Erhard Ploedereder

Building Consensus for Ada 9X

Language design is a most difficult task. While the original design of a language has the distinct advantage of filling a blank page, the revision of a language needs to abide by a number of constraints that limit the degree of design freedom. These constraints are both of a technical and a nontechnical nature. In a language revision, each desired change has both a benefit and a cost. Deciding which changes to incorporate into the language becomes a cost/benefit analysis within the framework of the existing constraints. In this article, we will explore some of these constraints and their impact on the Ada 9X revision process.

Why Revise Ada?

Ada [5] has fulfilled many of its original promises. Companies that have invested in the transition to an Ada-based technology have realized better engineering practices, lower error rates, and, above all, higher productivity in the building of large systems [6]. Since the design of Ada more than 10 years ago, advances have been made in the theory and practice of software engineering and its support in programming languages, some of it inspired by the experience with Ada. Further, we now have a decade of experience in using Ada in diverse applications and in implementing Ada for numerous architectures. We have recognized some areas in which the original design of Ada can be improved, and errors, ambiguities, and omissions in the existing standard corrected. Any language needs to evolve over time to reflect the progress in related technologies, to best meet the needs of its user community, and to attract new users. Ada must not be an exception.

In a two-year process, user requests for changes were collected, analyzed, and finally condensed in a Requirements Document, published in its final form in December 1990 [4]. These requirements have provided direction to the Ada 9X Mapping Team, tasked to develop language solutions matching these requirements [1, 2, 3].

Consistency of Concepts

Ada was developed with a number of underlying design principles and concepts. The revisions to the language must be in consonance with this existing overall philosophy of Ada. Extensions need to fit into this philosophy. This not only makes the description of Ada 9X easier, but facilitates the training of personnel in the revised language. Ada engineers should not need to change their style of design and programming in order to use Ada 9X, and must be able to gradually migrate to Ada 9X as they explore and experiment with the newly provided capabilities. We should not require paradigm shifts in the use of Ada, except when directly implied by a requirement. For example, the object-oriented programming paradigm is obviously new in Ada 9X.

Upward-Compatibility

A most stringent constraint on the language revision is the desire to keep the revised language upward-compatible with the existing standard. One needs to take into account the millions of lines of existing Ada code in development, use, and maintenance today which eventually will be migrated to Ada 9X environments. This transition should take place with as little effort as possible (i.e., ideally by simply recompiling the code with an Ada 9X compiler and obtaining a system of equal behavior).

This requirement limits the freedom in revising the language significantly, since Ada 9X essentially needs to become a superset of current Ada. Upward-compatible solutions are preferred over possibly more elegant solutions that would lead to incompatibilities between the two versions of Ada. Still, one also needs to take a long-term view of the usage of Ada. Occasionally, an essential user need cannot be reasonably met without introducing some amount of incompatibility. One then must find a solution least perturbing to the vast majority of existing Ada programs. Incompatibilities that can be easily diagnosed by a compiler and corrected by some automated process are clearly much more acceptable than those that silently alter the execution behavior of programs. For example, the introduction of new reserved words is an incompatible change, since they might coincide with identifiers chosen by the user in Ada programs. Yet, compilers are mandated to diagnose this incompatibility, and tools can be easily written that assist in correcting existing code that incurs this problem.

Time to Market

Many of the perceived problems with Ada were due to the immaturity of early implementations, rather than flaws of the language itself. Some of these perceptions linger, even though many mature Ada implementations are available today and most of the previously identified shortcomings have disappeared.

In revising the standard, we need to prevent a recurrence of this history. Mature Ada 9X compilers must become available in much less time than it took to produce Ada 83 compilers. In this context, any change, even a simplification, carries some cost to the user community. Thus, the impact of proposed changes on existing compilers has been carefully studied, in part by actual trial implementation. A number of changes of obvious, but limited benefit have been withdrawn when the perceived benefits were outweighed by the estimated transition cost.

A similar statement applies to the revision process itself as well. It needs to come to a successful conclusion in an acceptably short time frame, so the user benefits of Ada 9X can soon be exploited.

Language Complexity

Ada is not a simple language. It ad-

dresses software issues that earlier languages chose to ignore, causing complexity to be off-loaded to the users of these languages. Users then had to solve these issues by less elegant extralingual mechanisms. The design of Ada acknowledged the inherent complexity of developing large application systems. Managing this complexity can sometimes be made easier by adding supportive high-level features to a language. Still, these added language mechanisms have created a perception of complexity in Ada by sheer magnitude of the number of provided features.

Introducing additional capabilities and language features, while at the same time maintaining upward-compatibility, adds complexity to the language. It has been a difficult challenge to design these features in such a way that they fit seamlessly into the existing language, thereby keeping added complexity to a minimum despite the gains in expressiveness.

Some issues in software design and development are beyond the state of the art in language design, let alone ready for a single, standardized solution. A language design or revision needs to balance the overall complexity of the language with the feasibility of, and benefits achieved by, standardized solutions within the language.

A distinction needs to be made between the complexity perceived by an Ada novice and that seen by a user transitioning from Ada 83. For the latter, practically any change will, for some time, mean complexity, since a relearning process needs to occur. For the former, some of the changes simplify the language, as unifying concepts have been introduced and some restrictions have been eliminated.

Conflicting User Needs

Different user communities have expressed different and sometimes conflicting needs on the implementation properties of Ada features. In Ada 83, this conflict was partially addressed by leaving certain semantics implementation-defined. This has been a somewhat unsatisfactory solution, since the realization of implementation-defined behavior differed even among compilers targeted at the same application areas. However, conflicting needs are impossible to satisfy by unconditional language rules. An often cited example is the requirement for immediacy of task abortion, which many embedded real-time applications impose, but certain implementations on top of operating systems cannot provide.

Recognizing this problem, the Ada 9X teams have devised an approach that allows for unifying past implementation dependencies in normative annexes to the language standard. Thus, uniformity is brought to implementations that support certain annexes. Each annex addresses a particular class of applications, such as real-time systems, distributed systems, information systems, system programming, or safety-critical systems.

Annexes do not provide additional language features. They merely refine the semantics of features present in the core of the language, standardize on additional pragmas, or provide standardized packages in support of a certain class of applications.

Sign-Up by the User Communities

Different user communities have expressed different needs for changes to Ada. As very attractive change proposals have been presented in the course of evolving Ada 9X, these communities have quite enthusiastically embraced the solutions to their problems. Yet, an equally common theme has been that each community expressed its severe concern over the sum of proposed changes and invariably suggested the elimination of the favorite changes of other groups as the appropriate action. While this is to be expected in such a revision process, the overall high quality and individual attractiveness of the change proposals make it exceedingly difficult to decide which of the changes should ultimately be incorporated in the revised language. The Ada 9X teams have gradually narrowed the selection among previously publicized change proposals for eventual inclusion in the language to satisfy the Ada community as a whole. Yet, individual interest groups will need to agree to the compromise that satisfies their most essential needs, while leaving some of their other concerns potentially unaddressed by the language.

The difficult challenge to the Ada 9X teams and the standardization bodies is to arrive at such a compromise, making Ada 9X acceptable and attractive as a progression of Ada 83 for all user communities.

Sign-Up by the Suppliers

Ada implementers are concerned about the resources required to upgrade their products to conform to Ada 9X, while at the same time maintaining and enhancing their existing Ada products. Ultimately, this concerns the users as well, since the cost eventually must be born by the marketplace. Initially, however, vendors must worry about financing the upgrade to Ada 9X. Only if this cost is affordable, will the market be provided with a satisfactory number of quickly maturing Ada 9X compilers.

At the same time, Ada implementers are concerned that Ada 9X must be attractive enough to their entire existing customer base, so that transitioning by the user community can be expected and enhancements of the Ada 83 product lines eventually discontinued.

This dilemma is not unique to Ada; the revision of any language standard will force compiler vendors to perform some retooling. Recognizing the issue, the project sponsors have developed a validation strategy that speaks directly and realistically to compiler vendor concerns. Rather than the "all or nothing" approach to validation that was used from the outset for Ada 83, the transition policy for Ada 9X will allow a compiler vendor to invest first in those upgrades that are of most benefit to his or her particular customers. A vendor with an embedded systems market may thus choose first to implement protected types or the Real-Time Annex. A vendor oriented toward the information systems community may choose instead to first support the Information Systems Annex.

It must be emphasized that this approach is strictly for facilitating a

smooth transition to Ada 9X and does not contradict the long-standing Ada policy of "no subsets / no supersets." After a transition period of less than three years, full compliance with the Ada 9X core language will be required, and enforced through appropriate validation test suites.

Sign-Up by Educators

One goal the Ada program never truly achieved is a universal acceptance of Ada in academic curricula. Several measures are in place to make Ada 9X an attractive language candidate for the academic community. On the technical side, the inclusion of object-oriented programming paradigms as well as other enhanced software engineering support should cause added interest in Ada 9X both for teaching and research. The availability of Ada 9X will be greatly helped by the GNU NYU Ada Technology project, which is developing a freely available compiler that will make Ada 9X much more accessible to colleges and universities than Ada 83 ever was. This compiler will be available as early as the fall of 1993.

Still, there is considerable investment in such Ada material as 83 courseware and textbooks. One aspect of evaluating Ada 9X changes is to assess their impact on these existing investments and the cost of their upgrade to Ada 9X.

Building the Consensus

Obviously, revising any standard is an exercise in compromise among the various interested parties. The Ada 9X process has created a number of approaches to assess the viability of the proposed changes, both individually and collectively, before seeking approval of the revised standard.

First, the Ada 9X Mapping Team is in itself a consensus-forming group prior to proposing any particular change. Through the Ada 9X User/Implementor teams and the Implementation Analysis team, assessments are obtained on the consistency and completeness of the proposal in its interactions with other features of language, on the impact on tool implementations, and on the consequences for old and new application code. The Language Precision team evaluates changes from the viewpoint of formal semantic models.

A wider evaluation occurs by the Ada 9X Distinguished Reviewers. This group comprises language interpretation and implementation experts, educators, Ada suppliers, and highly qualified representatives of various application domains, such as hard and soft real-time systems, distributed systems, information systems, numerics, safety-critical systems, and large system design. Through daily electronic exchanges and quarterly meetings, this international group with 29 members from six countries evaluates the technical aspects of the proposed changes, as well as the necessary overall tradeoffs.

The Distinguished Reviewers have been joined by the Volunteer Reviewers, yet another, larger group of reviewers that have expressed a desire to be closely associated with the effort on a day-to-day basis, commenting on the various change proposals and their evolution.

Finally, the proposed changes are presented to interest groups and the general public in open national and international meetings, both to keep the community informed about the progress made, and to take guidance from the communal reaction to the evolution of the changes.

In April 1992, ISO/IEC JTC1/SC22 WG9, the working group tasked by ISO to address Ada standardization issues, met in Germany for a one-week meeting. The 12 represented countries voted unanimously to approve the Ada 9X Mapping Specification with some changes, and with a small number of remaining topics still to be studied and resolved. This vote was significant, as it represented the international go-ahead to initiate the writing of the revised language reference manual.

Ada 9X for Various User Groups

Readers of this article may well ask the question: What is in Ada 9X for me? Let me therefore attempt a characterization from the vantage point of some of the proposed changes as of the summer of 1992.

System programmers will find a number of added capabilities to better interface with existing software written in other languages, which in turn will facilitate the generation of language bindings to other standards. The object-oriented programming extensions to Ada will make it easier to extend existing packages through data type extensions and method refinements. Several aspects of low-level programming will be improved and better control over memory management will be available.

The real-time community will find better support for fast synchronization and communication among tasks. Also, better control over task priorities, order of entry selection, and the scheduling of tasks will be provided.

Producers of large systems will have the ability to provide data type abstraction through multiple packages, implementing the operations on a private type. This enables the construction of logical subsystems with a hierarchy of library units rather than a monolithic package, thus allowing for better modularity and offering reduced recompilation costs.

Information systems developers will have decimal types supported by the language and various rules that are intended to provide better compatibility with Cobol programs and databases.

Numeric applications will be able to rely on standard packages for primitive and elementary numeric functions and procedures and will benefit from some improvements for generic units. Several problems noted with the Ada 83 model of fixed and floating-point numbers will be addressed.

The international Ada community will find the changes necessary to support national character sets in 8- and 16-bit representations.

For safety-critical applications, the overall effort of narrowing the implementation-defined semantics of certain constructs, including the semantics of "erroneous execution," will be of some help. The work of the Language Precision team to formalize the definition of those parts of the language that are particularly rele-

vant to safety-critical applications deserves special mention.

For distributed systems, certain constraining rules of Ada 83 will be relaxed to facilitate the distribution of Ada programs across multiple processors. A partition concept will be applied and a mechanism for communicating among partitions, both via shared memory and via remote procedure call capabilities, will be supplied.

Summary

The Ada 9X effort, like most other standards activities, is an exercise in achieving a compromise, balancing the needs of different user groups among one another and against cost, time, complexity, and technical feasibility factors. After all the necessary trade-offs, Ada 9X will provide a number of exciting, well-integrated enhancements to the language. It will reflect the progress that has occurred in the design of programming languages to support the engineering of large software systems in a broad spectrum of application domains, ranging from real-time, embedded or distributed systems to information systems.

Acknowledgments

I appreciate the comments received from Tucker Taft and Ben Brosgol on earlier drafts of this article. They helped to improve its quality considerably.

References

1. Ada 9X Mapping Document, vol. II, Mapping Specification, Version 4.0., Office of the Under Secretary of Defense for Acquisition, US Department of Defense, Dec. 1991.
2. Ada 9X Mapping Document, vol. II, Mapping Specification, Annexes, Version 4.0., Office of the Under Secretary of Defense for Acquisition, US Department of Defense, Mar. 1992.
3. Ada 9X Mapping Document, vol. I, Mapping Rationale, Version 4.1., Office of the Under Secretary of Defense for Acquisition, US Department of Defense, Mar. 1992.
4. Ada 9X Requirements. Office of the Under Secretary of Defense for Acquisition, US Department of Defense, Dec. 1990.
5. Reference Manual for the Ada Programming Language. ANSI/MIL-STD-1815A-1983, US Department of Defense, Feb. 17, 1983.
6. Reifer, Don. SoftCost-Ada: User experiences and lessons learned at the age of three. In *Proceedings of TRI-Ada'90* (Baltimore, Dec. 1990), ACM, ISBN 0-89791-409-0.

CR Categories and Subject Descriptors: D.2.0 [**Software Engineering**]: General—*standards*; D.3.0 [**Programming Languages**]: General—*standards*; D.3.2 [**Programming Languages**]: Language Classifications—*concurrent, distributed and parallel languages, Ada, Ada 9X*; D.3.3 [**Programming Languages**]: Language Constructs and Features—*abstract data types; concurrent programming structures, modules, packages*

General Terms: Languages, Standardization

Additional Key Words and Phrases: Asynchronous communication, building consensus, classwide programming, compatibility, hierarchical name space, inheritance, language design, language evolution, polymorphism, program library, programming in the large, protected type, separate compilation, synchronization, tagged type, tagged extension, upward compatibility

About the Author:

ERHARD PLOEDEREDER is professor for compilers and programming languages at the Universitaet Stuttgart, Germany. Until recently he was vice president for technology at Tartan Inc., Monroeville, Pa., USA. He chairs the Ada 9X Distinguished Reviewers. His research interests include programming language design, code optimization, and programming environments. **Author's Present Address:** Universitaet Stuttgart, Breitwiesenstr. 20-22, D-7000 Stuttgart 80, Germany; ploedere@informatik.uni stuttgart.de