Object-Oriented Real-Time Systems Using a Hybrid Distributed Model of Ada 95's Built-in DSA Capability (Distributed Systems Annex-E) and CORBA

Scott Arthur Moody
Boeing Defense & Space Group
Reusable Computing Systems Research
P.O. Box 3999, M/S 87-37, Seattle, WA 98124
Scott.A.Moody@boeing.com

Abstract

This paper reports on the issues in design and development of Object Oriented Real-Time Distributed Systems using Ada 95. The paper is broken into the following parts: First, one of the general distributed real-time problems is introduced as it fits the domain within Boeing. Next, prototype solutions to the distributed capabilities are introduced including the new technologies present in Ada-95 through DSA (Distributed Systems Annex-E)[2] and extended with CORBA (Common Object Request Broker Architecture) capabilities[1]. Many issues are addressed with adoption and use of these technologies especially applicable as they relate to custom fine-tuned solutions used today to solve real-time constraints. As the various distributed solutions provide different strengths, a combination of the capabilities provides an attractive option. A hybrid distributed capability composed of DSA and CORBA has been developed and is discussed as it relates to the seamless introduction with the Ada language. Finally a set of research issues are raised.

Keywords: Real-Time, Fault-Tolerance, DSA, CORBA

1. Introduction

Ada 95[3] is in some ways a major revision of the Ada 83 language. The language designers learned from over ten years of Ada use what features were working and those that could be enhanced. This included extending the core language with better object-oriented features, enhanced tasking and distributed capabilities, removing limitations that made it hard to use Ada in open systems applications, and wrapping everything with a more advanced type safe generics model. Of prime importance in this paper are how the real-time aspects of Ada[6] were enhanced through the new Distributed Systems Annex-E making development of distributed programs a later design decision, or at least treating this as a special case of a multi-tasking design. One advantage of these approaches deals with finding system bottlenecks through execution, where different distributed partition makeups can be easily tried.

This work is part of the Boeing Defense and Space Group's (D&SG) Hardware and Software Reuse Initiative. A major focus deals with designing architectures to support the goals of re-configurable systems exhibiting plug-compatible component capabilities. The entire project also includes a reuse library supporting decision model based identification and extraction and instantiation of reusable components.

Component Based Solutions

A major goal of this project is to minimize the constraints of distribution on real-time systems. Use of the real-time abstractions in the Ada language, combined with distributed capabilities, like DSA and CORBA, are good starts. Other goals include easing the maintenance and development of distributed programs, support of rapid re-configuration of a system, and support for configuration of reuse libraries.

The advantages of component based solutions are expanded through the *hybrid* approach described in this paper. Through support of additional distributed interfaces, the Ada developer can mix and match capabilities. If this could be achieved without any syntactic changes (like DSA) then the maintenance and portability of systems would be enhanced. As an example of the resulting flexibility possible, Figure 1. shows graphically a set of servers (left) and clients (right) available for connection in a prototype system. Instead of a single distribution solution, the hybrid approach supports connection of clients using the various distributed protocols (middle) with the servers, in a redundant manner. For example, the top left server can support clients requiring CORBA at the same time with Ada clients using DSA or local clients making local procedure calls. To keep up with the times, even a Java client (using Ada to Java tools) can view the same information. Care must be taken so that a server doesn't become overloaded by too many simultaneous clients requesting information (possibly supported in the future through *quality-of-service* requirements).

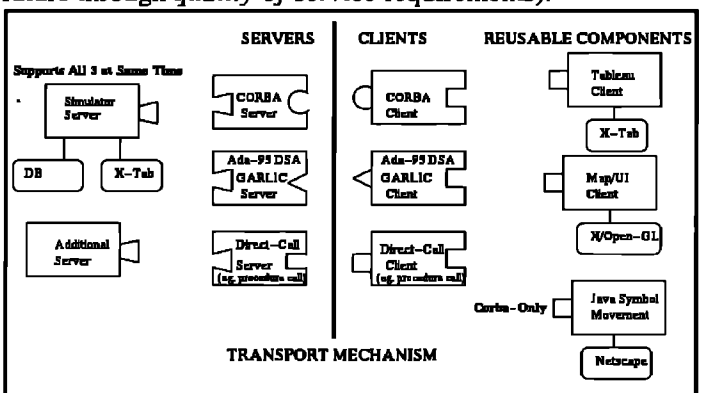

Figure 1: Configuration Variations including Transport

Other issues particularly relevant with the nature of the real-time problem domain deals with hard and flexible scheduling demands, and safe and efficient access to shared and possible distributed information resources. Ada 95 language improvements for real-time development processing are being examined and utilized where appropriate, and are important in a reuse effort because the resulting set of components may vary, effecting the scheduling demands. Numerous solutions are under development to support an Ada 95 (software) *plug-n-play* architecture using various scheduling algorithms. An advantage with Ada 95's Object capability is that a core executive capability can be developed which provides an architecture component that is then never modified. Instead concrete plugs are developed and added over time to conform to abstract definitions[7].

2. Real-Time Problem Context

This research work deals with examining an existing full-scale Boeing project as they develop a COTS-based Open System Architecture (OSA) for many different Command and Control System applications. As their system migrates from Ada 83 to Ada 95, this effort has examined the viability of commercial distributed processing capabilities, such as CORBA[1] and Ada 95's DSA Distributed Annex-E[2].

The runtime environment consists of both Sun-Solaris and DEC-Alpha Digital-Unix platforms of varying performance characteristics. Ada 95 is supported by the GNAT[4] production compiler and runtime system. DSA is implemented in the GNAT tool called GARLIC (GNU Ada Reuse Library for Interpartition Communication)[2]. (GLADE is the actual tool name). The CORBA/Ada-95 product is provided by Iona and OIS (Objective Interface Systems). A goal of the project is to have totally portable Ada 95 systems running on most Unix based platforms and communicating with local and remote networks.

2.1 Real-Time Object Solution Prototypes

This Boeing project is a major undertaking combining numerous designs that touch on all aspects of advanced real-time systems. Some of these areas are being prototyped with maximum use of the new object oriented features of Ada 95 integrated with Ada Tasking and extended with a distributed computing infrastructure. As this paper describes a Hybrid Distributed Architecture, this section will first describe one of many problems common to the class of so called Information Battle Management Systems (such as AWACS, Air-Traffic-Control, etc.). In general, for these multi-user systems, there is a *Main* Computer that collects and correlates volumes of information from many input sensors. Portions of this information is then broadcasted to a set of User *Display* consoles for viewing and interpretation, resulting in actions performed on the information back on the *main* computer (or elsewhere in the information space).

Figure 2 shows a simple interpretation of this problem. Additional requirements state that there may be redundant warm *backup main* computers that, if running, will receive snapshots of existing backup information. There can also be a varying number of *display* computers. One aspect of fault-tolerance that is added requires the warm *backup* to take over main processing duties if the hot *main* goes down, while *display* computers would then communicate with the new *main* processor. (This figure also shows a *controller* sub-system which is not part of the requirement, but is used in the object solution.).

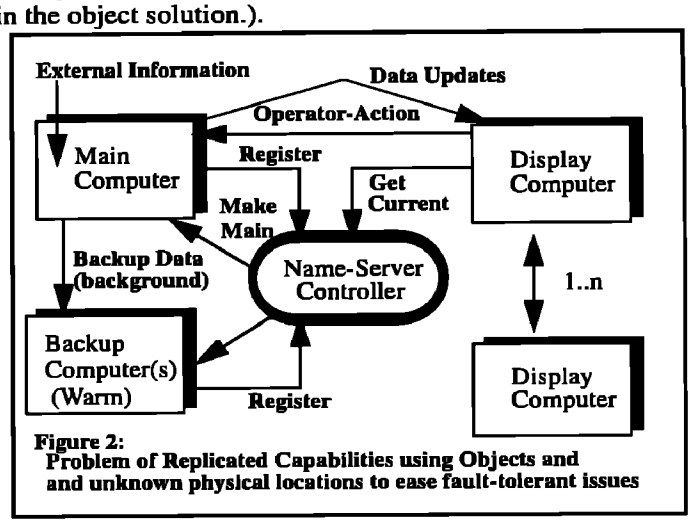

Figure 2:
Problem of Replicated Capabilities using Objects and and unknown physical locations to ease fault-tolerant issues

2.2 Issues in a Distributed Design Utilizing Objects

Through use of a technique called RACW (Remote Access to Class Wide Types), an example Object Oriented solution can ease faults through replication and use of a name server to act as a repository of known object class services. In this example, if a *Main_Computer* dies or looses contact (caught by exceptions in this example), then the *Name_server* informs one of the *backup* computers that they should take over. Future requests by the *display* computer to communicate with the *Main* are then directed to this new *main*. This is done through remote object references allowing capabilities to be in various locations. If another instance of the *main* computer starts it will take on a backup role and accept backup-data commands. Numerous backup computers can also be running for additional redundancy.

New techniques using Object designs, coupled with remote access pointers provides useful object abstractions providing class wide services that can reside in various network locations. An important feature of this design is that implementation of concrete services don't necessarily know they may be remotely accessed, and the implementation is postponed until later when code is automatically generated to support the distribution.

Additional designs (not described in this paper) support the Object Oriented ability for these warm backups to assume the main role while informing all the various sub-systems to run with the latest backup information. This is especially useful when a system configuration may have been configured from domain based reusable components that fit the correct architectural framework. It is easy in a paper like this to describe the solution to a portion of the problem but the other aspects must also be integrated showing why an overall object oriented design is imperative.

This example provides a good vehicle to discuss many framework issues dealing with fault-tolerance[9][10]. Issues such as replicating the single name server, providing heart-beats or handling connection exceptions could be abstracted for more error proof application code.

2.3 Further Requirements on Real-Time Distributed Systems

Multiple Sockets are sometimes used to increase priority based throughput. The idea is that low priority data is not pre-empting higher priority data. Priority Inversion is another issue that is difficult to deal with unless there are priorities for queued data.

An DSA implementation can pass task priority information onto a remote operation making the receiving task take on the same priority. This is already available to traditional Ada users but furthering the implementation to remote connections only serves to strengthen the correctness of systems, whether or not they are distributed. This concept may only be appropriate for application specific systems since priorities may not always make sense once distributed.

Other hard real-time constraints deal with both the latency of the underlying network, and the efficiency of the response once the network hands the data to the users software. Systems must also support various models for access to possibly large database in either a read-only or read-write mode. The later can make a parallel distributed system serial if others are also waiting on the same data.

3. Distributed Solutions Available

Three main class of solution techniques are available to Ada programmers: (1) Hand Built Socket-Based Calls (2) Ada Distributed Systems Annex-E (DSA) (3) CORBA (Common Object Request Broker Architecture) These are described in the following sections.

This paper does not describe other distributed capabilities such as Java RMI, DCE RPC, PVM or ILU, but they could probably also be added to the hybrid architecture as needed.

Another solution that isn't mentioned is the typical embedded procedure/function call of the native programming language. Both DSA and CORBA support non-distributed solution that can be broken apart later without modification. CORBA tools implement this through an always present indirection scheme, while DSA's interaction is totally removed and implemented instead with straight Ada compiler calls. (This can be changed by the user through the *All_Calls_Remote* pragma requiring a similar indirection.)

3.1 Custom Socket Based Solutions

Many internal real-time systems built today have a custom communication infrastructure. This stems from requirements for tight control over performance and firm requirement for the message queues. Some designs support "Priority Message Channels" where different sockets are allocated different priorities allowing higher demanded information not be blocked by slower (possibly large) data using the only socket.

In general, these custom solutions can support firm runtime constraints. Improved abstractions, such as class libraries, can make utilization much easier. The main drawback is the lack of standards and harder inter-language support.

3.2 Ada-95 Distributed Systems Annex-E (DSA)

Ada 95 Distributed Annex-E provides a powerful capability for distributing Ada programs to different processors. Moreover, this can be accomplished without changing any user code. Integration of external distributed mechanisms usually require various levels of code modification making portability to other capabilities harder. Unfortunately this Ada 95 capability only supports inter-Ada communication and is not directly compatible with CORBA's communication model. A short summary of Ada 95 Distributed Systems (Annex E) features includes:

- Development language treated like large Ada program, thus testing can occur with complete program before distribution occurs,
- Faster for same process communication (if not distributed then no overhead).
- (Distributed) Shared Memory described in Ada via *Shared_Passive* pragma.
- Consistent version of interfaces are guaranteed (no special syntax is required for distribution or communication).
- Communication within specified time supported via native Ada linguistic features (i.e. for fault-tolerance support)

With the new distributed processing capabilities being developed, distributed processing can be left to a later design decision, while the syntax for programming is shielded as much as possible (if the original system was designed to be multi-task safe). Initial performance experiences also favor GARLIC when communicating arrays of large data records in a homogeneous environment.

3.3 CORBA (Common Object Request Broker Architecture) and Ada 95 Bindings

Without providing in-depth discussion, CORBA can be thought of as another tool to ease development of distributed solutions. The main strength of CORBA is the requirement that an arbitrary implementation language does not dictate distributed interfaces. Instead, a language neutral specification is developed and published called an Interface Definition Language (IDL). Types, objects and object operations/methods are specified and then implemented through one of the numerous CORBA IDL language bindings. Eventually, distributed programs exchange services without knowing the actual implementation language. The CORBA runtime systems must ensure that information is communicated and represented correctly for all parties.

Because of this language neutral benefit, a majority of vendors are making their traditional programmatic interfaces (API's) available as CORBA services. Additionally, an extensive set of capabilities are under design and development for other common services. These range from data management to event services (publish/subscribe). Eventually these services will be purchased and reused in a portable manner.

Ada developed system are prime candidates for involvement in the CORBA movement. A CORBA/Ada-95 Mapping has been designed and standardized, and at least one product is available through Iona and OIS. But as mentioned throughout this paper, the Ada system designer has other language built-in tools available before integrating with CORBA is required. The next section describes a Hybrid capability developed to ease this integration while keeping the benefits of the DSA supported Ada solution.

4. Hybrid Distributed Computing Capability

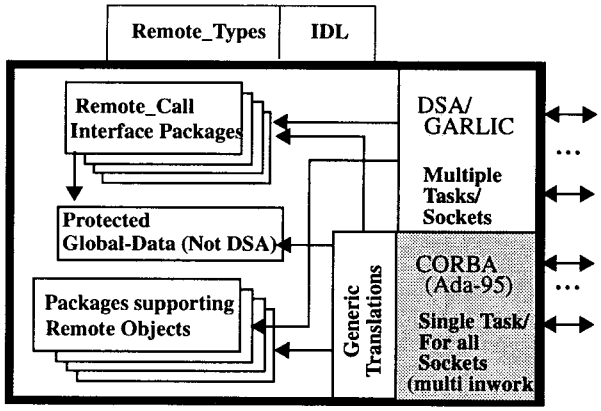

Figure 3: Hybrid DSA-CORBA Architecture

Figure 3 shows some aspects of the Hybrid Architecture. It is composed of a generic set of Ada Packages classified in three main areas: (1) Packages classified through pragmas *Remote_Call_Interface* and *Remote_Types* (2) Packages supporting possible remote class wide services and (3) Normal non-distributed packages providing state information such as global data structures. Safe access to shared application data can also use *protected* types.

The right side of the figure shows the two distributed capabilities of DSA and CORBA. Both capabilities currently allocate a socket connection to the *distinct* clients. This means if a client is multi-threaded it still gets only one connection to the service. (A customized DSA implementation could conceivably change this implementation for special cases.) DSA allocates a task for the different connections and can read the information in parallel. Currently the CORBA portion is allocated to a single Ada task and will support serial connections to numerous external clients. (Future versions of the CORBA toolset will provide multiple tasks for the different connections.)

Two other aspects of the figure need explanation. DSA can only provide direct external connections to the first two package classifications identified above. Global data can not be directly exported. In contrast, the CORBA solution relies on various manual translations to map requests to the correct implementations. An implementation can then indirectly access any visible package, including the global data structures. In most cases access should probably go through the DSA exported services, but some efficient data manipulations require access to the direct information, especially for local access to an array of large data structures. Of course, if the internal CORBA mapping is through DSA services, these services may be re-allocated to other servers without modifying the CORBA code (but incurring a possibly high communication overhead). Also note that other information such as Ada *Remote_Types* definitions and IDL files are also available to clients but typically they are manually used in client executable development.

An important aspect of this version of the Hybrid Architecture is that a correct Ada 95 distributed program is exposing some of its services through a CORBA facility. The general system engineering concept supports definition and construction of a well known and contained system. With Ada 95, a full system can be developed and various partitions built, tested, and re-configured, before external CORBA services are exposed. At that time the Hybrid Architecture is integrated.

If other lifecycle requirements require interfaces to external CORBA services then traditional use of the CORBA/Ada facilities are used. It doesn't make sense to provide a DSA interface to those external services unless internal prototyping can how show they might be used, or duplicate capabilities are internally developed before the external services are made available. At present very few automation tools were used to make the translations between DSA categorized packages and CORBA IDL. This is still a research issue.

Object Oriented Issues

Introduction of Object-Oriented concepts into traditional real- time distributed system development poses some interesting issues. In general, a resulting system should exhibit more flexibility but there are still concerns about performance. Object approaches take a step out in abstraction while implementations usually rely on the indirection of name_server concepts such as a CORBA ORB or Ada 95's RNS (Remote Name Server) implemented in DSA.

As an example, a straight forward use of distributed processing would involve defining interfaces that can potentially be distributed. These interfaces would usually expose a package as a set of services. A user of these services must then consciously incorporate the instances of the services. Any abstraction to deal with objects was incorporated into state information passed as parameters. This capability is still important and directly represented using *Remote_Call_Interface* Categorized Library Units.

Object designs for distributed problems, like the "main-display" from Figure 2, are important to understand as they are fundamentally different than traditional point connection solutions. First, procedural operations are performed on objects instead of statically known code sections (identified through pragma *Remote_Call_Interface*). This means that physical code can reside in dynamically determined locations by a remote form of class wide dispatching. An object implementation may even migrate to different physical locations during a single execution.

Inheritance and extension are techniques that are also supported through object designs. For example, a backup *main* computer could be implemented as an extension to *main* providing implementation for a private backup operations. This capability would not be available (or known) by the *display* computers; for all they know they are always communicating to an instance of the main computer class. This means that multiple instances of the same partition can run and take on some of the fault-tolerant requirements - without the client knowing anything about them except that they implement object capabilities.

Locking of import shared information is an issue that should be treated like any other multi-use application. Exposing an application to possibly concurrent distributed users definitely takes careful design. Ada 95's new *protected type* is a valuable and efficient tool available for this type of problem (see Figure 3).

These newer Object-Oriented solutions are still under a watchful eye in terms of performance, and the multi-tasking issues are still a concern when data is usually sequentially manipulated (for performance and determinism). There are also real-time concerns dealing with volumes of information being transferred, and how much fault-tolerance can be built into the language or framework capabilities, and how much must be manually developed.

Through use of other Ada real-time support, such as task threads and various timing tools (e.g. *delay until*, ATC, etc.), a self contained system (and possibly single CPU process) can be developed. This can be an accurate (or simulated) reflection of the end system's constraints, such as network speed and process load. Later breakup of the system into distributed partitions is then made easier (taking into account new network information). At that time, appropriate CORBA interfaces can be developed and integrated into the Hybrid Architecture.

5. Future Research Issues

The technologies and tools present in the paper are relatively new. This includes CORBA, DSA, Ada 95, and the Hybrid approach. Real-Time systems impose additional stringent requirements on these possibly generic capabilities. This section raises a few issues that need to be addressed with the end goal of portable and vendor neutral solutions.

5.1 Shared Data and Broadcast Issues

Processing time is at a premium in real-time systems. Replication and distribution are two techniques used to ease some of the processing requirements and add fault-tolerant aspects. Information must still be exchanged between these systems and the more time spent serving this exchange, the less time for other processing. A solution technique supported with physical networks is to broadcast or multi-cast information to everyone on the network or to a specific subset (one way for sharing at least read-only data). Also, Ada's protected object[6] provides rich semantic and linguistic access to shared memory. Will an API level interface, such as CORBA, ensure the same runtime protection and still help large scale software development?

At present there is no built-in broadcast capability in DSA or CORBA. There are various external tools providing this capability and the CORBA community is trying to define a specification. Seamless integration of this capability into the Ada language and DSA are still under investigation. The end goal would be to provide this capability in such a way that the syntax of the Ada code isn't compromised. This might be achievable through use of DSA configuration files or other runtime information. The must be accomplish without changing the Ada language semantics. Issues with this include what level of support is needed to ensure reliable delivery methods.

Some initial thoughts include extending the pragma Asynchronous with some kind of TCP/IP network group broadcast address, or adding information to the PCS configuration language, or maybe configuring the Ada tasking system to support task queues (through pragmas) where all those waiting on an event would be notified at the same time (versus one at a time).

Another related issue deals with the *Shared_Passive* capability. Can this be implemented with broadcast? As this capability hasn't been available, are there ramifications for designers in it's eventual use? In the end, will the flexibility of the resulting system be hindered such that broadcasted information could not be transformed back into a single address system without physically modifying user Ada code? Will it also be easy to change the underlying network easily (such as ATM, FDDI, Ethernet)?

5.2 Distributed Scheduling

Now that Ada 95 task priorities can be dynamically modified, various solution techniques are now available to system designers. For example, the DSA system uses this to help avoid Distributed Priority Inversion. New Ada-95 task features combined with DSA also provide fine control on failure semantics (e.g. The subtle difference between ATC and *delay until*, such as whether an abort occurs on call or completion).

As hybrid architectures are developed, what scheduling concerns should there be that are different than the same non-distributed solution? Ada developers enjoy various scheduling capabilities not available to other languages. What is the best way to merge this capability with the non-Ada developed distributed components? In general, what scheduling issues and policies should be exploited and controlled with the entire distributed network? Distributed scheduling is still a research issue and at present still relies on very application specific solutions.

5.3 DSA Implementation Issues

There are numerous issues that could be addressed with any DSA implementation. Performance and unique tailoring, such as priority message queues, should be examined. Solutions to many of the raised issues could involve new language constructs in the vendor specific DSA configuration file, including broadcast specification. These must be weighted against portability to other DSA implementations and the transport of the semantic information, that should belong in the Ada language, to other languages such as IDL or *makefiles*.

It is conceivable that DSA could support different network transport layers such that communication actually involves CORBA. This was not done for many reasons, and it would not be effective unless the *on-the-wire* information matched a CORBA specific definition.

It would also seem that the idea for configuration files to be first-class (non Ada) objects[5] could be supported without violating any rules. Within this Reuse project, the plan is for configuration files to be automatically generated based on desired products. Some form of hierarchal configuration concept is internally under development for intra-net availability[8].

There are still some fundamental lifecycle differences and costs between CORBA and Ada's Annex-E that must play into an overall development scheme. For example, modification of generated CORBA templates (the IDL specs) requires manual migration into possibly already existing implementation code. The GARLIC system does away with this because the communication code is generated every time changes are made. This is possible since the distribution specification is Ada and not in a neutral language like IDL.

5.4 Seamless Hybrid Integration with CORBA

Should automatic tools be developed to ease the creation of CORBA IDL specifications from already existing Ada distributed capabilities? The reverse already occurs with existing products - but integration with an application can still be time consuming. It would seem that this tool should map a CORBA model into an appropriate DSA supported model by using the various package categorizations. An Ada-to-IDL mapping should be designed to either automatically or manually support Ada designers as they start to publish CORBA IDL specifications.

Since there are different execution issues with the two models, can a correctly running Ada and DSA program be plug replaced with CORBA modules? As a cursory look at the differences, how would priority inversion be solved with CORBA when different task runtime models are in effect?

6. Conclusion

The Ada Programming Languages, both Ada 83 and Ada 95, were designed with a major focus on Real-Time computing. These languages provide high level *linguistic* support for aspects such as multiple threads, protected data, task synchronization, timing issues and scheduling policies. The Ada Compiler then ensures these high level features are implemented correctly on each underlying operating system.

The next challenge for the international computing field is incorporation of a necessary set of Real-Time concepts into the various Distributed Computing capabilities. This becomes very important as distributed objects are embedded into TV's or used to control telecommunication switches. When describing this challenge to the Ada community, one can say that the goal is to have the same Ada real-time linguistic support that is then extended to distributed computing, while ensuring predictable and deterministic results. Without this Ada context, one would have to describe the problem in a mixture of operating system concepts and scheduling techniques.

At least two international efforts are currently looking at many of the issues with real-time distributed computing. For example, the OMG is working actively on requirements and vendor participation in defining and developing Real-Time CORBA. Also, IEEE P1003.21 (POSIX) is developing a real-time distributed systems communication capability. Ada's Real-time language could have an important role in these initial requirements and designs, with the linguistic support of the underlying implementations, and possibly for the end-user language where the Ada runtime is built from the CORBA or POSIX lower level distributed primitives.

As the development of component based architectures prevail throughout industry, newer technologies such as CORBA are gaining importance. The Ada community should examine how much is gained with CORBA versus the amount of true Ada that is lost (such as weaker typing, no generics, no abstract types, no overloading, no shared memory model, etc.) and determine the best mix. After all the issues are addressed, a Hybrid Architecture, as described here, could support the long term goals of most system development efforts.

Acknowledgments

This research could not have been performed without Boeing commitment and recognition that Reuse is vital in the ever changing business climate and that Ada 95 has been recognized as one of Boeing's vital tools to support real-time system development and reuse. The OSA project, and the research extensions, should also be acknowledged as they let the reuse work absorb enough of the key personal to gain the knowledge for migrating key functionality while learning how to best use these new technologies.

References

[1] DEC, HP, et al. "The Common Object Request Broker: Architecture and Specification". Technical Report OMG 91-12-1, Object Management Group and X Open, December 1991.

[2] Y. Kermarrec, L. Pautet, S. Tardieu, "GARLIC: Generic Ada Reusable Library for Interpartition Communication", *Proc. TRI-Ada'95*, ACM Press

[3] Ada 95 Reference Manual, ANSI/ISO/IEC-8652:1995, January 1995.

[4] E. Schonberg et al. "GNAT: The GNU-NYU Ada translator, a compiler for everyone". *Proc. TRI-Ada'94*, Nov 94. ACM Press

[5] A. Burns, A. Wellings, *Concurrency in Ada,* Cambridge University Press, 1995

[6] Burns, Wellings, *Real-Time Systems and Programming Languages*, Addison-Wesley, 1997

[7] S. Moody, "Migrating Well Engineered Ada 83 Systems to Newer Architecture and Reuse Based Ada 95 Systems", *Proc. TRI-Ada'96*, Dec 96. ACM Press

[8] S. Moody, "STARS Process Engine", *Proc. TRI-Ada'94*, Nov 94

[9] E. Shokri, K. Tso, "Ada 95 Object-Oriented and Real-Time Support for Development of Software Fault Tolerance Reusable Components., *Proceedings of 2nd Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, Feb 1996, IEEE.

[10] R. Guerraoui, A. Schiper, "Fault-Tolerance by Replication in Distributed Systems" *Proceedings on Reliable Software Technologies - Ada-Europe'96*, Lecture Notes in Computer Science v1088, Springer-Verlag