What future for the Distributed Systems Annex? *(extended abstract)*

Anonymous authors

Anonymous institute

Abstract. In this paper, we try to identify the strengths and the weaknesses of the Ada 95 Distributed Systems Annex from an user and implementor point of view. We will then propose some ideas to improve this annex immediately or for the next revision of the language (Ada0X), and present some extensions that have been implemented so far.

Note to the reviewers: this article is sent as an extended abstract, because some of the ideas exposed in this paper are being actively worked on. The final paper will thus contain more realizations and conclusions.

1 The Distributed Systems Annex today

Communication between entities can be found everywhere nowadays: more and more computers are linked together using the Internet, many companies now have an internal network (intranet), and multiprocessor boards are now affordable for personal use. More than the raw computing power, the ability to communicate with other systems is one of the strongest arguments when selling machines, environments or programming languages.

The architects of Ada 95 had understood it, and this lead to the inclusion of a Distributed Systems Annex (DSA in short) in the latest language revision [7]. This annex, while fully consistent with the rest of the language, defines how some packages can be called remotely, and how complex data structures such as pointers on remote objects and remote subprograms can be built and used without giving up the safety and the type checking that made Ada so popular in domains where people are in need for reliability.

The reader unfamiliar with the DSA must know that each part of an Ada distributed system is called a **partition**. Each partition contains zero or more packages and subprograms, and each partition may have its own main subprogram.

1.1 Existing implementations

There are currently two implementations of the DSA, both for the GNAT Ada compiler:

1. **GLADE** (GNAT Library for Ada Distributed Execution), developed and maintained jointly by the ENST[1] and by Ada Core Technologies[2]. This implementation is freely available under the same license as GNAT, and commercial support

[1] `http://www.enst.fr/`
[2] `http://www.gnat.com/` and `http://www.act-europe.fr/`

is available through Ada Core Technologies. It includes a partitioning tool called **gnatdist** [9] as well as a partition communication subsystem called **garlic** [10].
2. **ADEPT** (Ada 95 Distributed Execution and Partitioning Toolset) has originally started as a joint project between Computer Sciences Corporation, the Texas A&M university and the ENST. It is based on an early implementation of GLADE [6], and has since evolved into a bridge between Ada and RMI (see section 3.3). It is now maintained by the Texas A&M university[3].

1.2 Typical uses of the DSA

The data located in this section is far from complete: it is solely based on publicly available information that went through public newsgroups and mailing-lists, unless otherwise specified.
Note to the reviewers: authorizations to include company informations are pending, and more specific examples will be included in the final paper.

Industry GLADE has been successfully used by several major companies, both in the US and in Europe. Of course, the acceptance of the distributed features of the language is bound to the acceptance of Ada 95 itself, and many industrial companies are still using Ada 83, even when using an Ada 95 compiler.

For example, a major electricity provider in France is currently experimenting the use of the DSA to serve as a session-tracker for their external WWW site, in order to keep track of who did what across separate WWW requests. Another project is to use GLADE as the heart of a caching server for an internal database accessed through the WWW.

Military Most military projects that were using Ada 83 had their own certified communication layer. It is thus often difficult to move to the DSA without rethinking the whole program architecture. However, GLADE is starting to show up in new developments, where distributed interactive simulations are involved.

For example, an implementation of RTI (Run Time Infrastructure), a normalized library accessible from Ada and C++ designed for inter-simulators communication, is being developed with GLADE acting as an easy-to-use communication layer. The DSA is hidden from the RTI programmer but is heavily used inside the library itself and constitutes indirectly the link between the simulators [4].

Education Ada has been used for a long time in software engineering classes, because of its high-level features such as genericity, strong-typing, encapsulation and tasking. It allowed students to learn programming and algorithmic without having to learn and use an error-prone unsafe language at the same time.

Ada 95 has increased the potential of the language in the field of computer science, by offering programming by extension via the tagged types and the child packages and distributed programming, using the DSA. From our own experience and from what has

[3] `http://www.tamu.edu/`

been written by students and professors on various public channels, the DSA is very appreciated while learning the basic concepts of distributed programming (remote procedure calls, distributed objects). For example, our students have been able to develop a complete messaging system based on distributed objects in a few hours, while it would have probably taken days if they had to use raw sockets and message passing.

2 Weaknesses of the current model

In spite of its quality and ease of use, the DSA has some limitations in its current form. We will present what we feel are the major ones.

2.1 Interoperability with other compilers

The Annex E of the Reference Manual describes the declaration of a system package `System.RPC` containing types and subprograms that must be used by the compiler in order to call the Partition Communication Subsystem (PCS). The goal of this interface was to allow a PCS to be used with any other Ada compiler implementing the compiler-specific part of the DSA, that is calling the PCS with the right arguments.

Unfortunately, the data exchanged between the compiler and the PCS are encapsulated into streams (in the Ada terminology) and contain compiler-dependent arrays of bytes. That means that even if two compilers were using the same PCS, they would not be able to talk to each other unless they agree on a common format for marshalling data. For an example, the first compiler could encode a predefined exception by storing an index pointing into its internal exceptions table in the stream, while the other one would wait for the full name of the exception being stored in the stream instead.

Even using a compiler from the same vendor on heterogeneous systems does not guarantee that your computers will understand each other, as one may use integers stored in higher-order byte first while the other will store them in lower-order byte first.

2.2 Interoperability with other languages

One of the strengths of CORBA [13] is that it allows various parts of a distributed system to be coded using different programming languages. The interfaces between the actors are described using an Interface Description Language (IDL) and are then translated into the language of your choice (a full comparison of CORBA *vs.* DSA can be found in [14]).

The lack of normalization of the protocol used to communicate between the partitions of a distributed Ada program forbids the development of any portable binding with remote Ada services. On the other hand, this allows distributed Ada programs to be more optimized as the strong typing is preserved all the path along, while an interface with foreign languages would require additional checks to ensure the validity of externally acquired data.

We are currently facing a situation where people have developed two-headed distributed programs: all the data exchanges between two Ada partitions are made through the DSA, while CORBA is used to export the corresponding services to the outside world. However, this way of doing things is costly because two interfaces (Ada and CORBA) need to be maintained and error-prone, as a mismatch between the two versions may be hard to detect. We will propose a solution to this particular problem in section 3.2.

Also, another approach for offering inter-language interoperability is to write small wrappers in Ada making use of the DSA, while the foreign code only calls those wrappers. Although this approach can be used in some specific situations, it reduces the list of the usable architectures to the ones targeted by an Ada compiler supporting the DSA.

2.3 Dynamicity of a distributed system

Today's networks are evolving so fast that the designers of distributed systems cannot ignore the scalability issues. A system based on a client/server model must envision an ever-growing number of clients. We have seen in the past few years some WWW sites being shut down because they were unable to accommodate the load caused by their popularity.

Ada distributed programs may suffer from the same problem. For example, each partition in a distributed system is given a unique identifier, called a `Partition_ID`. This identifier is of an integer based type, and must not be reused by different clients. That means that as high as the high bound of this type can be, there will be an arbitrary limit to the number of clients that will be able to connect one after the other to a server. Concretely, that means that an ever-running server coded with the DSA will reach this limit one time or the other. While this limit may be correctly adjusted for today's typical needs, it may become ridiculously low in the near future.

3 Some proposals to extend the DSA

In this section, we will propose some extensions that could be applied today, and some others that could be part of the next revision of the Ada standard.

3.1 Normalization of layers

The DSA may be seen as three independent layers of a stack.

The high-level layer This layer contains the spirit of the DSA; it consists into the description, at the Ada language level, of what can be distributed and the semantics of every remote operation. The three categorization pragmas solely dedicated to the DSA (`Remote_Call_Interface`, `Remote_Types` and `Shared_Passive`) open a large and coherent set of possibilities to distribute Ada objects.

The set of entities that can be called remotely could be slightly enlarged by introducing the notion of remote rendez-vous for example, but the whole philosophy of the DSA would be kept unchanged. However, one extra feature that could be described in this high-level layer could be the dynamic invocation described in section 3.4, but we have not yet finalized the proposal for expressing this.

The mid-level layer What we call the mid-level layer here is the declaration of the `System.RPC` package. As written in section 2.1, this standardization started from the point of view that a partition communication subsystem could be used on any compiler since the interface between the compiler and the PCS was entirely contained this package.

While this definition allowed us to start quickly the implementation of GLADE because one part of the design was implicitly contained in the annex, we soon realized that the requirement to go through `System.RPC` for every remote call introduced a lot of constraints.

To take one example, the only non-opaque parameter to the procedure used to do a remote subprogram call (`Do_RPC`), is an integer denoting the remote entity. That implies that one of the two following methods is used:

1. this integer is assigned at partitioning time and the system is closed and static (it is not easy to add new clients in a client/server architecture after the first partitioning step while the server is running, and also not easy to launch several instances of a single client as they must have different identifiers)
2. this integer is computed at run time, and the compiler must have a way of retrieving it by talking to the partition communication subsystem using another interface than `System.RPC`, which defeats the capability of using the PCS with another Ada compiler

We strongly think that the standardization of the PCS interface is useless and should be removed from the next language revision. Moreover, it is the only case in the Reference Manual where something that can be considered internal to the compiler has been described in an authoritative way.

If we transpose this standardization into the CORBA world and see `System.RPC` as the ORB[4] interface, this would mean that CORBA stubs and skeletons could only call the ORB they are based on using a normalized interface.

The low-level layer This part is not mentioned in the annex and this is the cause of the trouble described in sections 2.1 and 2.2. A normalization of the communication protocol would open the road to interoperability with other systems, either written in Ada (and thus using the DSA themselves) or written in foreign languages, using additional libraries.

It is perfectly possible to use a well-defined protocol without giving up any safety and efficiency as long as the whole program is written in Ada. However, interfacing with other languages less safe than Ada may of course require the generation of additional checks to ensure that the externally acquired data meet Ada strong-typing constraints. A new pragma placed in the remote package declaration would direct the compiler to generate or not to generate those checks.

A proposal for such a low-level protocol is in progress, but is out of the scope of this paper. We plan to implement it for GNAT and GLADE, for compilers generating processor-specific code and for the future GNAT to Java compiler. This would, amongst

[4] Object Request Broker, the heart of a CORBA product

other things, allow some partitions of a distributed system to run in native form (typically a server) while some others would run on a Java virtual machine (such as client applets in a WWW browser).

3.2 Interfacing with CORBA

As written in section 2.2, nothing prevents a user from maintaining two consistent interfaces for a service, one for the Ada side of the world using the DSA and a second for the other languages using CORBA, although this is a painful and error-prone task.

One of our students has been developing, using ASIS [8], a tool that automates the creation of an IDL file starting from an Ada package categorized as `Remote_Types` or `Remote_Call_Interface`. The server side of the service will be automatically generated in Ada (using for example the AdaBroker CORBA implementation [1]), and thus immediately accessible from other languages. Of course, the mapping between the package declaration and the IDL file will not be trivial for complex constructs (discriminated records, overloaded primitive operations for distributed objects) but it will be formally defined.

We are also seriously considering the use of IIOP (Internet Inter-ORB Protocol) as a basis of our low-level protocol, to ease the interfacing process between the DSA, CORBA and RMI. The basic idea behind this is effort splitting: implementing the full tasking requires a lot of resources from Ada compilers vendors, so does the implementation of the DSA. If some of the costs could be shared by the CORBA and RMI vendors, there would probably be more implementations of the DSA, as the existing infrastructures could be reused easily.

3.3 Interfacing with RMI

RMI (Remote Method Invocation) is yet another way of writing distributed programs using the Java programming language. It offers the possibility of having objects located on different hosts to communicate with each other, but also let objects with their implementation (in Java byte code) move from one host to another. This very powerful feature of code migration is a big step towards the development of mobile agents.

RMI is very interesting for Ada users at several levels:

- Compilers that compile Ada code into Java byte code can use the RMI objects and libraries to build distributed applications
- A bridge between the DSA and RMI is fully functional (ADEPT, see section 1.1) and allows Java users to access Ada services and vice-versa
- Sun Microsystems (author of Java and RMI) is working with the OMG[5] to use IIOP as a basis for RMI implementation, while IIOP will be extended to support the full semantics of RMI. That means that Ada code compiled into Java byte code and using RMI will be able to talk with services written using CORBA, and that the Ada/RMI bridge will be usable as a gate between the DSA and CORBA

[5] Object Management Group, the official entity in charge of CORBA normalization

The use of IIOP as the standard protocol for the DSA would allow to access CORBA and RMI services directly through the DSA without any need for an additional bridge. On the other hand, it would make it possible, using an Ada to Java byte code compiler, to transfer active objects and achieve code migration in Ada using only the DSA. This would be a big step forward in terms of fault tolerance and reliability, as critical services could duplicate themselves automatically in order to keep for example n functioning instances at any time.

3.4 Dynamic interfaces

One powerful feature of CORBA not found in the DSA is the capacity of using dynamic interfaces. Two mechanisms, DII (Dynamic Interface Invocation) and DSI (Dynamic Skeleton Interface), allow to register a type by describing its methods using their names and signatures, and to build a call to those methods dynamically.

The most obvious advantage of doing this is that the interface needs not be present at compilation time. For example, a calculator application can be enriched at run time by adding new functions (as remote objects designed by their name as typed by the user) that are called dynamically on demand. Explicit static calls to those functions appear nowhere in the calculator source code.

The introduction of such a mechanism in the DSA would considerably ease the interfacing with CORBA, as both side could interface with each other using this protocol.

3.5 Quality of Service

On a completely unrelated topic, an useful extension to the existing DSA specification would be the introduction of Quality of Service (QoS) parameters. For the reader unfamiliar with this notion, the QoS may be seen as a numeric value quantifying some properties of the underlying network, for example the maximum delay between two hosts or the guaranteed bit rate from one point to another.

As the requests for quality of service depend heavily on the location of every service, it makes more sense to include all the QoS related information into the file describing the distributed application rather than in the packages themselves. To this purpose, we intend to propose the format of the gnatdist (our partitioning tool) command file as a standard for describing Ada distributed applications. This file format will first be extended to contain the necessary QoS extensions.

Network characteristics It is common knowledge that QoS and crude packet switching networks don't mix well [12]. Unfortunately, those networks represent the vast majority of what is available in the industry or in universities (TCP/IP over Ethernet).

The introduction of the new IP revision (IPv6, previously called IPng) [5] added the notion of traffic class. This field, present in every IPv6 packet, is used by all the routers on a given path to prioritize the flow accordingly to a predefined policy. That means that within an organization, it may be possible to setup all the routers to exchange data between two partitions at the highest possible priority, thus obtaining the speed and the bandwidth of the slowest physical link on the data path.

A solution more applicable to larger scale applications is the use of ATM (Asynchronous Transfer Mode) networks. Those networks can negotiate a guaranteed bandwidth for a complete virtual circuit [2]. Once the required bandwidth has been allocated, it will never be used for something else unless the resource has been explicitely released.

One of our students is working on interfacing the ATM libraries related to resource reservation with Ada for his graduate thesis. The goal is to offer the ability to use ATM as the underlying networking protocol for GLADE. We are working together to define a syntax for the indication of the desired networking resources, that could be applied to both IPv6, ATM and other protocols dealing with resource reservations such as RSVP [3].

Priority related enhancements One of the most needed enhancements of the DSA would be a specification of how the priorities of the different partitions in an Ada distributed program are related to each other. Right now, it is unspecified whether the priority of the caller will be used or not for executing the remote subprogram. This is even worse as the various partitions may have different scheduling policies and priorities ranges, thus leading to a malfunction if priorities are propagated over the network.

We plan to include new pragmas in the configuration files to describe statically the behavior of incoming calls in regard with priorities. These pragmas would limit the number of concurrent incoming calls made to a partition, and assign priorities to incoming remote calls according to the scheduling policy of the target partition, as well as a maximum priority for the tasks (if needed) used within the partition communication subsystem. This would also ease scheduling predictability using Rate Monotonic Scheduling techniques [16].

Note that it is already possible in GLADE to use a light runtime containing no task at all, provided that the partition contains only client code and no direct (using `Remote_Call_Interface` packages) or indirect (using distributed objects) server code.

Note to the reviewers: three other extensions will be proposed in the final paper, namely the replicated RCI packages with state saving, the possibility of defining a true distributed virtual shared memory using for example IP multicast and the capability of having forwarding objects to ease the development of fault-tolerant distributed applications in Ada.

4 Existing extensions

In this section, we will present very briefly the extensions that we made to the DSA in GLADE. Some of those extensions could be hopefully standardized in the next revision of the language.

4.1 Filters

Two partitions in an Ada distributed program built using GLADE can transparently exchange compressed and encrypted data, after each part has proven its identity to the other [15]. Applications manipulating sensible data can thus be distributed over a wide area network without any risk of confidential information disclosure.

This capability is built in the partitioning tool, and a programmer can build and use her own filters following a well-specified interface.

4.2 Services

One of the strong points of CORBA is the fact that the core architecture has a reduced size by design. Every additional service has been standardized in its IDL form, and may be available from several vendors.

Following this idea, we have implemented several services using the DSA, including:

- a naming service, whose goal is to let the user name distributed objects and organize them in a hierarchy of names (*à la* Unix)
- a distributed semaphore service, using [11]
- an events service, that let a user declare that she is interested in getting some kind of events and get them as they arrive

Some services (the naming service for example) have a very wide range of usages, and would benefit from a standardization and an inclusion in the standard libraries defined by the language. In the event of a unification of the low-level protocol with CORBA, we of course suggest that the pre-existing specifications be used if they suit Ada users needs.

4.3 Persistence

One feature of the DSA that is not commonly found in other distributed systems is the ability of sharing data between partitions, using `Shared_Passive` packages. Although this feature was designed with multiprocessor systems in mind, we have extended it in GNAT and GLADE in order to use the underlying file system. That means that data defined in such a package can persist after the lifetime of a program if the files in which they are stored are not erased.

4.4 Partition restarting

One capability needed by most GLADE users was the ability to stop and restart a partition, on the same host or on another one, in order to upgrade the service or to survive a network outage. This feature, when combined with the `Shared_Passive` packages, allows a partition to be stopped and then restarted without losing its state, thus offering some kind of hot restart well known in fault tolerant systems.

5 Conclusions

We have shown in this paper that the Distributed Systems Annex of Ada 95 is well defined and consistent with the rest of the language, but would benefit from a standardization of the protocol used to communicate between the partitions. In particular, it is possible to extend it to make it interoperable with other languages, without losing any of the Ada safety and security. We have also identified some new services and extensions that could be either taken in account in the design of Ada0X or included in the next version of the Reference Manual.

References

1. Fabien Azavant, Jean-Marie Cottin, Laurent Kubler, Vincent Niebel, and Sébastien Ponce. AdaBroker, using OmniORB2 from Ada. Technical report, ENST Paris, March 1999.
2. M. Borden and M. Garrett. Interoperation of Controlled-Load and Guaranteed-Service with ATM. Technical report, The Internet Society, August 1998. RFC 2381.
3. R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. Technical report, The Internet Society, September 1997. RFC 2205.
4. Dominique Cannazzi. yaRTI, an Ada 95 HLA Run Time Infrastructure. In *Proceedings of AdaEurope'99*, Santander, Spain, June 1999.
5. S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. Technical report, The Internet Society, December 1998. RFC 2460.
6. Anthony Gargaro, Yvon Kermarrec, Laurent Pautet, and Samuel Tardieu. PARIS: Partitionned Ada for Remotely Invoked Services. In *Proceedings of AdaEurope'95*, Frankfurt, Germany, March 1995.
7. ISO. *Information Technology – Programming Languages – Ada*. ISO, February 1995. ISO/IEC/ANSI 8652:1995.
8. ISO. *Information Technology – Programming Languages – Ada Semantic Interface Specification (ASIS)*. ISO, 1998.
9. Yvon Kermarrec, Laurent Nana, and Laurent Pautet. GNATDIST: a configuration language for distributed Ada 95 applications. In *Proceedings of Tri-Ada'96*, Philadelphia, Pennsylvania, USA, 1996.
10. Yvon Kermarrec, Laurent Pautet, and Samuel Tardieu. GARLIC: Generic Ada Reusable Library for Interpartition Communication. In *Proceedings Tri-Ada'95*, Anaheim, California, USA, 1995. ACM.
11. Kai Li and Paul Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–359, November 1989.
12. Sape Mullender. *Distributed Systems*. Addison-Wesley, second edition, 1993.
13. OMG, editor. *The Common Object Request Broker: Architecture and Specification, revision 2.2*. OMG, February 1998. OMG Technical Document formal/98-07-01.
14. Laurent Pautet, Thomas Quinot, and Samuel Tardieu. CORBA & DSA: Divorce or Marriage? In *Proceedings of AdaEurope'99*, Santander, Spain, June 1999.
15. Laurent Pautet and Thomas Wolf. Transparent filtering of streams in GLADE. In *Proceedings of Tri-Ada'97*, Saint-Louis, Missouri, USA, 1997.
16. John A. Stankovic and Krithi Ramamritham. *Advances in Real-Time Systems*. IEEE Computer Society Press, 1993.