the distributed system annex. For the conclusion, we shall present a few actions that could help DSA go beyond the confidentiality it knows from the very beginning.

2 A programmer's view : getting started

CORBA is simple by itself and this leitmotiv appears everywhere for the promotion of the OMG solution. This assertion is certainly true when it comes to the core specifications of CORBA: the principles of an ORB are well understood and passing object references from one site to another does not require elaborate knowledge. CORBA relies on specifications that have been produced by software industrialists who are also natural competitors to each other. As a result, the produced specifications appear as a minimal set of agreement between all the OMG members: the common view of what a distributed system should offer to the programmer.

This initial minimal design of CORBA has been defeated by the new subsequent releases of the specification. In fact, the new releases address more and more elaborated issues and wider domains. The possibility to interact between quite different languages has reinforced in a way the complexity of the specifications: enabling object references to be passed between a SmallTalk client and a C server raises numerous questions that have to be addressed (e.g., memory allocation and control, mode of parameter passing). The knowledge of CORBA and its related services must be known by the programmer to develop a multi-threaded server.

On the contrary, OSF Distributed Computing Environment (DCE) [Shirley, 1992] [Rosenberry et al., 1993] was designed as a complete solution for programming a distributed application by selecting tools from various vendors. This approach is more suitable for information technologies and has been widely used for cooperative or client/server applications. "OSF DCE provides services and tools that support the creation, use, and maintenance of distributed applications in a heterogeneous computing environment." The result was a set of integrated services which address most of the issues that a programmer may need for a distributed application: e.g., security, name services, clock synchronization, concurrency and synchronization. In fact, the approach was quite opposite to the one chosen by OMG: instead of specifications, DCE is a product available on several platforms. Nevertheless, OSF has produced an almost unusable tool because of its complexity and of the numerous possibilities offered to the programmer. It creates complexity for the use of the tool before even using it.

In fact, for the resulting complexity, OSF DCE and OMG CORBA are very similar. OSF DCE puts the programmer in a puzzling context where he needs to understand hips of documentation before starting any coding. On the other side, CORBA looks simple but the resolution of even simple problems requires platform specific expertise.

Ada 95 and its DSA look complex at the first sight of the reference manual. The words used in the annex do not help by restricting and forbidding the programmer to make choices and selections. The wording itself has a negative impact on the reader because it prevents from doing things but does not propose a positive approach. The initial contact of any programmer with the annex may trigger rejection and a negative impact.

The distribution model of Ada 95 is simple and extend the non distribution version of Ada. That was also a prerequisite for the design of the annex and a wish to keep the integrity of Ada 95 language. The reference manual goes even onto the details by defining what kind of distributed

systems and applications are in the scope of the annex. A distributed program comprises one or more partitions that execute independently (except when they communicate) in a distributed system. A partition is either active or passive. An active partition has one or more threads of control. A passive partition has no thread of control of its own and its data and subprograms are accessible to one or more active partitions. The intention of a passive partition is to represent a shared address space across one or more computers on which the application is configured. Such definitions show clearly that DSA has been designed for all kinds of distributed systems: the general context of the definition makes the annex an interesting framework for numerous classes of distributed systems: from multiprocessors with shared memory, to workstations connected with a network through multiprocessor boards with communication bus.

From my experience as associate professor in computer science, the training of students to DSA is very smooth even for those who have no prior knowledge of Ada. This is made possible, because of the simple underlying model of Ada 95: the Remote Procedure Call is nothing else than a subprogram call, a distributed object is very similar to any other object and the call sequence syntax is very simple. GLADE and its configuration language (GNATDIST [Kerमारrec et al., 1996]) makes the transition to distributed systems even smoother and natural. In fact, the distributed execution of the application does not require any system-level knowledge and is very natural because the environment takes care of the network issues and the operating-system calls. All this makes it possible to run a distributed application without almost any prerequisite and this is very encouraging for students or beginners who have encountered difficulties with the use of sockets for example. Most of the students are able to build and run a distributed application in less than 2 hours of labs and after a 2 hour presentation of DSA. This initial contact generates also a lot of curiosity on the Ada language itself (I have never distributed so many of the Ada CD Roms!), on the underlying mechanisms that make all these transparencies and on the existing facilities of GLADE to address more complex issues.

3 Issues in distributed computing

The client / server paradigm appears as an interesting candidate for many applications. This is also the case with the notion of distributed object where object oriented features such as inheritance and polymorphism associated to distribution promote a more dynamic and structured computational environment for distributed applications. The development of a distributed application requires different skills and users interventions. Among them we have at least the following points that need to be addressed:

- How do we locate a service ?
This initial concern deals with the discovery of the distributed context. A client and a server do not need to be launched at the same time and therefore the client must be able to determine at run time on which computing element the service is available. This lookup is provided by directory/name service whose network address is well known by all entities in the distributed system.
- How do we provide/access the service ?
Specific actions have to be made by the server to advertise its offers and services. The initial design choice

consists in defining the contract (or the interface) between the clients and the provider. Once the server is ready to receive requests, it advertises the network addresses of the services it offers. On the client side, the operation is symmetric since it needs to determine the network address of the requested service. Traditionally in distributed systems, the directory/name service contains information that go beyond the sole network address: additional information may be stored so that the client can be more selective.

- How to interact with the service ?
The interaction with the service goes beyond the message exchange and nowadays the RPC and the method invocation on remote objects appear as de facto standards. This evolution is rather natural when we consider all the potential dangers of the message passing approach. The new paradigms rely on the network message but this is done transparently for the programmer.
- How do we provide security ?
Putting data and control without any security concerns makes them vulnerable and accessible to outside or inside intruders. Security reconstructs the protection offered by the operating system that cannot be extended to the network and as such must be part of any distributed application. Security issues can be highly complex due to the underlying mathematical background and very simple for the user who can specify the levels he/she desires.
- How to handle heterogeneity ?
Heterogeneity in a distributed system can be present at any level: from the very nature of the processing element, to the compiler through the operating system. When a client and a server interacts they have to go beyond heterogeneity so that communication/cooperation can take place.
- How do we make it scalable ?
This issue is of importance when the number of processing elements becomes high and when the various elements of the distributed application needs to interact very often. This issue plays a special role in the context of an information system at the enterprise level or in the context of electronic commerce. The software itself can take into account this load by replicating service explicitly. Nevertheless, "system" level services need to take into account this factor: a name service acts as a database server and if it was unique, this would cause bottleneck and congestion for the entire application.
- How do we deal with distributed management issues ?
Even when composed of only two processing elements, a distributed system has to be managed and supervised. This issue is even of more importance when thousands of equipments are connected together. For example, it is almost impracticable to have all the distributed application code generated all at once, clients and servers may be generated and activated at any time. Even if the reliability of any computing equipment is rather high, the probability of a failed equipment in a large distributed system makes it necessary to provide tools and services to reconfigure the network and the application itself: i.e.; to relocate a server on another machine, to replicate a server if clients are too numerous. All the operations have to be done while part of the application is running. Moreover, when the application has been

running over some period of time, feedback and analysis may indicate the need for tuning: servers may have to be relocated, their security parameters may have to be changed, the consistency model for ensuring coherent replicas of the name space may be inadequate, etc...

OSF DCE, OMG CORBA and Ada 95 proposes responses to these issues. Ada 95, as a programming language, does not provide elements for managing distributed systems. We believe that the most complete response is available with OSF DCE: this can be seen with the documentation associated with any of the DCE services. A lot of situations have been taken into account and DCE provides facilities to configure the service so that to fulfill the specific needs of any user. This exhaustiveness has a cost and the size of the complete DCE documentation may act as a repulsive for any beginner.

Ada 95 is also unique because there is no need to bring in extensions to deal with distributed features. Moreover, Ada is to be used for the interface definition of servers and there is no extra IDL language. This contributes to the easiness of programming and removes the necessity of complex mappings/bindings between the implementation language and the interface definition language.

4 Software engineering issues

Distributed system programming differs from centralized system programming because of the nature of the distributed target and the lack of an implicit global memory / global state. Thus elaborated forms of cooperation and coordination have to be used.

4.1 Understanding the concepts

This point is related to the ease of understanding the rationale behind the application and the distributed infrastructure. This issue plays a major role for the designer since errors can be introduced when the concepts are difficult to understand or when the parameters of services are rather obscure or complex.

OSF DCE fits well in the complex platform definition because it combines various tricky components and interfaces. Let us consider alone the POSIX 1003.4 interface for concurrent programming: the parameters to be used when creating a thread are numerous and many values have defaults. One of the parameter of *Pthread_Create* contains an attribute value which can only be set by a group of subprograms (e.g., to set the priority, various scheduling parameters). This complexity at the call level is also reinforced when such calls have to be made in sequence to obtain a desired operation. Registering a server and making it ready to receive calls from clients require numerous calls whose semantics / behavior are far from natural and immediate. In such a situation, the programmer has almost no choice and uses the copy/paste approach: you take an example provided by the constructor or by a colleague, you copy sequences of it and integrate them in your application. If it worked in the previous context, everything should be fine in the new context!

This practice is still in use even in large and critical projects but the potential dangers are numerous.

OMG CORBA on the other side claims simplicity and this is really the case for the specifications but implementors have to more specific and the complexity is moved on the implementation and on the use of the platforms. And one can guess that the classical copy/past of code sequence is still in use.

On the other side, Ada 95 presents an elegant and clear definition of distribution. The programmer can concentrate on his/her application and does not need to decorate his/her code with ciphered systems calls. In fact, the situation for distributed system programming is very similar to using application-level tasks vs. programming with a thread library: it is a compiler task to place the right calls in the right place while the programmer limits his/her role to indicate the behavior/property he/she wants. This features makes the annex directly usable even by beginners and therefore Ada should play a major role in labs devoted to distributed systems.

4.2 Code portability

Early criticisms of CORBA were centered on the low detail of the ORB specifications. CORBA itself indicates a common and basic interface and implementors feel free to implement the OMG specifications according to their views or add ad hoc tools to ease the programming of distributed applications. Martin Libicki wrote of CORBA: "OMG deliberately chose to standardize practices and syntaxes and abjures specifying the fine details of every sort of objects. The result may be guidelines so loosely defined that conformance may leave applications generally unable to trade objects without considerably more hand-tooling". This remark can be easily verified by the current proliferation on platform specific extensions: you do not develop an CORBA application but an ORBIX (or any other vendor) application (see for example, [Baker, 1997]). This situation leads to many incompatible implementations of the specification and also to complex protocols so that ORB can interact even if client and server do not use the same CORBA platform. From experience in network, we know that adding an extra layer to adapt data generates costs and is a rather time consuming operation.

Nevertheless, portability of a distributed application using CORBA is not reached: you cannot take your code written for platform X and make it run straight on platform Y.

Portability for OSF DCE is immediate: the same code was running for the various servers and controllers. There still could exist incompatibilities linked to the different compilers used (we know that all compilers have bugs!) but we can assume their number to be reduced. This portability can be seen when running the same source code on various machines, operating systems or platforms obtained from different vendors.

Portability for Ada 95 is rather difficult to estimate. GLADE is available on numerous platforms and the same source code can be adapted on any of them. Portability across multi-vendors platforms has not yet been investigated because of the lack of competitors for DSA. The annex also specifies the interface between the compiler and the Partition Communication Subsystem (PCS). The availability of such an interface also makes a distributed Ada application

portable. The user can switch easily from one PCS implementation to another and select the one that is better fitted to his/her criteria.

4.3 Checking consistency of the application

A centralized application requires consistency between the various files and this operation is done either by the programmer himself (i.e.; through a makefile) or with the help of the language (e.g.; Ada) which makes it impossible to generate an executable file if the various files refers to different sources. At least the entire application uses the same resources and refers to the same values and data. Ensuring the consistency of the entire distributed application is very important because the erroneous situations that may be triggered by the configuration are very hard to understand and detect. Moreover, clients and servers executable codes may be generated at different times and therefore we need to ensure the consistency of multiple targets. Solutions to this problem can be done at compile/bind time with the help of Makefile. Nevertheless, the situation can be more complex as indicated in the following situation. Suppose a server using a resource R of version V1, the server is compiled and then started ; later resource R is modified and version V2 is now available ; a client uses resource R (with version V2) and is compiled and started. This example indicates that the consistency checking is not only a compile/bind time operation and requires dynamic checking.

In OMG CORBA, no reference to the version exists and we can believe it is on the programmer's responsibility to take care of this consistency issue. The newest release of the CORBA specifications may have addressed consistency but we are not aware of the work in progress. In OSF DCE, version checking has been recognized from the very beginning. A version attribute is associated to any interface. This attribute is composed of two numbers: a major and a minor. When a client and a server interacts, a run-time check verifies that both use the same major number for the interface. Otherwise, there is an error. The existence of the minor number has been decided to take into account small changes in the interface (e.g., adding comments or fixing typographic errors) which do not jeopardize global consistency.

For Ada 95 DSA, see E.3 (6), the reference manual defines consistency in the following terms: "In a distributed program, a library unit is consistent if the same version of its declaration is used throughout. It is a bounded error to elaborate a partition of a distributed program that contains a a compilation unit that depends on a different version of the declaration of a shared passive or RCI library unit...". The annex also introduces a 'Version attribute to deal with the run-time consistency of the distributed program. It is therefore impossible to have an application whose partitions use different versions of the same specification. This rule is a continuation of the compilation rules applied to single Ada programs.

4.4 Configuring and taking distribution into account

When using OSF DCE or OMG CORBA, distribution has to be taken into account at the early stages of the design of the application. The designer has to define the interfaces of services that may be called from remote. This early decision has an enormous impact and has to be done very carefully and any change challenges the global software architecture

and its efficiency. This situation can be explained by the clear distinction between the interface definition language and the implementation language.

On the other side, Ada 95 is used to define the interfaces. The use of configuration pragmas restricts the structure of the program since rules control the chain of withed units and what can appear in the categorized unit. If we ignore this limitation, it is quite possible to take into account distribution after the coding of the entire application. GNATDIST makes it possible to configure the distributed application without recompiling the entire application for every tested configuration. This late decision on what to distribute and how to configure presents several advantages for the life cycle of the software. For example, the global organization may be changed to take into account evolutions in the hardware or in the users' needs without modifying the source code ... thus avoiding the introduction of errors.

4.5 Debugging

In the previous sections of this paper, we have seen that distributed objects and RPC make life simpler for programmers when building distributed objects. The distributed environment provides facilities and mechanisms and allows programmers to deal with remote objects as if they were local. Unfortunately, these environments do not prevent errors at various levels of the application: in the design or in the implementation. Moreover, the errors which could be found in centralized systems are also present in distributed systems and are also extended with more elaborated errors (e.g.; deadlock, termination detection, incorrect synchronization, lack of cooperation when sharing resources).

An initial document produced by the PVM (Parallel Virtual Machine) team spotted the difficulty when debugging the distributed application. Their proposal was rather simple and implied 3 sequential steps:

1. Run program as a single task to eliminate computational errors.
2. Run program in parallel on a single host to eliminate errors in message passing
3. Run program on a few workstations to look for problems with synchronization and deadlock.

This very simple model may help detect erroneous situations but the three steps do not involve the same source code at all. Therefore, one cannot assume that because we are debugging at level 2 that no more computational errors remains. In fact, the introduction of concurrency may have introduced additional errors since the source code has been modified in depth.

OSF DCE remains rather obscure on the debugging issue and only [Knouse, 1996] briefly mentions hints. We shall present them very briefly and the interested reader may find the complete approach in Annex A of [Knouse, 1996] :

1. Use loopback on one test system.
2. Use separate windows for each client and server.
3. Use a connection-oriented RPC protocol.
4. Set breakpoints at operation manager entry points.

5. In Xdb, turn off signal interrupts and Xdb does not support threads well. Or use `dde` for thread awareness.

For OSF DCE the debugging strategy is indicated in about one page which is rather limited and very dense at the same time. The use of separate windows for each client and server makes it almost impossible to use the debuggers when the number of applications chunks is greater than 3 or 4. Moreover, the author suggests the use of connection-oriented protocol to avoid time out from a client when waiting for a datagram response from a server.

The strategy for debugging CORBA applications is not very explicit and even in [Baker, 1997] no reference is given for this task. We may think that the approach selected for OSF DCE remains the same.

Ada 95 presents an integrated approach where the various aspects of the distribution are extended from the sequential features: e.g.; classic subprogram call vs. remote subprogram call, dispatching vs. remote dispatching, access to subprogram vs. remote access to subprogram. Except for limited and specific constructions, Ada incorporates facilities for programming distributed systems as a consistent and systematic extension to those provided for programming non-distributed systems. Thus the benefits of a type-safe object-oriented programming language are made available and contribute to the early detection of numerous errors (at compile time). Moreover, the same Ada code can be run either in a distributed way or in a non-distributed way and this makes it possible to use with benefits the non-distributed debugging environment. Unfortunately, not all errors can be detected using this approach and tricky situations will not be present when running in a non distributed context: e.g.; a distributed server which offers RPC services will include tasks (e.g.; agent tasks) for servicing the clients while its execution in a non distributed context will be only a classical subprogram call.

5 Why CORBA is successful and why Ada DSA is not ?

This sentence may appear a little bit too strong. We do not compare here the technical merits of both approaches but we place the discussion on the commercial side. Nevertheless, OMG has been very successful while promoting CORBA and this is not the case with Ada 95.

From the very beginning, OMG has included in the design loop potential users and vendors. From a group of 10 institutions or companies, OMG has successfully attracted more than 700 nowadays. Even if we can question the effectiveness of the cooperative work with a group of this size, the result is that individuals and companies are proud to contribute to CORBA and they let the world know it. Being a member of OMG acts as de facto reference for prospective clients and as good reference for the dynamism of a company. And we know the auto-catalytic effect! For Ada 95, the design process of the new release of the language was rather confidential and has certainly led to successfully release an ISO approved standard. But even within the Ada community, people or companies do not put forward their contribution in a commercial-marketing approach we can find for OMG contributors.

OMG is also successful because it addresses issues and concerns of users. On the OMG web server (see <http://www.omg.org/corba/whatiscorba.html>), we can read: "CORBA is a signal step on the road to object-oriented standardization and interoperability. With CORBA, users gain access to information transparently, without them having to know what software or hardware platform it resides on or where it is located on an enterprises' network. The communications heart of object-oriented systems, CORBA brings true interoperability to today's computing environment."

Annex E of Ada 95 reference manual does not reference at all what are the benefits of the annex. For the first sentence, we have: "This annex defines facilities for supporting the implementation of distributed systems using multiple partitions working cooperatively as part of a single program". And the rest of the section presents limitations and restrictions on what the programmer cannot do. The attraction to the annex is far from being immediate in such a context.

OMG has a complete plan on how OMG CORBA architecture will look like when all the specifications / components will be made available or delivered. In CORBA, we can find, for example, services which are essential for implementing objects. In the list, we can find for example:

- A concurrency control service that protects the integrity of an object data and state when concurrent requests have to be serviced.
- A persistent object service that supports the persistence of an object data and state when the object is no longer present in memory.
- A transaction service which enforces the principles of atomicity, isolation, and durability in the context of operations to several objects.
- A security service that will provide various levels of security for the distributed applications: e.g.; authorization, confidentiality, integrity, digital signature.
- An event service that supports the notification of waited events.

These services can be considered as responses to users' expectations. As mentioned earlier, most of the distributed platforms provides such a set of services and tools. Nevertheless, OMG went beyond this common services with the notion of facilities, which aim at building applications across a wide range of application domains. CORBA proposes horizontal facilities which are user oriented and vertical facilities which are domain specific. Horizontal facilities are grouped and are composed of numerous entities:

- User interface facilities
- Information management facilities
- System management facilities
- and task management facilities.

The vertical facilities have been identified by OMG after a complete evaluation of the application domains from users' expression on interests (OMG RFI). The domains selected cover most of the potential applications domains for which distribution is a must:

- Accounting,

- Application development,
- Distributed simulation,
- Imagery,
- Information superhighways,
- Security,
- Currency,
- Computer integrated manufacturing, etc.

Work is still in progress for these facilities and obtaining a consensus on these specifications is a rather daunting task. Nevertheless, these efforts have very positive impacts on the community and they contribute effectively to the dissemination of the merits of CORBA:

- Work is being carried at the user level and the application features are taken into account with their domain specificities. The technical consideration related to the platform and other technical information is kept away.
- The work is collaborative and people are working together (and anyone is welcome to contribute). The minutes of the meetings and the the various forums can be downloaded and read via the web.

Unfortunately, this level of cooperation is not so well developed for DSA. Working groups do exist in the Ada community and their output is extremely valuable: e.g., IRTAW (International Real-Time Ada Workshop) (see <http://www.cs.fsu.edu/~baker/irtaw99/>) for a brief overview of the technical contents of the last workshop. Annex implementors (I hope we shall use someday the plural) may consider that CORBA services and facilities are already available from the Ada language, or have been implemented already, or that their implementation is trivial, or that there is no emergency in implementing them because there is no direct request from users. L. Pautet and his team have already started this effort of explanation and promotion through tutorials and users' manuals. But the overall efforts go much beyond the possibilities of even a skilled and motivated team. We believe the Ada community should go beyond and envisions a complete distributed platform which takes into account users' needs. It is a community effort and could be the last chance to move DSA away from confidentiality.

6 Conclusions

OMG CORBA had an enormous impact on the computer science community because of the new concept of distributed object and interoperability. CORBA has benefited of a large consensus of vendors and software designers because most of those players could contribute in the standard. In fact, the OMG approach reinforces the contribution of software vendors who need to contribute in order to keep track of the ongoing work. The results appear nevertheless as a minimal agreement between all the approaches and no strong orientation can be seen.

As an alternative solution, Ada 95 appears rather weak both in the Ada community and outside. The merits and advantages of the integrated approach proposed by Ada are outstanding and can address most of the issues related to distributed system programming. To make designers and programmers aware of the Ada 95 approach, we propose the following steps:

1. The Ada community should promote the possibilities of the annex to the outside and inside. DSA is certainly limited and many required features are not defined but this cannot preclude any extension to be proposed or implemented. This could be achieved with success stories and experience feedbacks.
2. Vendors of Ada environments should offer the distributed system annex to their customers. Up to now, GNAT is the only system which makes it available on various platforms. The existence of competition could indicate that an Ada solution exists and a market. Moreover competition could boost the development of environments to fulfill the users' requirements. The lack of multiple implementations of the distributed system annex prevents existing projects to use the new paradigms of Ada 95: for long term running projects, the transition from environment X to Y generates costs and also delays.
3. Promotion of future or in-progress efforts. The announcement of future releases or current efforts in progress has a double impact: it shows that things are still moving and that implementors believe in the annex; and it smoothes angers of customers who wait for a specific service (this last technique is well known by many vendors even outside the computer industry!).
4. Ada as unique features which makes it possible to address information system, real time systems and distributed systems. Moreover, the experiences in embedded systems have produced numerous high quality environments. The combination of technologies (e.g., real time distributed systems, distributed embedded systems) could be flagships for a users who are in need to reliable technologies for their critical / long life applications.
5. The role of the OMG committee has played a major role for CORBA in proposing enhancements and also starting discussions and interactions between the platform implementors and the applications users. IRTAW working group could be a valuable candidate to propose environments and extensions of the Ada model for distribution. But we believe that this group should have a larger audience and address in more details distributed system issues. Moreover, this work could be integrated if Ada 95 norm was revisited.
6. The Ada community should point out the inconsistencies of the CORBA model and its complexity for the programmer. ¹ CORBA users will be in the same situation as the C++ programmers a few years ago: puzzled by the difficulties and the flaws of the model, most went to Java. Java is once again a strong competitor with RMI.

¹We have made such a study in the context of OSF DCE. The results of the evaluation of the platform showed severe flaws and errors when services are combined [Gourhand and Kermarrec, 1997]. This evaluation showed the limits of integrating pieces of software from different origins.

References

- [Baker, 1997] Baker, S. (1997). *CORBA distributed objects using ORBIX*. ACM Press and Addison Wesley.
- [DEC et al., 1995a] DEC, HP, and et al. (1995a). The common object request broker : architecture and specification. Technical report, Object Management Group and X Open.
- [DEC et al., 1995b] DEC, HP, and et al. (1995b). CORBAServices: Common object services specification. Technical Report OMG 95-3-31, Object Management Group and X Open.
- [Gargaro et al., 1996] Gargaro, A., Kermarrec, Y., Nana, L., Pautet, L., Smith, G., Tardieu, S., Thierault, R., and Volz, R. (1996). ADEPT technical report: Ada 95 distributed execution and partitioning toolset for GNAT. Technical report, Texas A&M University. (<http://www.cs.tamu.edu/research/ADEPT/>).
- [Gargaro et al., 1995a] Gargaro, A., Kermarrec, Y., Pautet, L., and Tardieu, S. (1995a). PARIS : Partitionned Ada for Remotely Invoked Services. In Eurospace, editor, *Ada Europe Conference, Franckfurt Germany*, Heidelberg. CNES and European Space Agency, Lectures Notes in Computer Sciences.
- [Gargaro et al., 1995b] Gargaro, A., Kermarrec, Y., Pautet, L., and Volz, R. (1995b). ADEPT : Ada Distributed Execution and Partitioning Tools. Tutorial, ACM Conference, Anaheim, California.
- [Gourhand and Kermarrec, 1997] Gourhand, Y. and Kermarrec, Y. (1997). Evaluation de l'interactions de services dans osf dce. In *Electronic Journal on Networks and Distributed Processing, Proceedings of NOTERE (Nouvelles Technologies de la Repartition)*, Pau, France. http://rerir.univ-pau.fr/english/conference_proceedings.html.
- [Intermetrics, 1995] Intermetrics (1995). *Ada 95 reference manual*. International Standard ANSI / ISO / IEC-8652:1995.
- [J Siegel et al., 1996] J Siegel et al. (1996). *CORBA fundamentals and programming*. OMG Press.
- [Kermarrec et al., 1996] Kermarrec, Y., Nana, L., Pautet, L., and Tardieu, S. (1996). GNATDIST : a configuration language for distributed Ada 95 applications. In ACM, editor, *ACM Tri Ada conference*, Philadelphia, Pa.
- [Kermarrec et al., 1995] Kermarrec, Y., Pautet, L., and Schonberg, E. (1995). 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.
- [Knouse, 1996] Knouse, C. (1996). *Practical DCE programming*. HP professional books. Prentice Hall.
- [Rosenberry et al., 1993] Rosenberry, W., Kenney, D., and Fisher, G. (1993). *Understanding DCE*. O'Reilly and associates, inc.
- [Shirley, 1992] Shirley, J. (1992). *Guide to writing DCE applications*. O'Reilly & Associates, Inc.

- [Volz et al., 1997] Volz, R., Gargaro, A., Smith, G., Thierault, R., and Waldrop, R. (1997). Future directions in Ada - distributed execution and heterogeneous language interoperability toolsets. In *8th International Real-Time Ada Workshop*, Ravenscar, UK.
- [Volz et al., 1995] Volz, R., Thierault, R., Kermarrec, Y., Pautet, L., Tardieu, S., and Smith, G. (1995). Ada 95 distribution annex implementation for GNAT. Technical report, Texas A&M University, College Station, Texas.