Ada in Distributed Systems: An Overview

Marcin Paprzycki & Janusz Zalewski *
Dept. of Science & Mathematics
University of Texas–PermianBasin
Odessa, TX 79762-0001
(915)552-2258
paprzycki_m@utpb.edu, zalewski_j@utpb.edu

Abstract. This paper reviews the available literature on the use of Ada in distributed systems. The following issues are discussed in more detail: units of distribution, program partitioning, building configurations, interprogram communication, type checking, and Ada '95 partitions model. Several systems and a few applications are also covered briefly.
Keywords: Ada, distributed systems, program partitioning

1 Introduction

Issues of distributing Ada programs over a network of multiple processors have been the subject of intensive studies by practitioners and researchers since the Ada language definition was established, and have resulted in several publications. Work started from the very beginning, as early as in the year Ada became an official standard [25, 26, 60]. Following the preliminary study [90], the major further steps were the continuation of a European project [6] and the symposium at Southampton [18]. Finally, the Distributed Systems Annex to Ada 95 has been produced and is now an international standard [1].

The driving force behind using Ada in distributed systems is the ability to create Ada programs that can be spread and executed across multiple processors, in one or more sites. Since Ada was originally not well defined for distributed systems, this requirement became an issue.

One of the primary problems in distributed programming, in general, and programming in Ada, in particular, is whether an application should be designed as a single program or as multiple programs. Both approaches have been advocated in the Ada world, single program by APPL [25], Aspect [52, 54], and DARTS [23, 104], and multiple programs by Diadem [6] and Dark [11, 12, 92, 93]. This is in contrast to an approach based on operating system calls, where the language layer and compiler do not know about interactions amongst participating software components.

Among the questions asked to investigate and establish the technology of distributed programming in Ada were the following [99]:

- In terms of the language, what program units may be distributed?

*Current address: Dept. of Computer Science, Embry-Riddle Aeronautical University, Daytona Beach, FL 32114, zalewski@db.erau.edu

- How is the distribution of units specified (partitioning problem)?
- How are the distributed units conceptually assigned to physical units (configuration, mapping)?
- How are the distributed units actually loaded onto target nodes?
- When is the distribution decided upon, at the compilation, linking, loading time?
- How do the distributed units communicate, via what mechanism?
- How is type consistency preserved across the network?
- Are distributed units dynamically replaceable? If so, how?

2 Major Issues

In the following sections, issues related to most of the above mentioned questions will be reviewed as investigated in the literature. In particular, problems of selecting units of distribution, partitioning, configuration and mapping, communication, and type consistency are outlined.

2.1 Units of Distribution

Early works on distributing Ada programs suggest adopting packages or tasks, or both, as distribution units. For example, Arévalo and Alvarez [3] discuss packages and tasks as distribution units. Among advantages of using packages in this role, they list:

- reuse of packages as library units
- high degree of parallelism due to the concept of package interface
- the concept of a package as a logical unit of decompisition is consistent with the distribution unit concept.

Advantages of tasks as distribution units are the following:

- tasks are active as opposed to passive packages
- tasks already have a communication mechanism (rendezvous)
- task typing scheme permits dynamic replication.

After additional considerations, they chose tasks as the basic unit of distribution in their model.

Volz and coworkers [71, 95] considered packages as a unit of distribution, in addition to tasks and data items. Carlson Goethe et al. [23, 104, 105] use tasks, task objects, packages, variables, procedures and functions as distribution units. Nishida et al. [84] also selected tasks as a distribution unit and mapped them onto C programs. McFarland et al. [78] also use tasks as distribution units. Even pin newer approaches, a task is a primary distribution unit [16].

It has been noticed rather early that a distribution unit which combines properties of packages and tasks would be needed [5]. Hutcheon and Wellings [52], following the work of Tedd et al. [90], advocated writing distributed Ada programs as collections of virtual nodes. They chose a package as a basis for virtual node, since packages are library units that encapsulate information and provide a well defined interface to it. Distributed programs are composed of virtual nodes linked together.

In [87], separate programs, virtual nodes, and tasks are considered as units of distribution. Tasks are suggested as the most natural unit of distribution, since they are the primary Ada construct for concurrent programming.

From virtual nodes evolved a concept of partition as a unit of program distribution. Technically, a partition comprises library units in the same logical address space.

Gargaro et al. [44, 41] suggest the use of the concept of a virtual node and distribute an Ada program among partitions, making **partition** a keyword. The requirements on a virtual node are the following:

- to encapsulate state, message passing communication (via procedure or entry calls) and no direct data sharing
- at least one thread of control per node to provide for concurrent execution
- transparency regarding intranode and internode communication
- a node should be a type in a language
- indirect node naming should be used to permit dynamic reconfiguration.

These and other issues of choosing a distribution unit in Ada are reviewed more completely by Volz [94].

2.2 Distribution Activities

Historically, several approaches to distribution of Ada programs have been proposed [26]. The primary issue is whether the program should be built as a single entity to be divided into smaller pieces distributed across the network and communicating via built-in language constructs or should be composed of a number of loosely coupled units cooperating via operating system calls.

Most authors [52] clearly indicate that of the two models of distributing Ada programs they choose to distribute single programs spread in pieces over the network, because this approach preserves Ada's strong typing ability, in contrast to a collection of communicating programs which loses this ability. Independently compiled programs can communicate without problems but do not allow for type and consistency checking in the underlying language.

Similarly, the Diadem project [5] chose the language-oriented approach, where the software is designed as a single program but is subsequently split into separate components for distribution over the network. This is despite the advantages of the operating-system based approach, which favors efficiency at the cost of portability and generality.

Another important issue is when, during program development, the partitioning of an Ada program into smaller distributable units should be done, either before the actual code is written, which is known as *prepartitioning*, or afterwords, which is called *postpartitioning*.

Postpartitioning means that the program is developed without taking into account any possible program division and task-to-processor mapping, and the whole job of partitioning and allocation is done by system software. This approach has a tremendous advantage that releases a programmer from even thinking about distribution and the whole job is done by an automatic tool. Such a tool or tools, doing program transformation into smaller distributable units and allocating these units to physical processors, can be roughly compared to a loader performing address resolution and allocating a program in a uniprocessor's single address space (a difficulty in a distributed system is multiple address spaces). Another advantage of postpartitioning, as a direct consequence of the former one is a greatly simplified portability. The most obvious disadvantage of postpartitioning is the significantly increased complexity of a run-time system.

Prepartitioning is an alternative approach that relies on including program distribution decisions into the design and implementation process. Hutcheon and Wellings [52] argue that adopting a prepartitioning approach (splitting a program before it is written) is more natural, because potential program distribution

should be clearly visible to the programmer as it may affect the interfaces to be provided and makes communication overheads explicit. In another article [54], the same authors support the view that prepartitioning is preferred to postpartitioning, because the programmer needs to be aware of distribution. On the other hand, as already mentioned, postpartitioning would be difficult to implement for a language as sophisticated as Ada and requires a costly run-time system for arbitrary distribution.

It was discovered rather early [5, 52] that partitioning, no matter how done, should be performed as a logical step in isolation from any decisions regarding hardware configurations. This is known as separating the partitioning step from the logical-to-physical mapping, referred to as configuration. In other words, the activity of designing the virtual nodes or partitions should be done without referring to the exact configuration of the underlying computer network. Bishop [17] goes even further in separating development activities, by saying that an additional adaptation phase is necessary to allow for modifications of a partitioned program to ensure that partitioned units can still communicate as required on a given hardware configuration.

Various authors use the idea of partitioning and present their own views on this phase of developing distributed Ada programs. Nielsen [82] borrows from the concept of virtual nodes and develops a complete method for designing distributed Ada programs. A pre-translator developed by Volz et al. [95] was intended to translate a single Ada program into a set of communicating Ada units, one for each node of the network. Dobbing [31, 32] discusses AdaMap, a partitioning tool developed by Alsys. It is based on an I/O package rather than remote procedure calls. Luckham et al. developed a language for specifying a distributed Ada program, based on task sequencing [73], but the article actually says nothing about the distribution. STRAda [16] also assumes transformations.

2.3 Configuration

After the partitioning activity has been completed, the next step is to describe the target configuration and allocate program components to individual nodes.

Atkinson [5] gives the major reason for splitting apart the partitioning phase and the configuration phase. While the language-oriented approach to distribution can be strictly enforced during partitioning, allowing the program to be defined in terms of virtual nodes to preserve strong typing and consistency checking capabilities of the Ada language, the configuration phase allows exploitaition of the operating-system approach to gain efficiency. Moreover, different physical system configurations can be constructed from the same software and even their dynamic changes are possible.

Gargaro et al. [42] introduce the concept of a node as a new type of library unit, whose purpose is to collect together instances of partitions for execution on a single physical resource in the target architecture. Only a single node is allowed on a single processor. A node concept has been later abandoned on the way to Ada 95.

Nielsen [82] described a similar approach to configuration of Ada programs, based on choosing a set of processors to support virtual nodes and mapping virtual nodes to the chosen hardware configuration. The crucial criteria of the hardware selection part of the latter phase are related to the following ten issues:

- device drivers
- special functions
- time-critical functions
- performance requirements
- processor homogeneity
- fault tolerance

- expandability
- interfacing capabilities
- processor/memory compatibility
- location and size constraints.

The actual mapping of virtual nodes to physical processors is done via pragmas and separate compilation techniques that follow those used by Jha et al. [62].

An alternative distribution scheme using a separate language has been investigated and adopted by Cornhill et al. [25, 27, 36, 37, 61]. Basically, they defined the partitioning language APPL, whose job is to provide a tool for the programmer to partition the program onto available hardware after it has been written.

Gargaro et al. [43] suggest that separate load modules for each node type be created. The program starts up from the distinguished node which elaborates its own partition and creates other nodes. Mapping of distribution units to physical network nodes has been also considered by Volz et al. [71, 95] using a pragma SITE. Other authors [3] do not consider the definition of configuration facilities or even configuration schemes.

2.4 Communication

In Ada 83 there was no facility defined for interprogram commmunication, in general, or intertask communication for tasks residing on different processors. That's the reason of major concern for choosing the distributed communication method. In general, the following facilities for the interprocessor communication amongst Ada programs have been investigated:

- remote rendezvous
- remote procedure calls
- message passing [28, 29]
- sockets [84].

The principal requirement and idea about distributed communication in Ada is *communication transparency* [5, 41] which means that remote (interprocessor) comunication should use the same protocols as local (intraprocessor) communication. This seems to be especially difficult to achieve when Ada programs run on a network of heterogeneous processors connected by arbitrary communication links.

The Diadem project chose a remote rendezvous for two reasons [5]:

- a rendezvous takes place between active tasks, while an RPC call is executed in the caller's thread
- entry calls enable using types of virtual nodes.

Consequently, the Diadem project [7] used remote rendezvous as a communication vehicle, although without excluding remote procedure calls.

Volz et al. [95] determined that conditional entry calls fail when the call is to a remote machine, because a response cannot be "immediate", as required by Ada 83. Timed entry calls over a network may also cause problems, for example, when the caller's waiting time expired and it withdraws from a call, but the callee has already responded although the response has not yet arrived due to a delay in communication.

Lundberg [74] presented a tasking protocol for distributed systems to significantly reduce global communication overheads. It relies on a system process, called the agent, and special structures called ghost tasks.

In particular, when a caller task A issues an entry call to a task B located on another processor, a three-parameter message is sent with the following parameters: calling task's name, called entry's name, caller's priority. Task A then waits until the rendezvous is completed. For timed entry calls, the duration of a timeout is measured on the accepting task's processor, not the caller.

Several authors preferred the use of remote rendezvous as the most natural means of extending Ada towards distribution, despite the difficulties with interpreting semantics of remote entry calls. For example, Arévalo and Alvarez [3] chose remote entry calls with a timer placed in the calling node. Nakama and Nakao [81] use rendezvous as an interprocess communication mechanism for distributed programs. Wengelin et al. [105] discuss the issue of a task calling a procedure on another node. The calling task is suspended during the call. Bayassi et al. [15] developed a generalized rendezvous mechanism to deal with applications distributed among several processors.

Hutcheon and Wellings [52] chose remote procedure calls (RPC), as a means of communication, for the following reasons:

- RPCs are common to many other languages as a mechanism for transferring control
- RPC allows easy communication between virtual nodes written in different languages
- techniques for reliable RPCs are better understood than those for remote rendezvous which is fairly new.

Gargaro [40] gives one important reason why remote rendezvous should not be supported by the distributed Ada: to synchronize control over a distributed system, although possible in principle, tremendously increases complexity and introduces semantic difficulties.

Intermediate or combined approaches are possible, such as in [77], where the preprocessor automatically changes the rendezvous between tasks in different virtual nodes into the appropriate remote procedure calls. Also, Nielsen et al. [82, 83] considered both remote rendezvous and remote procedure calls. They propose a layered approach to implement communication. One approach, with TCP/IP protocol on the lowest layer, incorporated into a message passing layer called directly from an application layer has been criticized as having too many disadvantages. Instead, a remote procedure call has been investigated with XDR protocols for communication between tasks on different processors. Tasks actually use a remote entry rendezvous mechanism, similar to that reported in [6], which is implemented using remote procedure calls and TCP/IP protocols.

Sometimes a message passing mechanism is assumed as a typical communication vehicle [16]. However, message passing usually suffers from tha lack of type safety in the message data and lack of unit consistency in shared program units. In the most positive practical example, Cross et al. [28] make their solution resemble file I/O in Ada to allow communication between Ada programs running on heterogeneous processors. A channel is made a logical communication medium. Channels are typed as are files in Ada. Before a message is transmitted via a channel, it is sent from a program to a port attached to the channel. Many ports can be connected to the same channel. A complete package named `Distributed_Communication` has been developed, consisting of two parts: the machine independent and machine dependent part. The machine independent part consists of a generic package `Channel_Manager` and several utilities. It hides the details of various physical communication media and their device drivers.

Gentleman et al. [45] provide a summary of discussions on using Ada rendezvous for multiprocessors and compare the issues raised to those involved with another language and another operating system.

2.5 Type Consistency

Preserving strong typing capabilities of Ada across the network is the major concern of program developers. Without this capability, Ada would be nothing else but another programming language like dozens of those

used to program distributed applications. To make this to happen is not easy, however, and a lot of efforts have been put into considerations of choosing the right approach and not compromising too much regarding efficiency.

Strong typing usually means static type checking, which is not necessarily the most appropriate for distributed programs. Gargaro et al. [41] suggest that for a network execution a collection of separate, executable binaries needs to be developed, but the elaboration of these executable modules may occur concurrently in separate network units, following the invocation of each executable.

McFarland et al. [78] opt for a distribution of single Ada programs, to support strong typing and interface consistency checking. The thread of program execution is tied to the place of static data items. Tasks and subprograms which access static data execute only in the single process where the data exist. Calls to these program units result in remote entry or subprogram calls.

A major concern in several papers [97, 98, 100] is that an additional language construct is necessary to allow types to be shared across the network. This is in direct contradiction with the concept of a package, since it includes state and respective objects need to be replicated. A stateless, replicable unit would be necessary. Gargaro et al. [42] proposed an intermediate concept of public library units, on the way to Ada 95. They were equivalents of invariant-state packages, identifiable by a **public** keyword. Interface to a public unit could contain: types (not access types, though), task types, constant, subprograms, packages, etc., but not shared variables.

3 Ada 95 Partitions Model

Soon after the standardization of Ada 83, it became clear that the language does not have sufficient power to enable programming distributed systems, while preserving its strengths targeting uniprocessor systems. In fact, Ada 83 was designed for single processor machines, despite its multitasking capabilities and claims in the Reference Manual stating otherwise. The multitude of diverse approaches to cure this problem finally led to a solution based on certain fundamental assumptions. Dobbins [33] suggested that the solution be based on three fundamental principles:

- There should be minimal disturbance to the existing Ada language definition
- The safety aspects of the language are not compromised
- The requirements of the implementors of distributed systems be met.

The actual model was evolving slowly and was being refined throughout the years of work [40, 41, 42]. The resulted standard, called Ada 95 [1] contains *Distributed Systems Annex*. First of all, Ada 95 does not provide any linguistic construct to represent a unit of distribution. Rather than that, the standard defines a unit of distribution to be a partition: a set of library units that may run in a separate address space. A set of partitions forms a program. Several partitions may reside in the same address space, but a single partition cannot be split into multiple address spaces. Active partitions have threads of control, passive partitions do not.

Furthermore, tasks were reluctantly but consequently abandoned as a unit of distribution due to their impossibility of becoming compilation units. Similarly, remote rendezvous did have less and less supporters due to an unspecified semantics of a timed entry call [102]. In particular, the new standard says that all calls to remote partitions must be made through a remote subprogram call, via a special remote call interface (RCI) package. Asynchronous remote procedure calls are also possible to allow both the caller and the server proceed independently, but those can only be made with **in** parameters and not for functions.

An important fact about remote subprogram calls is that they can be bound not only statically, prior to partition elaboration, but also dynamically during elaboration. Two ways of dynamic binding defined in

Ada can be used for remote subprogram calls:

- dereferencing a remote access-to-subprogram type object
- dispatching a class-wide type controlling operand of a primitive operation.

Interprogram communication has been certainly compromised in Ada 95 by an unwillingness to increase in size the run-time support software, a situation that has been experienced by projects which have implemented such features as remore rendezvous.

A vital need to control interfaces between distributed program units and an important issue of preserving types across the network have been also addressed in Ada '95. These are the primary features distinguishing a multi-partition Ada program from a collection of programs communicating over a network. Such requirement, however, causes a lot of problems to avoid inconsistencies. This is done in four ways, determined by categorization pragmas which distinguish library units:

- pure
- providing remote types
- remote call interface (RCI)
- shared passive.

The programming interface to handle all remote communication is called the Partition Communication Subsystem (PCS) and is defined in the standard package `System.RPC` [1]. First implementations of PCS, such as the one described in [67] (of a generic software component designed to accomodate various network protocols and communication systems) or in [70] (in a form of Ada bindings for Distributed Computing Environment DCE), are underway.

Ada 95 does not provide any means for configuration, that is, mapping partitions to physical targets, neither it deals with the physical allocation. Both these issues are considered implementation-dependent and therefore beyond the scope of the language standard.

4 Complete Systems and Applications

Below we review briefly commercial systems reported in the literature and a handful of important applications of distributed Ada.

4.1 Research Projects vs. Commercial Systems

In the last decade, several authors reported on their work progressing towards programming distributed applications in Ada. The most widely known are such project names as Aspect [52, 54], Dark [11, 12], DARTS [23, 104], Diadem [5, 7], Dragoon [8, 9], Honeywell's APPL [25, 37, 62, 63, 64], Lund [2], and Michigan [71, 95, 97, 98]. All those were strictly research projects, with substantial impact on the new standard but with little or no impact on commercial developments or applications.

On the other hand, a number of developments targeted at complete systems were reported in the literature. Fisher and Weatherly [38] presented probably the first commercial system for distributing Ada programs, DARP (Distributed Ada Run-Time Package).

A design of a network operating system is reported to support the use of Ada on a distributed military application [85]. The underlying architecture consists of 1750A microcomputer modules interconnected via a PI-Bus. Communication is based on an intermodule messaging scheme. Tartan 1750A target Ada compiler

was used with the run-time package ARTClient to support communication. Implemented almost entirely in Ada, it supports run-time reconfiguration of Ada programs. Reconfiguration is defined as changing the existing mapping of existing application software onto the available hardware. The messaging scheme supports logical task-to-task communication.

A complete toolset for distributing Ada programs is reported in [78]. It is based on a single-program concept. Several tools are applied to produce a distributed Ada program. First, the complete concurrent Ada program is automatically analyzed for potential distribution. Suggested distribution is determined from the use of tasks, resulting control flow and data access patterns. Then interfaces among distributed units are generated wherever necessary. Next mapping onto hardware is done by the distributed system builder.

The most recently announced [91] system, named DVM, provides a number of low-level and distributed services, such as:

- subprogram pointer service
- marshalling of Ada types
- time management
- automatic reconfiguration in response to system failures, and more,

to assist in developing distributed Ada programs.

Barnes [13], Bishop and Thomas [21], and Dobbing [31, 32] discuss the most complete environment for distributing Ada programs, based on the transputer architecture. While Barnes only outlines the Ada I/O system, Bishop and Thomas present the program development rules and examples, and Dobbins discusses the details of the partitions model.

4.2 Applications

Until recently, very few applications of distributed Ada have been reported. One such article [72] describes the use of Ada in CIWS (Close In Weapon System) run on heterogeneous machines. More recent studies present the need and opportunities for the use of Ada 95 in distributed simulation [51], and the actual application of distributed simulation in Ada [86].

The latter, claimed to be the first embedded distributed Ada 95 application, actually executes on several 68030 microprocessors housed in VMEbus crates, interconnected via a high-speed communications network. Two SpareStations 20 provide the necessary user interface to the simulation testbed. The original Ada implementation included remote procedure calls and remote rendezvous, which has been retained despite its divergence from Ada 95 Distributed Systems Annex. The system is entirely consistent with Ada 95, including the partitions concept, but has several extensions resulting from the early implementation when the full Ada 95 standard was not yet completed.

The actual system is a modification of the existing, commercially available, flight simulator designed for flight training in combat situations. The new distributed version takes advantage of distributed Ada technology to provide several enhancements of the uniprocessor version. The distributed version of the program consists of three active partitions and one passive partition. The first partition executes the code pertaining to the pilot's own aircraft. The second partition handles all the foe aircraft maneuvers, and the third partition does all the computations related to missiles once they are in flight. The passive partition contains the shared state. The execution is run in a cyclic fashion involving periodic calls, every 30 Hz, of the various subsystems.

5 Summary

This paper deals mostly with research issues related to distributing Ada programs across the network of multiple processors. Such problems as unit of distribution, program partitioning, configuration, interprogram communication, and type consistency have been covered. A brief overview of Ada 95 Distributed Systems Annex was given. Also, reviewed were several complete systems and a few applications.

In adition to the work covered, several other research studies has been conducted for distributed Ada, for example, in such areas as program specification and design [73], deadlocks in distributed Ada programs [88], etc. Initialization and termination of multiprocessor programs was also an important research issue studied in [22, 39]. Particularly important is the area of fault tolerance in distributed Ada programs [3, 24, 41, 43, 46, 69], but this was out of scope of this overview and needs a separate treatment. Another important area of research and practical applications, which requires more attention is the object-oriented approach to distribution of Ada programs [8, 48].

Acknowledgements

This work was supported in part by a grant from the Defense Information Systems Agency, under the Undergraduate Curriculum and Course Development Program: Software Engineering and the Use of Ada, Contract F29601-94-K-0046.

References

[1] Ada Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995, January 1995

[2] Ardö A., L. Lundberg, The Mumps Multiprocessor Ada Project, pp. 235-258, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[3] Arévalo S., A. Alvarez, Fault Tolerant Distributed Ada, Ada Letters, Vol. 8, No. 7, pp. 118-122, Fall 1988, & Vol. 9, No. 5, pp. 54-59, July/August 1989

[4] Armitage J., J. Chelini, Ada Software on Distributed Targets: A Survey of Approaches, Ada Letters, Vol. 4, No. 4, pp. 32-37, 1985

[5] Atkinson C., Programming Distributed Systems in Ada: The Diadem Approach, pp. 45-53, J. Zalewski, W. Ehrenberger (Eds.), Proc. IFIP/IFAC Conf. Hardware and Software for Real-Time Process Control, North-Holland, Amsterdm, 1989

[6] Atkinson C., T. Moreton, A. Natali, Ada for Distributed Systems, Cambridge University Press, Cambridge, UK, 1988

[7] Atkinson C., S.J. Goldsack, Communication between Ada Programs in Diadem, Ada Letters, Vol. 8, No. 7, pp. 86-96, Fall 1988

[8] Atkinson C., A. Di Maio, R. Bayan, Dragoon: An Object-Oriented Notation Supporting the Reuse and Distribution of Ada Software, Ada Letters, Vol. 10, No. 9, pp. 50-59, Fall 1990

[9] Atkinson C., A. Di Maio, From Diadem to Dragoon, pp. 105-136, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[10] Atkinson C., S.J. Goldsack, Ada for Distributed Systems: A Compiler Independent Approach, Proc. 7th IFAC Workshop on Distributed Computer Control Systems, K.-D. Müller (Ed.), Pergamon Press, Oxford, 1986

[11] Bamberger J. et al., Kernel Facilities Definition: Distributed Ada Real-Time Kernel Project, Technical Report CMU/SEI-88-TR-16, Software Engineering Institute, Pittsburgh, PA, July 1988

[12] Bamberger J., Distributed Ada Real-Time Kernel, Technical Report CMU/SEI-88-TR-17, Software Engineering Institute, Pittsburgh, PA, July 1988

[13] Barnes J.G.P., Ada on Transputer Arrays, pp. 1-9, Proc. 1st. Intern. Conf. Applications of Transputers, IOS Press, Amsterdam, 1990

[14] Baumgarten U., Distributed Systems and Ada – Current Projects and Approaches. Comparative Study's Results, pp. 260-278, D. Christodoulakis (Ed.), Ada: The Choice for '92, Springer-Verlag, Berlin, 1991

[15] Bayassi M., A Practical Use of the Ada Rendezvous Paradigm in Distributed Systems, pp. 312-324, J. van Katwijk (Ed.), Ada – Moving Towards 2000, Springer-Verlag, Berlin, 1992

[16] Bazalgette G. et al., STRAda - An Ada Transformation and Distributed System, pp. 287-299, J. van Katwijk (Ed.), Ada - Moving Towards 2000, Springer-Verlag, Berlin, 1992

[17] Bishop J., Three Steps to Distribution: Partitioning, Configuration and Adapting, Ada Letters, Vol. 8, No. 7, pp. 97-100, Fall 1988

[18] Bishop J. (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[19] Bishop J., M.J. Hasling, Distributed Ada: An Introduction, pp. 1-14, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[20] Bishop J., S.R. Adams, D.J. Pritchard, Distributing Concurrent Ada Programs by Source Translation, Software Practice and Experience, Vol. 17, No. 12, pp. 859-884, December 1987

[21] Bishop J.M., K.S. Thomas, Experience with Multi-transputer Ada, Concurrency Practice and Experience, Vol. 5, No. 2, pp. 133-151, April 1993

[22] Burns A., Efficient Initialisation Routines for Multiprocessor Systems Programmed in Ada, Ada Letters, Vol. 1, pp. 55-60, 1982

[23] Carlsson Göthe W., D. Wengelin, L. Asplund, The Distributed Ada Run-time System DARTS, Software Practice and Experience, Vol. 21, No. 11, pp. 1249-1263, November 1991

[24] Clematis A., V. Gianuzzi, Software Fault Tolerance in Concurrent Ada Programs, Microprocessing and Microprogramming, Vol. 32, pp. 365-372, August 1991

[25] Cornhill D., A Survivable Distributed Computing System for Embedded Application Programs Written in Ada, Ada Letters, Vol. 3, No. 6, pp. 79-87, 1983

[26] Cornhill D., Four Approaches to Partitioning Ada Programs for Execution on Distributed Targets, pp. 153-162, Proc. 1st Intern. IEEE Conf. Ada Applications and Environments, IEEE Computer Society Press, Los Alamitos, CA, 1984

[27] Cornhill D., Partitioning Ada Programs for Execution on Distributed Systems, pp. 364-370, Proc. Intern. Conf. on Data Engineering, IEEE Computer Society Press, Los Alamitos, CA, 1984

[28] Cross J.K., M.J. Kamrad. S.J. Fernandez, Communication among Distributed Ada Programs, pp. 627-632, Vol. 2, Proc. NAECON '91 IEEE 1991 National Aerospace and Electronics Conf., IEEE, New York, 1991

[29] Cross J.K., M.J. Kamrad. S.J. Fernandez, Distributed Communications, Ada Letters, Vol. 10, No. 9, pp. 85-93, Fall 1990

[30] Dewar R. et al., Distributed Ada on Shared Memory Multiprocessors, pp. 222-234, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[31] Dobbing B., Experiences with the Partitions Model, Ada Letters, Vol. 13, No. 2, pp. 65-77, March/April 1993

[32] Dobbing B., Experiences with the Partitions Model, Ada User, Vol. 13, No. 2, pp. 79-84, June 1992

[33] Dobbing B., Distributed Ada - A Suggested Solution for Ada 9X, Ada Letters, Vol. 10, No. 9, pp. 94-102, Fall 1990

[34] Dobbing B., I. Caldwell, A Pragmatic Approach to Distributed Ada for Transputers, pp. 200-211, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[35] Doubleday et al., Building Distributed Ada Applications from Specifications and Functional Components, pp. 143-154, Proc. TRI-Ada '91 Conf., ACM, New York, 1991

[36] Eisenhauer G., R. Jha, J.M. Kamrad, Targeting a Traditional Compiler to a Distributed Environment, Ada Letters, Vol. 9, No. 2, pp. 45-51, March/April 1989

[37] Eisenhauer G., R. Jha, Honeywell Distributed Ada -- Implementation, pp. 158-176, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[38] Fisher D.A., R.M. Weatherly, Issues in the Design of a Distributed Operating System for Ada, IEEE Computer, Vol. 19, No. 5, pp. 38-47, May 1986

[39] Flynn S., E. Schonberg, E. Schonberg, The Efficient Termination of Ada Tasks in a Multiprocessor Environment, Ada Letters, Vol. 7, No. 7, pp. 55-76, November/December 1987

[40] Gargaro A., Towards Distributed Objects in Ada 9X, pp. 20-31, L. Collingbourne (Ed.), Ada - Towards Maturity, IOS Press, Amsterdam, 1993

[41] Gargaro A.B. et al., Adapting Ada for Distribution and Fault Tolerance, Ada Letters, Vol. 10, No. 9, pp. 111-117, Fall 1990

[42] Gargaro A.B. et al., Towards Supporting Distributed Systems in Ada 9X, pp. 310-323, B. Lynch (Ed.), Ada -- Experience and Prospects, Cambridge University Press, Cambridge, UK, 1990

[43] Gargaro A.B. et al., Supporting Distribution and Dynamic Reconfiguration in AdaPT, Distributed Systems Engineering, Vol. 1, No. 3, pp. 145-161, March 1994

[44] Gargaro A.B. et al., Supporting Reliable Distributed Systems in Ada 9X, pp. 301-330, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[45] Gentleman W.M., T. Shepard, D.V.P. Thoreson, Administrators and Multiprocessor Rendezvous Mechanisms, Software Practice and Experience, Vol. 22, No. 1, pp. 1-39, January 1992

[46] Goldsack S.J., Session Summary: Recovery and Reconfiguration in Distributed Systems, Ada Letters, Vol. 8, No. 7, pp. 108-112, Fall 1988

[47] Goldsack S.J. et al., Ada for Distributed Systems – A Library of Virtual Nodes, pp. 253-265, Proc. Ada-Europe Intern. Conf., Cambridge University Press, Cambridge, UK, 1987

[48] Goldsack S.J., C. Atkinson, An Object-Oriented Approach to Virtual Nodes: Are Package Types an Answer? Ada Letters, Vol. 10, No. 4, pp. 78-84, Spring 1990

[49] Goldsack S.J. et al., Translating an AdaPT Partition Model to Ada 9X, Ada Letters, Vol. 13, No. 2, pp. 78-90, March/April 1993

[50] Goldsack S.J. et al., AdaPT and Ada 9X, Ada Letters, Vol. 14, No. 2, pp. 80-92, March/April 1994

[51] Hamilton, J.A, D.A. Cook, U.W. Pooch, Distributed Simulation in Ada 95, pp. 105-113, Proc. TRI-Ada '95 Conf., ACM, New York, 1995

[52] Hutcheon A.D., A.J. Wellings, Distributed Embedded Computer Systems in Ada: An Approach and Experience, pp. 55-64, J. Zalewski, W. Ehrenberger (Eds.), Proc. IFIP/IFAC Conf. Hardware and Software for Real-Time Process Control, North-Holland, Amsterdam, 1989

[53] Hutcheon A.D., A.J. Wellings, Ada for Distributed Systems, Computer Standards and Interfaces, Vol. 6, No. 1, pp. 71-82, 1987

[54] Hutcheon A.D., A.J. Wellings, Supporting Ada in a Distributed Environment, Ada Letters, Vol. 8, No. 7, pp. 113-117, Fall 1988

[55] Hutcheon A.D., A.J. Wellings, The Virtual Node Approach to Designing Distributed Ada Programs, Ada User, Vol. 9, Supplement, pp. 35-42, December 1988

[56] Hutcheon A.D., A.J. Wellings, The York Distributed Ada Project, pp. 67-104, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[57] Hutcheon A.D., A.J. Wellings, Elaboration and Termination of Distributed Ada Programs, pp. 195-204, A. Alvarez (Ed.), Ada - The Design Choice, Cambridge University Press, cambridge, UK, 1989

[58] Inverardi P., F. Mazzanti, C. Montanegro, The Use of Ada in the Design of Distributed Systems, J.G. Barnes, G.A. Fisher Jr. (Eds.), Ada in Use, Cambridge University Press, Cambridge, UK, 1985

[59] Jansohn H.-S., Ada for Distributed Systems, Ada Letters, Vol. 8, No. 7, pp. 101-103, Fall 1988

[60] Jessop H.W., Ada Packages and Distributed Systems, SIGPLAN Notices, Vol. 17, No. 2, pp.. 28-36, February 1982

[61] Jha R. et al., An Implementation Supporting Distributed Execution of Partitioned Ada Programs, Ada Letters, Vol. 9, No. 1, pp. 147-160, January 1989

[62] Jha R., J.M. Kamrad, D. Cornhill, Ada Program Partitioning Language: A Notation for Distributing Ada Programs, IEEE Trans. Software Engineering, Vol. 15, No. 3, pp. 271-280, March 1989

[63] Jha R., G. Eisenhauer, Honeywell Distributed Ada – Approach, pp. 137-157, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[64] Jha R., G. Eisenhauer, Distributed Ada - Approach and Implementation, pp. 439-449, Proc. TRI-Ada '89 Conf., ACM, New York, 1989

[65] Keefe D. et al., Pulse: An Ada-Based Distributed Operating Systems, Academic Press, London, 1985

[66] Kermarrec Y., L. Pautet, Ada Communication Components for Distributed and Real-Time Applications, pp. 530-536, Proc. TRI-Ada '92 Conf., ACM, New York, 1992

[67] Kermarrec Y., L. Pautet, S. Tardieu, GARLIC: Generic Ada Reusable Library for Interpartition Communication, pp. 263-299, Proc. TRI-Ada '95 Conf., ACM, New York, 1995

[68] Knight J.C., J.I.A. Urquhart, On the Implementation and Use of Ada on Faul-Tolerant Distributed Systems, IEEE Trans. Software Engineering, Vol. 13, No. 5, pp. 553-563, May 1987

[69] Knight J.C., M.E. Rouleau, A New Approach to Fault Tolerance in Distributed Ada Programs, Ada Letters, Vol. 8, No. 7, pp. 123-126, Fall 1988

[70] Kram R. et al., Ada 83/95 Bindings to OSF's Distributed Computing Environment (DCE), pp. 38-48, Proc. TRI-Ada '95 Conf., ACM, New York, 1995

[71] Krishnan P., R.A. Volz, R.J. Theriault, Implementation of Task Types in Distributed Ada, Ada Letters, Vol. 8, No. 7, pp. 104-107, Fall 1988

[72] Looney M.J., A. O'Brien, Distributed Application designedd Using Mascot and Implemented in Ada, pp. 1-8, J. van Katwijk (Ed.), Ada - Moving Towards 2000, Springer-Verlag, Berlin, 1992

[73] Luckham D.C. et al., Task Sequencing Language for Specifying Distributed Ada Systems: TSL-1, pp. 444-463, Vol. 2, J.W. de Bakker, A.J. Nijman, P.C. Treleaven (Eds.), PARLE - Parralel Architectures and Languages Europe, Springer-Verlag, Berlin, 1987

[74] Lundberg L., A Protocol to Reduce Global Communication in Distributed Ada Tasking, J. of Parallel and Distributed Computing, Vol. 10, pp. 261-264, November 1990

[75] MacDonald J.R., Ada Distributed System Development, pp. 111-121, Proc. 3rd Annual NASA Ada Users Symposium, NASA Johnson Space Center, Houston, TX, 1990

[76] MacDonald J.R., J.D.Johannes, K. Schwan, Ada Dynamic Load Control Mechanisms for Distributed Embedded Battle Management Systems, pp. 156-160, Proc. 1st IEEE Workshop on Real-Time Applications, IEEE Computer Society Press, Los Alamitos, CA, 1993

[77] Mangold K., AMPATS - A Multiprocessor Ada Tool Set, pp. 300-311, J. van Katwijk (Ed.), Ada -- Moving Towards 2000, Springer-Verlag, Berlin, 1992

[78] McFarland G. et al., A Tool Set for Distributed Ada Programing, pp. 71-79, Proc. 3rd Intern. IEEE Conf. Ada Applications and Environments, IEEE Computer Society Press, Los Alamitos, CA, 1988

[79] Moreton T. et al. Tools for the Building of Distributed Ada Programs, Ada Components: Libraries and Tols, Proc. Ada-Europe Conf., Cambridge University Press, Cambridge, UK, 1987

[80] Morris D.S., T. Wheeler, Distributed Program Design in Ada: An Example, Proc. 2nd Intern. IEEE Conf. Ada Applications and Environments, IEEE Computer Society Press, Los Alamitos, CA, 1986

[81] Nakama M., Z. Nakao, A Design and Implementation of an Ada IPC Interface, IEICE Trans. Information and Systems, Vol. 77-D, No. 5, pp. 574-578, May 1994

[82] Nielsen K., Ada in Distributed Systems, McGraw-Hill, New York, 1990

[83] Nielsen K., H. Carlsson, Interprocessor Communication and Ada in Distributed Real-Time Systems, Computer Communications, Vol. 13, No. 8, pp. 451-459, October 1990

[84] Nishida H., T. Itoh, R. Nakayama, Distribution of Ada Tasks onto a Heterogeneous Environment, pp. 155-165, Proc. TRI-Ada '91 Conf., ACM, New York, 1991

[85] Roark C., D. Paul, A Network Operating System (NOS) to Support Real-Time Distributed Ada Applications, pp. 705-710, Vol. 2, Proc. NAECON '90 IEEE 1990 National Aerospace and Electronics Conf., IEEE, New York, 1990

[86] Rogers P., M. Pitarys, The First Embedded Distributed Ada95 Aplication, pp. 270-279, Proc. TRI-Ada '95 Conf., ACM, New York, 1995

[87] Seelye R.W., P.D. Pull, The Effect of Multiprocessor Architectures on Ada Application Software Development, pp. 43-48, Proc. 10th IEEE/AIAA Digital Avionics Systems Conf., IEEE, New York, 1991

[88] Shih C.-S., J.A. Stankovic, Deadlock Detection in Distributed Real-Time Systems and Its Application to Ada Environments, Computer Science and Informatics (India), Vol. 21, No. 1, pp. 1-28, July 1991

[89] Taylor B., Distributed Systems in Ada 9X, Ada User, Vol. 10, No. 3, pp. 127-131, July 1989

[90] Tedd M., S. Crespi-Reghizzi, A. Natali, Ada for Multi-microprocessors, Cambridge University Press, Cambridge, UK, 1984

[91] Thompson C.J., V. Celier, DVM: An Object-Oriented Framework for Building Large Distributed Ada Systems, pp. 179-191, Proc. TRI-Ada '95 Conf., ACM, New York, 1995

[92] Van Scoy R., J. Bamberger, R. Firth, An Overview of DARK, Ada Letters, Vol. 9, No. 7, pp. 91-101, November/December 1989

[93] Van Scoy R., J. Bamberger, R. Firth, A Detailed View of DARK, Ada Letters, Vol. 10, No. 6, pp. 68-83, July/August 1990

[94] Volz R.A., Virtual Nodes and Units of Distribution for Distributed Ada, Ada Letters, Vol. 10, No. 4, pp. 85-96, Spring 1990

[95] Volz R.A. et al., Some Problems in Distributing Real-Time Ada Programs Across Machines, pp. 72-84, J.G. Barnes, G.A. Fisher Jr. (Eds.), Ada in Use, Cambridge University Press, Cambridge, UK, 1985

[96] Volz R.A. et al., Translation and Execution of Distributed Ada Programs. Is Is Still Ada? IEEE Trans. Software Engineering, Vol. 15, No. 3, pp. 281-292, 1989

[97] Volz R.A., P. Krishnan, R. Theriault, Distributed Ada: A Case Study, pp. 15-57, J. Bishop (Ed.), Distributed Ada: Development and Experiences, Cambridge University Press, 1990

[98] Volz R.A., P. Krishnan, R. Theriault, Distributed Ada: Case Study, Information and Software Technology, Vol. 33, No. 4, pp. 292-300, May 1991

[99] Volz R.A., T.N. Mudge, Timing Issues in the Distributed Execution of Ada Programs, IEEE Trans. Computers, Vol. 36, No. 4, pp. 449-459, April 1987

[100] Volz R.A. et al., Distributed and Parallel Ada and the Ada 9X Recommendations, Distributed Systems Engineering, Vol. 1, No. 4, pp. 224-241, 1994

[101] Wellings A., Issues in Distributed Processing Session Summary, Ada Letters, Vol. 7, No. 6, pp. 57-60, October 1987

[102] Wellings A., Session Summary: Distributed Execution Units of Partitioning, Ada Letters, Vol. 8, No. 7, pp. 80-85, Fall 1988

[103] Wellings A., Support for Distributed Systems in Ada 9X, Ada Letters, Vol. 11, No. 6, pp. 61-63, September/October 1991

[104] Wengelin D., L. Asplund, Application of Ada on a Distributed Missile Control System, pp. 300-305, Proc. TRI-Ada '90 Conf., ACM, New York, 1989

[105] Wengelin D., M. Carlson Göthe, L. Asplund, Untitled. (On a portable solution to the problem of suspending a caller on one node during a call to a remote node.) Ada Letters, Vol. 10, No. 1, pp. 97-99, January/Februaty 1990