CORBA vs. Ada 95 DSA
A programmer's view

Yvon Kermarrec

ENST Bretagne
Département IASC
29285 Brest Cedex
France

Yvon.kermarrec@enst-bretagne.fr

1 Context of the study

This paper analyses and presents discrepancies between the Ada 95 model for distribution and CORBA [DEC et al., 1995a] [J Siegel et al., 1996]. The aim of this paper is not to make a complete evaluation of both approaches but to take the view of the programmer: that is how to build a distributed program from the existing tool box and what are the perspectives once a technical choice is done for a platform.

Many people perceive CORBA 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.

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 from the Ada community know about the potentials of DSA) and the annex was about to be thrown away from the final 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 Ada solution.

We believe that DSA [Kermarrec et al., 1995] [Gargaro et al., 1995] has a better approach of distributed systems than CORBA and other middlewares 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 Ada (e.g., asynchronous transfer of control, tasking, memory control, subprogram access, dispatching). 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 around a few issues that we analyze. In the first section, we describe the initial contact by a beginner with 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 address the issue of portability which constitutes a major element for ensuring independence from a vendor or a specific environment. Next we present how to locate a server on both approaches and, then, the way to indicate which interactions can be addressed to a server. The last section describes extensions and additional services that can be offered to assist the programmer with well known services. As a conclusion, we give a few hints to develop the use and the on-going activities around Ada and distributed systems.

2 A programmer's first 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 multithreaded server, for example.

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 and integrating 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. The management of a DCE administrative unit (a cell) requires admin privileges and the expertise of a system administrator. Fortunately, the configuration decisions can be modified and tuned based on feedbacks and initial measurements. On the other side, CORBA looks simple but the resolution of even simple problems requires an implementation specific expertise. To summarize, CORBA has reached to transpose the problems so that they have to be solved at the implementor level (see [Baker, 1997] to know more on the way Iona implements CORBA).

Ada 95 and its DSA looks 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 and 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 simple the Ada 95 language. The reference manual goes even on to 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 the types 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 for distribution: the Remote Procedure Call is nothing else than a subprogram call, a distributed object is very similar to any other object, and the dynamic RPC (Remote Access to subprograms) is well understood through subprogram pointers. GLADE and its configuration language (GNATDIST [Kermarrec et al., 1996]) makes the transition 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.

3 Code portability

Early criticism 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 [Baker, 1997] (or any other vendor) application.

The current implementations of CORBA can indicate a conformance to the current standard and this claim is very legitimate because of the current wholes and imprecise points of the standards. This generates confusion as the various vendors offer products according to their current understanding of the standard. This certainly goes against the portability of the application code and this indicates a failure in the role of CORBA as a middleware layer. One of the role of the middleware, or the enabling technology, is to avoid dependencies between the application code and any of the operating system resource by defining an adequate and complete API.

In fact, ORB vendors can be considered as satisfied by this current status. Once a project has selected a given ORB, the project is hooked because it represents a transition cost to move from one solution to an alternative one.

In this context, OSF DCE is much more successful than OMG CORBA. DCE has defined the complete API between the application code and the DCE layer.

OSF followed current standardization efforts to ensure the future of the software developments. For example, the thread service of DCE follows the POSIX 1003.4 initiative. This is also true for any of the other services offered by DCE. Based on our experience, we have been able to compile and run application designed for PCs (with an IBM version of DCE) on a set of workstations without modifying any line of code (with an HP version of DCE).

The Ada 95 approach guarantees portability of the application code because the configuration of the distributed application and the mapping of fragments onto the various elements of the distributed system is outside the language itself. These operations are post compilation issues and are kept separate and outside of Ada 95 scope. This makes it possible to investigate different configuration models without modifying the application itself and to select the one that fits the best. Another side benefit relies in the debugging approach because in most cases a distributed application can be run on a distributed platform and then moved smoothly to a distributed context.

The designers of the annex even went further for the specification of a completely portable environment by defining the API for the PCS (Partition Communication Subsystem and System.RPC. This effort is more controversial. On the one hand, it defines an API to reach the communication layer (the PCS) thus making it possible to switch from one implementation to another one (or to more from one version to another one smoothly). But, on the other side, this interface is too minimal to be complete and therefore any implementation must extend it and thus jeopardize the initial intends.

In contrast to CORBA, a distributed Ada application is portable across platforms and vendors. This argument should be verified even when other implementors will develop solutions and products for DSA. This portability cannot be guaranteed for CORBA because of its nature: implementors have to define their own extensions or their interface.

4 How to locate a service

The point is to determine how to reach the requested server. For classical RPC, we have the notion of binding which consists in associating a network address to a service or a server. In contrast to normal subprogram call, this operation cannot be done at compile (or link) time and can be determined at execution time only: the network address contains the IP address for example of the machine on which the server is running. The binding process can be well understood when using DCE: the programmer needs to follow predefined steps to bind if the stored address is not fully resolved: e.g., determining the host name of the server, contacting the end point mapper, and then contacting the application server itself.

In CORBA, the naming service [DEC et al., 1995b] is outside the core specifications. A server registers itself with the naming service and then clients will inquire about its address. This CORBA service introduces the notion of context and a set of API to access and manage the name service. Interestingly, these services do not seem to be sufficient and for example ORBIX defines its own additional service *_bind*. All entries of this API takes a name (the object to be reached) and a context as parameters. This raises the major issue of dealing with application consistency. In fact, one server can implement an interface (version v1) and a client can request an object (interface version v2). In CORBA, there is no predefined check to determine that both client and server will use the same version of an interface. One way to ensure consistency of the application would be through *Makefile* but this is on the programmer's responsibility. The presence of such an inconsistency will generate various troubles and these situations are very difficult to detect when testing. OSF DCE proposes an almost complete consistency check: a certain flexibility is given to deal with minor changes of the interface which do not impact with the interface itself (e.g., fixing typo, adding constant declarations).

The context for DSA is a little bit different but remains similar. An Ada client accesses a server with a name (e.g., a package name which has properties and restrictions). The current implementation of GLADE implements a name server to be able to locate a service and to deal with Partition_Id attribute. Consistency rules of the entire application must be verified and we are in a situation comparable to dealing with consistency at the library level (compilation rule of withed units, for example). The reference manual imposes that the consistency check is done and this will prevent inconsistent interactions between a server and a client. This check can be done at run time level but might be done if the configuration process is static. This implicit name service does not prevent any implementation of annex E to offer a more elaborate environment: a kind of data base of object references with an API to insert/extract references.

5 How to define and to talk to a service

OMG CORBA requires that an interface be written in a specific language which deals only with interface definitions : OMG IDL. For numerous historical reasons, OMG IDL is very similar to C++ and the syntax of both languages are closely related. The definition of an interface consists in indicating the methods, parameters and attributes of objects. The needs of such an IDL has been recognized by early version of RPCs. The interface will be used by the clients and the servers: the client will know how to address a request to the object server (and this presents a required information when it comes to dynamic interfaces calls) ; and a server will have to implement the various services as defined in the interface. The clear distinction of the implementation language and of the interface language requires mapping between any pair of them. As a secondary advantage, the IDL language appears as programming language which its own syntax, semantics, and functionalities. At the first sight, this point might appear as minor but, in the context of Ada 95, it will help us to understand the negative impact attached to Ada 95 IDL. The distinction between the implementation language and the IDL generates troubles because a mapping (a correspondence) has to be established between both models: the involved issues can be complex above all if the implementation language does not support objects, exceptions,... Moreover, the programmer needs to be aware of the possibility of 2 distinct inheritance hierarchies: the interface hierarchy and the implementation side hierarchy.

The distributed system annex makes it possible to define an interface in Ada 95 itself thus avoiding the need of an ad hoc language. The Ada IDL can be considered as a subset of Ada 95 and this generates confusion. In fact, Ada 95 as an IDL is in the same situation as C++ and OMG IDL: an implementation language vs. an interface definition language. Therefore, the annex restricts what can appear in an interface (the Ada IDL is a subset of Ada) and the wording use negative and "don't" terms which are not encouraging at the first sight. Nevertheless, the use of an Ada solution for both the interface and the implementation avoids complexity when putting them together. This selection precludes inter-operation with other languages because it is quite certain that no other product will use Ada 95 as the IDL.

6 How to provide additional services

The core specifications of CORBA are too minimal and once again they are not sufficient to build a distributed application. OMG has therefore defined additional specifications for common services [DEC et al., 1995b]: they deal with persistence, security, etc. CORBA defines also other components for specific applications.

This situation is very similar to what is available in OSF DCE. But once again, OSF went beyond specifications by providing tools and a complete interface available on any DCE compliant platform. The level of detail is too fine grained for most of the usages and gives enough material to write books. Nevertheless, they constitute an interesting base of reference of what should be made available to applications. Moreover, these services have been successfully implemented: for example, the consistency model between replicas of the name services highlights the issues when node failure may occur and we still need to maintain consistency between the data at any time. The management of these services is rather difficult and are on the responsibility of the cell administrator. OSF tries to define DME (Distributed Management Environment) to give some form of genericity when managing and configuring various parameters of the configuration.

DSA is minimal and encourages in some ways the extension of the environment with additional services. Unfortunately, there is no join work nor agreement on what should be made available. GLADE implements very useful services for security (e.g., filters [Pautet and Wolf, 1997]); fault tolerance and replication has been investigated [de las Heras-Quiros et al., 1997], etc. But these efforts are from individuals and not from a working committee and are done by a unique team (the implementors of GLADE). The existence of additional services is to be considered as a major selection criterion and the lack of a well defined (or in progress) set of services similar to the ones proposed by CORBA or DCE discredits certainly DSA. Moreover, the existence of such a committee will give a clear signal to the community that things are moving around Ada too.

Among the possible services, we can reference:

- A name server or trader which functionalities could be similar to the approach offered by DCE or CORBA.
- Persistence (of the date or a server state).
- Security and privileges services. Kerberos could be a good reference in this domain.
- Time service to guarantee that clocks of the various processing elements are synchronized and that event can be distinguished by their occurrence date.
- Fault tolerance and reconfiguration facilities: those services could be called by the programmer (e.g., transaction, duplication) or triggered by the administrator of the distributed platform (e.g., consistency of replica).
- Administration tools and services to configure the system and to obtain various feedbacks.

7 Conclusions

OMG CORBA had, and continue to have, 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. Moreover, the difficulty are moved to the implementors who extend the initial specifications with numerous services and API. Moreover, many issues around CORBA are still obscure (e.g., matching interface inheritance with implementation inheritance, version consistency, IDL data types).

As an alternative solution, Ada 95 appears today 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. The intend is to show how Ada can be used to solve classes of distributed system problems. Combined features of Ada could also be presented so that to address distributed real-time, or embedded applications.

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 too. Moreover competition could boost the development of environments to fulfill the users' requirements.

3. Promotion of future or in-progress efforts. The announcement of future releases or current efforts in progress around DSA 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. The Ada community should point out the inconsistencies of the CORBA model and its complexity for the programmer. 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.

5. A working group should define and propose a complete environment with services for distributed systems. 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 done with compiler vendors and also potential users so that to fit users' needs.

References

[Baker, 1997] Baker, S. (1997). *CORBA distributed objects using ORBIX*. ACM Press and Addison Wesley.

[de las Heras-Quiros et al., 1997] de las Heras-Quiros, P., Gonzales-Barahona, J., and Centeno-Gonzales, J. (1997). Programming distributed fault tolerant systems: The replicAda approach. In ACM, editor, *Tri Ada '97*, pages 21–30, St Louis. ACM.

[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., 1995] Gargaro, A., Kermarrec, Y., Pautet, L., and Tardieu, S. (1995). 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.

[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.

[Pautet and Wolf, 1997] Pautet, L. and Wolf, T. (1997). Transparent filtering of streams in GLADE. In ACM, editor, *Tri Ada '97*, pages 11–20, St Louis. ACM.

[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.