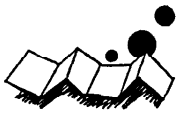
解　説



最近のAdaの動向†

石畑　清††　筧　捷彦†††

1. はじめに

日本ではAdaに関する活動が一向に活発にならないが，欧米では処理系やプログラミング支援環境の作成など盛んな活動が行われている．軍関係を中心として，Adaで書かれたソフトウェアの蓄積も相当量に上っているはずである．

本稿では，欧米におけるAdaに関する活動の現状を言語仕様，処理系，支援環境，関連諸団体の四つに分けて解説する．

Adaの欧米での利用状況についても書くよう求められたが，筆者らの手元にはこれに関する資料がまったくないため，割愛した．

2. 言語仕様

Adaの言語仕様は，1983年2月にANSI規格[1]となった時に固定されている．したがって，新機能の追加などの大きな動きはない．しかし，現在の文法書の不明確な点，矛盾した点などを修正する作業は続けられている．この作業には実現困難な機能の手直しが含まれているので，言語自体もごくわずかではあるが変わってきている．

ここでは，最初にAdaの言語仕様を議論している組織を紹介し，次に議論されている仕様の詳細について述べる．

2.1 組　織

現在，Adaの言語仕様に関する議論は，二つの組織の下で行われている．一方はISO（国際標準化機構），もう一方は米国国防総省（以下，DoDと略す）である．

Adaの国際規格化は1980年にANSIから提案されていたが，ISOとDoDの間で意見の食い違いがあり，なかなか作業を始められなかった．その原因は，AdaをDoDがほとんど支配する形になっていたことに対するISO側の反発である．DoDは，Adaを商標として登録し，その使用に認可を要求していた．また，DoDの検定を通ったものしかAdaの処理系として認めないという態度をとっていた．ISOは，これらの方針がISO規格にまで及び，国際規格化の作業そのものがDoDの意向によって左右されるのではないかと懸念した．そこで，ISOにおける作業はISOの規定どおりに行うこと，商標や処理系検定はISO規格に適合するための条件としないことを認めるようDoDに要求した．DoDは，当初この要求に反発していたが，結局ほぼ全面的に譲歩することで解決を見た．

ISOにおけるAdaの国際規格化の作業は，1984年に始まった．作業を担当しているのは，TC 97のSC 5-WG 14であったが，ISOの組織変えにともなってSC 22-WG 9と名前が変わった．

DoD側でAdaの言語仕様に関する責任を持っているのはAda Boardである．Ada Boardは，Adaの規格化や言語仕様の問題に関して，AJPO (Ada Joint Program Office) を補佐する立場にある．DoDの中のAdaに関する作業はAJPOが統轄しているが，Ada BoardはAJPOとは独立した組織としてある．

Adaの言語仕様に関する詳細な検討は，言語保守委員会 (Language Maintenance Committee, 略してLMC) で行われている．ここで，世界中から送られてきたコメント（処理系作成者からのものが多い）をもとに検討が行われる．LMCは，Ada Boardによって作られた組織であるが，現在はWG 9とAda Board双方共通の下部機関となっている．これは，作業の重複を避けるためである．ISO規格とANSI規格の完全な一致を図るためにも有効であろう．

LMCの検討結果に関する報告は，WG 9とAda Boardの合同の会議に送られ，そこで投票にかけられる．ただし，会議は合同でも投票はWG 9とAda

† Recent activities on Ada by Kiyoshi ISHIHATA (Department of Information Science, Faculty of Science, University of Tokyo) and Katsuhiko KAKEHI (Department of Mathematics, Faculty of Science, Rikkyo University).
†† 東京大学理学部情報科学科
††† 立教大学理学部数学科

Board それぞれ独立に行われ，両方の承認を得られなければ LMC に差し戻しになる．Ada Board を通った決定は，さらに AJPO に送られ，そこでも審議される．なお，Ada の設計者 Ichbiah も LMC のメンバであるが，今やその一員であるに過ぎない．その意見が多数意見に敗れることもある．

日本では，情報処理学会の規格委員会の下に Ada ワーキンググループが結成されている．ISO から投票の求めがあった時には，このグループで審議されることになる．しかし，今までのところこのグループの活動は不活発である．

2.2 検討作業の概要

現在のところ，LMC などで行われている作業は，Ada 文法書に手を加えることを目的としたものではない．1986年末までは，現在の文法書で説明のつかない部分に適当な解釈をつけることに専念する予定である．新しい機能を導入するなどの大改訂はそれ以降の話題になる．

Ada の ISO 化は 1986 年に完了することを予定している．この場合も，現在の文法書のままで規格とするらしい．Ada 文法書の実際の文章の書き直しに関する案は，まったく世に出ていない．LMC で行われている議論は，将来規格が改訂される時に取り入れられることになると思われる．なお，言語の形式的記述法によって Ada を記述し，これを規格として採用するという案もあったが，時期尚早という理由で見送られた．

LMC から出される結論には，確認，拘束力のある解釈，拘束力のない解釈，などの種類がある．確認は，文法書を詳しく調べることによって導けるはずの結論の場合に使う言葉である．解釈は，文法書が矛盾していたり，規定が不足していたりするために，結論を導けない場合に使われる．解釈に拘束力があるのは，文法書が不十分でも合理的な解釈が一つしかありえない場合である．拘束力がないのは，解釈が二つ以上成立する場合である．この場合，LMC は最も適当と思われる解釈を与える．この解釈は，将来の文法書改訂の際に確定される性質のものであり，処理系の検定などにすぐには影響しない．

Ada が巨大で複雑な言語であるだけに，文法書の不備は数多く見つかっている．また，そのままでは処理系による実現が困難な点もいくつか発見されている．LMC では，これらの個所に統一解釈をつけたり，必要な規則を補ったりしている．実現困難な点については使用法に制限をつけるなどして言語を若干手直しすることもある．

2.3 言語仕様の変更点

一般に，LMCで行われている議論は，重箱の隅をつつくような細かなものばかりである．言語を大きく変えるような問題がないのはもちろん，一般の Ada プログラムに影響しそうなものもあまりない．しかし，処理系を作る立場から考えると，どのような細かな点でもはっきりさせておかなければならない．

ここでは，これらの議論のうち主なものを取り上げ紹介する．個々の問題の詳細に立ち入ることは避け，どのようなことが問題になっているかだけを紹介することに努めたい．

2.3.1 単純な誤り

文法書の単純な誤りまたは規定漏れがいくつか見つかっている．これらの解決策はほとんど自明である．

- ループパラメタの型は決められているが，部分型が決められていない．ループパラメタを case 文の式に使った時に問題が生じる．ループパラメタの値の範囲に相当する部分型を持つことに決められた．
- 値が制約を満たすかどうかを検査しなければならないという規定がいくつかの場所で落ちていた．位置による配列集成式，割当て子などである．
- 完全型が現れる前に密閉型を所属判定の中で使ってよいかどうかが正しく規定されていなかった．文法書は単純式の中に使うことを禁止しているが，所属判定は単純式でないからである．単純式の代わりに式と言えば，問題はない．
- out パラメタに対する実パラメタが型変換の形をとっている場合，返る時だけでなく入る時にも型変換を評価しなければならない．この規定が文法書から落ちていた．

2.3.2 静式に関する規定

静式は，コンパイル時に評価できるように，使える演算子などが細かく定められている．ところが，この規定に抜け穴があっていろいろな問題が起こっている．

- 所属判定を使って静的でない静式が作れる．これを避けるために，静式の中では短絡制御形や所属判定を使えないことにした．ただし，これは現在の文法書から読み取れるという説がある．
- 静式はスカラ型に対してしか定義されないが，スカラ型でないオペランドを持ちながら結果はスカラ型となる式をどう扱うかが問題になる．連結した上で比

較するなどである．このようなことを認めると処理系の作り方がいくらでも複雑になりかねないので，スカラ型でないオペランドを含む式は，一切静式と認めないことにした．

●汎用体仮型の式でも静式の条件を満たしていれば静式と認められる．この式の評価に問題はないが，配列集成式の添字に使われた場合に制約の検査ができなくなる恐れがある．これを避けるために，仮型の式は静式として認めないことにした．

2.3.3 言語設計者の意図の尊重

文法書の規定の中には，常識から判断して厳しすぎると感じられるものがいくつかある．しかし，これらは言語設計者がいろいろな問題に気を配って定めた結果であるので，勝手に変更することは好ましくない．ただし，将来言語の改訂が行われる時には検討の対象になるだろう．

●実数リテラルを固定小数点型の乗除算のオペランドとして使うことはできない．リテラルの型を決められないからである．固定小数点型の一つを選ばなければならないが，DURATION と無名の固定小数点型の二つの可能性が必ず存在し，一意に定まらない．一見不便であるが，処理系の都合を考えれば当然の結論である．

●実数のべき乗は，規則に忠実に繰り返し乗算しなければならない．乗算の組み合わせを変えたりして効率化を図ることは禁止される．精度が変わってしまうからである．

●離散範囲 1..10 は特例として INTEGER 型と解釈されるが，−1..10 は認められない．−1 はリテラルでなく式になるからである．−1 の場合は，型を指定しなければならない．この方針には反対が多いが，設計者は現状のままでよいとしている．

●定数につけた別名を静式の中に使えるかどうかが曖昧である．設計者は別名を許さないつもりであったが，反対意見が優勢である．この場合は，今までの例と違って設計者の意見が否定されることになりそうである．

2.3.4 タスクのスケジューリング

スケジューリングの際にタスクの優先順位が考慮される．しかし，個々の場面で優先順位を調べるべきかどうかがはっきりしないことがあった．LMC の結論は，常に優先順位に従ってスケジューリングをするということである．

●優先順位の低いタスクの実行中でも，高いタスクが実行可能になればそちらに制御が移る．プリエンプティブ・スケジューラが必要である．

●select 文で二つ以上の選択肢のうちのどれを受け付けるかを決める時，エントリを呼び出したタスクの優先順位を考慮しなければならないことになった．優先順位は，エントリの待ち行列からのタスクの取り出しには影響しないが，select 文の選択肢の選び方には影響を与えることになる．

2.3.5 入出力

文法書は入出力について細かく規定しているが，処理系作成者からさらに細かい部分に関する質問が寄せられた．これらを一つずつ解決する作業が行われている．OS との適合性やユーザ・インタフェースなど考慮すべきことが多いので，仕様をまとめるのは大変である．

●一時ファイルを CLOSE した時に消去してよいかどうかが議論された．再び OPEN された時に備えて消去しないということも考えられるが，処理系を簡単にするために消去してよいことになった．

●省略時のファイルはモードが変わらないことを原則としている．ところが，いったん CLOSE してから OPEN することによってモードを変えられることが発見された．例外が発生することにしなければならないが，いつ発生させるかで議論が分かれている．OPEN の時という説と次の GET や PUT の時という説がある．

●特殊文字を列挙型の手続きを使って出力するとどうなるかが不明である．目に見える形での表現が用意されていないからである．IMAGE 属性の値を出力する案が有力であるが，まだ決着していない．

●整数型の GET は構文に合わなくなった所で止まると規定されている．したがって，小数点などは読み込まない．ところが，拡張数字の範囲は，構文でなく意味上の制限であるので，いったん読み込んでから調べる．おかしな数字があっても（8進数で8や9など）そこで止まってはならない．常識的でない結論になるが，文法書に従う限りそう解釈しなければならない．この問題は，構文規則に対する違反か意味規則に対する違反かで結果が違ってくるために起こった．同様の問題は多重定義についても起こっている．

●GET_LINE は行終端子を読み飛ばし，必要ならばページ終端子も読み飛ばす．このため，端末から入力する場合，ページ終端子があるかどうかを調べるために余分の先読みを必要とする．これは具合が悪いの

で，端末の場合に限ってページ終端子を読まなくてもよいことにした．行終端子とページ終端子の関係に関する定義に問題があったのがそもそもの原因だと思われる．

2.3.6 解決困難な問題

以下の各問題は，文法書の不備であるが，解決が自明でないものばかりである．一応解決したものも，苦しげな方法に頼っている．さらに研究する必要があるだろう．

●列挙リテラルとパラメタなし関数の両方を引き継ぐ派生型の定義が可能である．この二つはホモグラフであるので，両方を見せるわけにはいかない．ところが，どちらが隠されることになるか，文法書からは読み取れない．この問題は，列挙リテラルを既定の操作の一種と見なし，関数によって隠されるとすることで解決した．

●整数型定義による整数型の宣言は，型宣言と部分型宣言の二つの列と等価である．このうち，後者には型変換が含まれている．ところが，型変換を含む式は静式でない．したがって，文法書を忠実に読むと，静的な整数型は存在しないことになる．これでは困るので，この場合に限って型変換があっても静式として許すことになった．

●汎用体仮密閉型が汎用体の中で算体宣言などに使われている場合，実型に判別子がついていてはならない．これは，判別子に省略時の式があっても同じである．この規則は，他の規則と比べて厳しすぎ，汎用体の有用性を大きく損ねることになりかねない．そこで，省略時の式があればよいことにしようと考えられている．しかし，他の問題ともからんでいる部分があるので，解決は難しそうである．

2.3.7 実現困難な点の改善

以下の各問題は，主として処理系による実現を容易にするための変更である．

●普遍型の式を評価する時に，処理系の提供する最も広範囲な型を越える場合は例外を発生させてよいことになっている．これは，多倍長演算などを使わずに単純な機構で計算できるようにするためである．ところが，このためには，計算途中の値すべてについて例外を発生する規定がなければならない．文法書は式の値が範囲を越える場合についてしか規定していないので不十分である．オペランドが範囲を越える場合にも例外が発生するよう変更することになった．

●タスクの共有変数の非同期のアクセスをエラーとする条項は，スカラ変数に対してだけ決められている．配列やレコードでも，要素単位のアクセスだけを問題にすればよいという考え方である．ところが，これでは1語に多くの要素を詰めることが難しくなる．この問題にはうまい解決策が見つかっていない．適当なプラグマを用意して対処することになるだろう．

●密閉型や不完全型に判別子がついている時，この型に判別制約をつけて部分型を作ることができる．ところが完全型が宣言されるまで，判別子の値が内部での使われ方に矛盾しないかどうかの検査ができない．文法書には，この検査をいつ行ったらよいかが示されていない．解決策として，完全型の宣言まで検査に必要な情報をためておいて，その時点で検査することが考えられた．この案は，同一単位内であれば簡単だが，パッケージの密閉部で宣言された不完全型の中身をパッケージ本体で与えている場合には，処理が難しくなる．そこで，この場合は判別制約を禁止することにした．

3. 処理系の動向

1979年に言語仕様の大方が定まると，AJPO は処理系の検定を行うための準備作業にかかった．一連のテストプログラムと，それらでテストした結果を評価するための支援システムを作ることにしたのである．以上のものをまとめて，ACVC (Ada Compiler Validation Capability) と呼ぶ．実際の作業は SofTech 社に委託された．

DoD としては，この ACVC を使った検定に合格したものしか，Ada 処理系の名前の使用を許可しない方針をとった．ACVC のテストプログラムには，単に誤りなく動くものばかりでなく，コンパイル時または実行時に誤りを検出し，必要なエラーメッセージを出すことを要求するものも含まれている．

DoD は，ACVC と対をなす活動として，Implementer's Guide なる資料を作り，配布してもいる．これは，処理系作成者を手助けする目的で，各言語仕様の実現法，文法書の解釈などを解説したものである．

言語仕様が見直され，規格化されようとする時期になってもなかなか検定に合格する処理系は出てこなかった．というよりも，実際に完成したコンパイラそのものが存在していなかった．早くからインタプリタを完成させていたのは New York 大学であった．このインタプリタは，言語仕様の評価作業にも，また初期の言語教育にも，ネットワークを通じて広く利用され

表-1 検定済み Ada コンパイラ (1985年1月27日現在)

開発者	機械 (O S)	検定年月 (ACVC 版)
Rolm/Data General	MV 4000, MV 6000, MV 8000, MV 10000 Rolm MSE-800 (AOS/VS)	84- 5 (1.4)
TeleSoft	O-Bus/M 68000, LabTEK/M 68000 (ROS) Callan UniStar 300 (Unix V 5, V 4. 1a) Sun 120/M 68010 (Unix 4.2 BSD)	84- 5 (1.3) 84- 8 (1.3) 84- 8 (1.3)
New York 大学	VAX 11/780 (VMS 3.5)	84- 8 (1.4)
DEC	VAX 11, MicroVAX (VMS 4.0)	84- 9 (1.4)
Dansk Datamatik Center	VAX 11/750 (VMS 3.0)	84-10 (1.4)
Karlsruhe 大学	VAX 11/750 (VMS 3.0) Siemens 7.51 (BS 2000 7.)	84-11 (1.4)
ALSYS	ホ ス ト: VAX 11, MicroVAX (VMS 4.0) ターゲット: ALTOS, ACS 68000 (ALTOS OS 1)	84-12 (1.5)
Verdix	VAX 11/750 (Unix 4.2 BSD)	84-12 (1.5)
Honeywell	DPS 6 (GCOS 6 MOD 400)	84-12 (1.4)
SofTech	VAX 11/780 (VMS 3.6)	84-12 (1.4)

ていた．検定が1983年に至ったのは，ACVC 側の準備の問題もあったものと思われる．

New York 大学に続いて，Rolm 社 (1983 年6月) と Western Digital 社 (1983 年8月) とが処理系検定に合格した．この年の2月に言語仕様の方は MIL/ANSI 規格となった．この時点の検定が完全にこの新しい規格にのっとったものであったかどうかは定かでない．

新しい規格に合わせて ACVC の手直しを行う作業も同時に進められていた．手直しはまた，いくつかの検定作業そのものの経験から見つかった不具合の修正にも及んだものと思われる．ACVC 1.2 版は 1983 年7月に世に出ている．

その後も小きざみに ACVC の改訂は進められ，以後，6カ月ごとに版を改めるとの方式が定着した．新版に変えると同時に，6カ月先に採用する予定の版も公開する方針をとっている．1985年6月からは 1.6 版が採用されている．

検定済みの処理系についても，その検定の有効期限は1年とし，続けて Ada 処理系を標榜するには再検定を受けることを義務づけている．New York 大学と Rolm 社とは再検定に合格したが，Western Digital 社 (現 GenSoft 社) は再検定を申し出なかったため，有効期限切れとなったままになっている．

表-1 に 1985 年1月 27 日現在で検定に合格している処理系の一覧を掲げておいた．1984 年の後半になって，多くの処理系が合格していることが見て取れよう．

表にある処理系のうち，直接に DoD からの契約によっているのは SofTech 社のものだけである．SofTech 社は陸軍からの発注を受けているのに対し，もう一社 Intermetrics 社が空軍からの発注を受けて処理系を開発中である．一応 1985 年中には検定に合格する予定との記事もあるが，定かではない．

SofTech 社の処理系は，検定合格に先立って，NATO および SEATO 傘下の国家機関および米国国内の企業で，他機種への移植を試みるところへは公開するとの方針がとられた．諸国がどう反応したかは明らかでないが，米国国内からは 62 社の申込みがあり，審査の上 32 社との契約が行われた．今後，これを出発点とした処理系が検定に合格してくるものと思われる．

4. 支援環境の動向

もともと，Ada 計画そのものが開始されたのは，DoD 関連の組込み型ソフトウェア (特に保守) にかかる費用の低減を目的としていた．共通の高級言語の存在は，こうした目的にとっての一つの前提条件に過ぎない．その上に，開発・保守にあたるための支援環境，さらには，それらの作業を支える方法論に至るまで，いわゆるソフトウェア工学の最新の成果を取り込むことが必要となる．

て現在進行中のものは，共通支援環境のあり方（方法論を含めて）を探るものが主体となっている．

英国では，国防省が PSE(Project Support Environment) という名前で支援環境の整備にあたっている．開発済みのソフトウェア設計・開発法 (MASCOT) を順次改訂し，Ada言語にも合わせていく計画だという．また，EQC というセンタを設けて，Ada 処理系の検定・評価にあたらせる予定である．ICL の PERQ 2 上にマイクロプログラムされた仮想機械 FLEX 上に開発中の Ada コンパイラも近々完成の予定であるという．

英国ではまた，高品位・高性能ソフトウェアの生産性向上を目指した ALVEY 計画が実施されている．この中では ECLIPSE という名の支援環境（先出の MASCOT などの方法論を支援する）を，FOUNDATION と呼ぶ基本支援環境上に作る作業が進行中である．この計画の中で，Ada は支援される対象言語の一つとして位置づけられている．

5. Ada 関連の諸団体

米国の学会である ACM には，1981 年に SIGPLAN (Special Interest Group for Programming LANguages) 配下に AdaTEC (Technical Committee on Ada) が設けられ，Ada LETTERS という雑誌を隔月刊行するようになった．AdaTEC は 1984 年に SIGPLAN から独立し，SIGAda として活動を続けている．各種の研究会合を開催するほかに，委員会を設けて独自の作業も行っている．委員会には，歴史・政策・教育・処理系実現・支援環境・設計方法論などがある．

米国内には，Ada のユーザ会にあたる AdaJUG (Ada-JOVIAL Users Group) という団体も作られている．DoD 側としても，これらを含めて，Ada に関する諸情報の交流のため Ada IC (Information Clearinghouse) を設けて，便宜を図っている．

欧州では，広く Ada Europe という組織が作られていて，研究会合を開催するなどの活動を行っている．Ada Europe は CEC の支援を受けた組織である．英国には Ada UK という組織が作られていて，季刊の雑誌 Ada UK News を刊行するなどの活動を行っている．

欧州での Ada 関連の活動は比較的活発で，Cambridge University Press からは Ada Companion Series と銘打った図書が 10 冊近く出版されているほか，ADAKOM とよぶネットワークも用意されている．

参 考 文 献

1) U.S. Department of Defense: Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815 A (1983)；邦訳-情報処理振興事業協会（編）：最新 Ada 基準文法書，bit 別冊，共立出版 (1984).
2) U. S. Department of Defense: Requirements for Ada Programming Environments "Stoneman", p. 44 (1980).
3) Military Standard Common APSE Interface Set (CAIS), proposed MIL-STD-CAIS, p. 317 (1985).
4) U. S. Department of Defense: Ada Methodologies: Concepts and Requirements (1982), in SIGSOFT Software Engineering Notes, Vol. 8, No. 1, pp. 33-50 (1983).
5) U. S. Department of Defense: Software Technology for Adaptable, Reliable Systems (STARS) Programs Strategy (1983), in SIGSOFT Software Engineering Notes, Vol. 8, No. 2, pp. 56-108 (1983).
6) Freeman, P., Wasserman, A. I. and Houghton, R. C., Jr.: Comparing Software Development Methodologies for Ada: A Study Plan, SIGSOFT Software Engineering Notes, Vol. 9, No. 4, pp. 22-55 (1984).

(昭和 60 年 11 月 13 日受付)