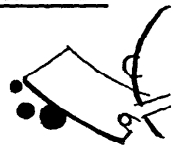
報　告



パネル討論会

Ada とどうつきあうか

昭和56年後期第23回全国大会† 報告

パネリスト

筧　捷彦[1]，坂村　健[2]，近山　隆[3]
安村　通晃[4]，**司会**　和田　英一[5]

1. ことのおこり

Adaに至る米国国防省の一連の計画を筆者がはじめて知ったのは，ACM Sigplan Notices 1975年9月号にのった Robert G. Estell の投書によってであった．それによると国防省は実時間プログラム用の言語を検討しはじめ，それがきまると Fortran や Cobol と同程度の影響力をもつと考えられる．世のプログラム言語の専門家は，悪い言語が選ばれないよう注意していてほしい．意見があれば自分なり（投書主は米海軍所属）国防省なりへ送ってほしいというのであった．

日本では積極的な動きはなく，この計画の内容が次にわかったのは，やはり Sigplan にのった Steelman の記事であった．また Dijkstra による4色言語の批判や CMU で設計した Tartan 言語も Sigplan にのり，作業のすすみ具合いがはっきりしてきたが，1979年6月号の Sigplan は Ada の文法書そのものであり，ついに Ada の全容を知ることができた．

我々の研究室ではさっそくプログラム言語の専門家として Ada 文法書を精読し，矛盾や曖昧と思われる点について国防省へ報告した．同時に情報科学科の米田研と一緒に処理系の試作に着手した．

Ada設計グループは暫定版文法書に対して寄せられた意見を基に改訂版を作る作業を開始したが，我々も意見を送ってあったため，日本でも改訂版の草稿を読んでほしいという要請が Ichbiah から筧を通じて寄せられた．そこで1980年の春頃，次々と送られてくる草稿に目を通しては意見を返すという作業がつづいた．この活動が改訂版の序文にある Tokyo Study Group である．

† 日時　昭和56年10月14日，15：00～17：00
　場所　東京大学

1) 立教大学，2) 東京大学，3) 東京大学
4) 日立製作所，5) 東京大学

改訂版は80年9月頃出版され，複製は共立出版から bit の別冊として出版されたから，読んだ人も多いと思う．改訂版については処理系を作成していないがそれは暫定版のときは，文法書をよく読む方便として処理系を作ったためだからである．

2. Ada とのつき合い方

うえにのべたように我々のこれまでの Ada とのつき合い方は，使うためというよりも，Adaは影響力の大きそうな言語だから，変な言語にならないようにできるだけのことはしようという動機によるものであった．10年ほどまえに Pascal をどんどん講義に使ったように Ada をとり入れるかというと，どうもそうはなりそうもない．それは Ada がサブセットを許さないからで，かくも難解な文法を学生に全部読んで貰うわけにはいかない，また処理系もすぐ使えるようになるかどうかわからない，というような理由による．しかし一方，パッケージや並列処理の説明も，情報工学のような学科ではなんらかのプログラム言語を使って教えなければならないので，その辺では Ada 風の記述を使うことも十分考えられる．Ada全体はやはり玄人プログラマの言語であり，我々が個人的に使ったり，学部学生に教えたりするのには，あまりにも重たいと思う．

3. パ　ネ　ル

今回のパネル討論は Ada の要求仕様書にくわしい日立製作所の安村君，Adaの暫定版の処理系を作ってAdaの文法規則の実現法を工夫したし，文法にも非常にくわしい東大の近山君，また Ada では支援環境も重要なシステム構成要素になっているが，それに通じている立教大学の筧君，さらに最近，Intel が発表した Ada 用マイクロプロセッサ iAPX 432 をよく知っている東大の坂村君にパネリストをお願いし，それぞ

れの分担のところからみて Ada とどうつき合いたいか，個人的感想をのべていただきたい．Adaがつき合う相手として，良い友達か，悪い友達か，あるいは普通の友達かは，参加者が自分できめてほしい．

（司会　和田　英一）

支援環境の面から

筧　捷彦

米国国防総省が巨額の金をつぎ込んで開発を進めている Ada 企画では，言語設計・言語処理系だけにとどまらず，その支援環境の設計・開発が同時に推し進められている．その支援環境設計の特長を紹介し，我我のつき合い方を考えてみる．

1. Ada と支援環境

組込み型計算機用として開発された言語 Ada は，また汎用の記述言語としても優れた能力を持つことになった．譜包 (package) の導入，分割翻訳の導入などは，実用的な汎用言語として初めてのものといってよい．大型の譜構 (software system) を構築する際の有効な道具となるであろう．

Ada開発では，言語設計・言語処理系（翻訳子）にとどまらず，同時にその支援環境の設計・開発も進められている．これは，これまでの言語開発では見かけられなかった特徴である．ある言語がいかに優れたものであっても，適切な支援環境なしではその能力は生かし切れない．支援環境の良し悪しは，その言語の有効性を大きく左右する，という最近の考え方を採り入れての結果であろう．特に，譜包と分割翻訳との組合せは，支援環境があって始めて生きてくる．

言語 Ada の開発においては，(1)要求仕様書の確定，(2)（言語）仕様書の確定，(3)具現，という手順を踏んでいる．(2)以降は数社に競争させて，その内の最良のものを選ぶ．国防総省としては，省内での統一言語制定ということに重点を置いていたから，(2)までは総省自身がその直接の管轄に当り，(3)の段階では，配下の各軍に担当させている．

支援環境に関しては，総省内で統一的な仕様を定めることまでは意図していないように見受けられる．1980年に，要求仕様書 Stoneman を確定した後は，(2)，(3)の段階を一括して各軍に委ねている．

各軍（陸・海・空）相互には，技術水準の差，あるいは高級言語による開発や支援環境の利用といったことへの態度の著しい差がある．現状で支援環境の統一仕様を作ってみても，それが受け入れられる素地があるわけではない．

こうした背景をもとに，要求仕様書 Stoneman は，(a)省内での技術移転を推し進めるための素地作り，(b)従来的な作譜環境をも容認する寛容さ，(c)進んだ考え方を持つ層への指針，といったものを包括した形でまとめられている．

2. 要求仕様の概要

Stoneman では，支援環境を APSE (Ada Programming Support Environment) と呼ぶ．その基本構想は (1)APSE は開発用の親計算機に置き，対象となる組込み用計算機の譜構の開発にあてる；(2) APSE の各道具類は，Ada によって記述する；(3)核となる部分は，界面 (interface) を確定することにとどめ，既存の譜構を利用する道を残しておく；(4)従来の作譜環境にも見られた道具類は，APSE に必須のものとする；(5)進んだ道具類，あるいは一定の方法論に基づいた高度の支援環境も APSE として扱えるだけの余地を残す，ということにある．

界面を，KAPSE (Kernel APSE) と呼ぶ．その中心は，様々な情報を貯え管理できる料基 (data base) とのやり取り，入出力装置とのやり取り，道具とのやり取り，道具間でのやり取りである．これらの界面は言語 Ada での譜包仕様 (package specification) として記述することを要求している．

翻訳子・結合子 (linker)・虫取り子 (debugger) 等の道具も，この界面上で Ada によって作成される．このため，Adaで書いた算譜に関して，その原譜・中間語形・目的語形あるいは実行形式についても，譜包仕様で定めておくことを要求している．

料基には，算譜・資料を含めあらゆる形の情報を収める．各情報には，履歴等の属性を付加して管理する．少なくとも，版 (version) の管理，群 (configuration group) の管理のほか，区分 (partition) を置いての管理ができることを要求している．

必須の道具類だけを備えた支援環境を，MAPSE (Minimal APSE) と呼ぶ．MAPSE には，翻訳子，結合子，虫取り子のほか，文面編集子，指令通訳子といった道具に加えて，変数の代入参照関係・副譜の引用関係を呈示する程度の解析子，あるいは料基についての履歴情報を呈示する程度の道具を置く．また，MAPSE には，対象機用の入出力といったごく基本的な備譜 (library) を置くことを要求している．

Ada 言語向きの文面編集子，高級言語風の指令通訳

子，あるいはさらに進んで，検証のための道具といったものは，すべて APSE の可能な道具として列挙するにとどめている．また APSE の形態として，こうした道具類を単にとりそろえただけのものではなく，ある一定の方法論を採用した支援系をも許容することが述べてある．

3. どうつき合うか

言語 Ada の開発過程においては，我々が関与する余地があった．要求書の作成，仕様書の作成の各段階において，国防総省は広く内外の意見を求める手順を踏んだからである．また，現在開発中の翻訳子も，しかるべき価格で広く頒布することをうたってもいる．

支援環境については，これに比べると，ほとんど我我の関与する余地がない．現在すでに何社かとの契約のもとで開発が進められているが，その仕様の詳細は知られていない．また，開発のいきさつからしても，これらの支援環境を我々が利用できるようになるとは思えない．

支援環境——特にひとつの汎用言語にまとをしぼっての支援環境のあり方についての研究は，比較的手薄の分野であった．米国でのこの大規模な開発を，ひとつの見本として，我々は我々なりの支援環境のあり方を研究していく，というのが基本的なつき合い方だといえよう．

参 考 文 献

1) DoD: STONEMAN—Requirements for Ada Programming Support Environments (1980).
2) 筧 捷彦: Ada と支援環境，情報処理，22-2 (1981).
3) Ada に関する調査，工業用計算機システムのソフトウェアに関する調査報告，第Ⅰ部，日本電子工業振興会，56-A-181 (1981).

Ada とアーキテクチャ

坂村 健

Adaのような抽象データ型言語は，データ表現とそれに対する基本操作群をひとまとめにした抽象データ型 (abstract data type) のサポートを持ち，広義の意味でのモジュール・プログラミングを許すため，ソフトウェアの生産性を上げ，また高信頼化プログラムの作成を可能にする．しかしながら，これらは言語の特質を言語使用者から見た場合であり，処理系の作成はともかく，効率の良いオブジェクト・コードを生成するのは現在のマシンの上では難しいだろう．もし従来のいわゆるノイマン型計算機に実現するとオーバヘッドが大きいと想像される．

実用にするには少なくとも，①パッケージ（ドメイン），②実行時における抽象データ型，strongly typed，③ローカル・グローバル変数（リファインメント），④桁数指定を行うための可変精度計算，⑤constraint，⑥access type，⑦assign statement などの言語特有の性質を実行時に積極的にサポートするような機構が必要であろう．たとえば，「対象」の型や属性が正確に表現でき，また個々の「対象」が一意的に識別するようなアドレシング，いわゆる capability addressing の実現を積極的にサポートするような機構，種々生成されるオブジェクトを管理するためのガーベージ・コレクション機構，論理アドレスから物理アドレスへ高速にマップする機構が必要である．すなわち，必要とされる機構のほとんどはメモリに関するものであって，多種多様の型が取り扱えて，しかもそれの大きさもダイナミックに変化するというようなメモリを設計しなければならない．また，レジデュアル制御，ビットアドレシングなどの機構も有効であろう．

具体的なインプリメンテーションの手段は色々考えられる．知られているものとしては，IBM の System/38 並びにインテル社の iAPX 432 などがある．そして，iAPX 432 は Ada をとくに意識してデザインしたという．たとえば，①オブジェクトを基本的な操作対象としたアーキテクチャをとるためセグメント，オブジェクト，ドメインという階層で管理を行うこと，②capability addressing の実現のため，メモリ・アドレシングの方式として，ディスクリプタを採用していること，③命令フォーマットもビット単位に可変長とし，orthogonal な構成を取り高級言語指向の命令形式を取っていること，④ダイナミックなメモリ割り付けのサポートを行っていること，⑤メモリを構造化したセグメントとして管理し，2^{32} もの巨大仮想空間をサポートできること，⑥ソフトウェア・トランスペアレントなマルチプロセッサ構成を取れること，⑦スタック，ストレージの動的割り付け，環境の動作的切り換え，メッセージ・データセグメントを通信ポートオブジェクトを通して転送する高級命令などがあることである．

これらの具体的な実現方法は公表されていないため不明ではあるがチップ集積密度などから推測するにそのほとんどはファームウェアによるものであろう．効率に関してはインテル社の発表によれば，iAPX 432

は1台で 0.5 MIPS, 2台で 1 MIPS, 5台で 2 MIPS ということである.（参考までに, PDP 11/34 が 0.2 MIPS, VAX 11/780 が 1 MIPS, IBM 370/158 が 2 MIPS である.）また, マルチプロセッサ構成のパフォーマンスとメモリバスの本数に関しては, 1メモリバスの場合は4台まで台数とパフォーマンスは比例するが, これ以上増やしてもパフォーマンスは上昇しない. 2メモリバスの場合は8台位で, 4メモリバスの場合は15台位で飽和するというデータがある.

望ましい, より効率の良い進んだ形としては抽象データ型データ構造を効率良く取り扱えるような機構を備えることであり, それには従来のような1次元アドレシングではない強化されたメモリアーキテクチャが要求されよう. タグ付きデータ, スタック, ヒープなどを効率良く実現するようなもの, たとえば機能メモリ, 連想メモリをどのように構造に取り入れるかが重要である. また, データの多様性に対処するため, 種々の応用指向型プロセッサの開発も重要となろう.（これをたとえばユニバーサルホストマシンで構成することは現実的である.）また,「対象」ごとにプロセッサを割り当てるようなマルチプロセッサ形態のアーキテクチャなどがインテル社 iAPX 432 の実験からも判るように効果的である. いずれにせよ, 以上実現のためにはより進展した VLSI 技術を必要としよう. 80年代初頭の現在は残念ながらファームウェア・ベースが良いところである. iAPX 432 は, 先陣を切ったということで, 一つのコマーシャル・スタンダードであろうか. なお, Adaインプリメンタにとっても効率の良いコードを生成する処理系を現在のマシン上に作るのは「重い」と感じる人も多いはずで, このような意味でも Ada といえばまず新しいマシンありきということなのであろう.

なお, Adaということをもう少し広く捉え, 抽象データ型言語とし, それを一般的にサポートするようなマシンを考えることはアーキテクトにとっても興味深い研究テーマである. 特に並列処理を考えた場合大きくいえばアクターマシンを作るようなことになり, 解決されなければならない多くの問題を含んでいると思われるが, 抽象データ型マシンは次世代コマーシャルマシンとしても有望であると考えられるので, 80年代は VLSI の研究進展と合わせ多くのアイデアが出され, 実際にマシンが開発されると思われる.

Ada の守備範囲

近山 隆

Adaはこれまでに広く使用されてきたどの汎用プログラム言語にも見られなかった機能を数多く持つ. 近年普及が著しい Pascal には比較的よく似た文法をもつが, 以下のような機能は Pascal にもない.

分割翻訳 (separate compilation)
譜　庫 (program library)
譜　包 (package)
汎用譜 (generic unit)
多重定義 (overloading)
表現指定 (representation specification)
タスク機能 (tasking)

しかし, こうした新しい諸機能は処理系の実現の容易さを配慮した上で言語に取り入れたものであるから, 処理系はあまり大規模なものにはならずに済むであろう.

Adaの文法は, 現在の計算機ハードウェア, 操作システム, 言語処理系作成技術の水準での高効率な実行を目指した文法設計となっているので, 算法の論理的記述に必要なもの以外の情報をプログラム中に記す必要がある場合や, 効率上の理由から文法に制約を生じた点も見られる. たとえば, 譜包仕様中の密閉部に書く型定義などがこれにあたる. こうした一見例外的な文法規則は, 処理系の実現法を念頭に置かずには理解しにくいので, 言語の習得の障害になりやすい. しかし, これとても従来のプログラム言語の多くよりは制限が小さくなっており, 改良が認められる.

Adaは初心者の教育用には不向きであろう. Adaの文法は大きすぎるし, 部分言語を作ろうとしても各機能は補完関係にあるので作りにくい. たとえば, 汎用譜と譜庫の機能なしには入出力すらできない.

あまり大きくないプログラムを Pascal と Ada の両者を用いて書いて見れば, Pascal の方がはるかに簡潔に記述できるのがわかる. 両者の文法は似た点が多いが, Adaの方がより厳格で, 書かねばならないことも多い. 千行程度までの規模のプログラムの開発には Pascal を用いた方が高い生産性を得られそうである. それ以上の規模になると, Adaのもつ抽象化機能が有効に働きはじめ, さらに大規模になれば分割翻訳や譜庫の機能による共同作業の能率向上も加わり, 両者の関係は完全に逆転するであろう.

Adaの多重定義などの機能は，使い方を誤まるとはなはだ読みにくいプログラムを生ぜしめるのではあるまいか，という不安感は一部に強いようである．事実わかりにくく書こうと思えばいくらでもわかりにくくできるが，逆に上手に多重定義を用いればむしろ理解の一助となることは，数学で用いる記法を見てもわかる．どのようなプログラム言語でも，下手に書こうとしても下手にできないような例はない．重要なのは上手に書けば上手に書けることである．つまり，わかりやすい算法の抽象構造を，できるだけ直接的に表現できるような言語が望ましい．この点，いく分未整理な点は散見されるものの，Adaの抽象化機能は従来の言語よりはるかに豊富で，抽象レベルから実際のプログラムへの写像が直接的に行いやすいし，そのためプログラム自身のドキュメントとしての価値も高くできる．もちろん抽象的レベルでの算法がわかりやすくできていなければ，プログラムもその混乱をひきつぐだけであるから，Adaは高レベルでの設計の巧拙が顕われやすい言語であるといえる．

結論として，Adaはプログラム開発の専門家が，長期にわたって保守を必要とする，大規模なプログラムを，共同作業によって開発する際に，もっとも真価を発揮するプログラム言語である，と筆者は考える．

Ada の設計思想とつき合いかた

安村 通晃

1. はじめに

プログラミング言語 Adaは1980年代の新しい言語として注目を集めている．ここでは，まず，Adaの成立過程と適用目的を明らかにして，Adaの特徴と性格を浮き彫りにする．次に，要求仕様書に込められた設計思想と実際の言語との対比を行う．最後に，個人としての Ada 観と Ada とのつき合い方を述べて締め括る．

2. Ada の成立過程

米国国防省 (DoD) では，軍用組込みシステムのための共通高水準言語制定の委員会 (HOLWG) を1975年に発足させ，以来そのための要求仕様書制定の作業と既存言語評価の作業を続けてきた．要求仕様書は，Strawman (1975) に始まり，Woodenman (1975)，Tinman (1976)，Ironman (1977) を経て，Steelman (1978) まで改訂されてきた．一方，既存言語の評価の結果，要求を満す適当な言語がないと判断され，PL/I, Algol 68, Pascal のいずれかをベースにした新言語を作るべしとの勧告が出された．この方針に従い，4社が新言語の提案を行った．それらは，カラーコードで，Blue 言語，Red 言語，Green 言語，Yellow 言語と呼ばれる，いずれも Pascal をベースとした言語であった．このうち，Green 言語が最終選定され，1979年6月に Preliminary Ada として世に出た．この Preliminary Ada に対して，世界各地からのコメントを受けて改訂されたのが1980年7月に仕様が明らかにされた Standard Ada[2]である．

3. 組込みシステム (Embedded System)

米国国防省では，ソフトウェアだけで一年間に約30億ドルも使っていると推定されるが，その過半数 (56%) が組込みシステム用であり，データ処理 (19%) や科学計算 (5%) の割合は少ない．組込みシステムとは，電子機器，兵器，船，航空機，通信機器等に組み込まれたシステムのことであり，したがって高度の信頼性が要求される．また，そのソフトウェアの特徴としては，プログラムの規模が 50 ksteps ないし 100 ksteps と大きく，かつ10年から15年の長期にわたって使用される点にある．ところが，これまでは，組込みシステム用には，450 もの汎用言語またはその方言が使われてきた，という問題があった．

4. Ada の特徴と性格

以上述べてきた点から，Ada の特徴は，

(1) 組込みシステム用であること，
(2) 大規模プログラム開発向きであること，
(3) 最近のプログラミング (言語) 方法論を反映していること，

が挙げられる．すなわち，Adaは組込みシステム用として，並行処理，例外処理，機械依存の指定機能等を含むし，大規模プログラム開発向きの，分割コンパイルやパッケージ等の機能をもつ．また，プログラミング (言語) 方法論の成果として，言語設計論等を援用した高信頼化の達成をねらっている．

一方，Ada の性格としては，

(1) サブセットや方言を許さない，
(2) 検定 (validation) を通ったもののみが，Ada とよばれる，
(3) コンパイラは DoD から低価格で提供される，

ことなどにより，統一のとれた互換性の高い Ada の普及をねらっている．

5. Ada の設計思想と実際の言語

Steelman[1]などの要求仕様書に盛り込まれた言語の設計思想は次の通りである：

（1） 汎 用 性 (generality)
（2） 信 頼 性 (reliability)
（3） 保 守 性 (maintainability)
（4） 効　　率 (efficiency)
（5） 簡 潔 性 (simplicity)
（6） 実 現 性 (implementability)
（7） 機械独立性 (machine independence)

このうち，(1)については，組込みシステムとして必要な機能を含むこと，(2)については，コンパイル時の型検査等，(3)については，読みやすさの強調，(5)では，複雑さを避け，一様性 (uniform) を保つこと，などが内容として挙げられている．

実際の言語では，要求仕様のなかの機能はほぼ達成されており，(2)以下の項目に比べ，(1)の汎用性が最も強調された仕様となっている．なかでも，特に簡潔性が後退しており，言語が大規模・複雑になった結果といえる．

6. Ada への期待と不安

Adaに対しては，次の3点で大きな期待がある．まず第一に，Adaが並行処理・例外処理・機械依存の指定機能を有することから，制御用・システム記述用の標準言語候補としての期待が大きい．これらの分野では，従来適当な標準言語がなかっただけに，Adaの標準化と普及が進んだ段階では影響が大きい．第二に，大規模プログラム開発用として，Adaがパッケージ（抽象データ型）や分割コンパイルの機能をもつ点である．これらの機能は，従来，実験用言語かローカルな言語にしかなかった．最後に，コンパイル時の型検査や実行時の各種の例外検出などの高信頼性の言語仕様は，長い目でみたプログラミングの生産性向上にもつながり，期待が大きい．

一方，Adaに対する不安として，複雑でかつ大規模であることなどへの懸念がある．たとえば，汎用体 (generic) や多重定義 (overloading) はネストした有効範囲と組み合わせると相当複雑になり，プログラマやコンパイラに負担を強いる．また，これらの汎用体や多重定義は，本来の必要性からすればごく僅かであるのに対し，これらを一般的に言語に取り込むことにより，言語仕様全体が一挙に膨らんだ感がある．さらに，省略を数多く持ち込んで，書きやすさを強調している点などへも不安を感ずる．

7. Ada とのつき合いかた——おわりに代えて

Adaは，単機能としても，また従来なかった組合せとしても，新しい試みがいくつか含まれている．このため，まず，Adaの本格的な処理系の登場を待ってその使用経験を積むことが必要であろう．筆者個人としても使ってみたいし，さらに大規模なプログラム開発報告も聴いてみたい．それまでは，Adaに対して，軽率な判断は慎みたい．

次に，Adaに含まれる言語機能のうち有効と思われる機能は，他の言語にも取り込んでいく必要があろう．たとえば，Adaの方式での分割コンパイルがうまくいくとすれば，他言語（の処理系）へも適用すればよい．

最後に，Adaに含まれる新しい言語の動きに対する技術的理解は今後も続けていく必要があるだろうし，また，具体的に Ada のプログラム・パッケージがユーザから持ち込まれたときの態勢も徐々に整えていく必要があると思われる．

参 考 文 献

1) DoD: Steelman, Requirement for High Order Computer Programming Languages, p. 20 (1978).
2) DoD: Reference Manual for the Ada Programming Language, Proposed Standard Document (July 1980).
3) Carlson, W. E. et al.: Introducing Ada, Proc. of ACM Ann. Conf., pp. 263-271 (Oct. 1980).