ソフトウェア工学　３９－４
（１９８４．１１．２８）

Ａｄａ処理系の実現方式

細谷慷一　　田中　清　　藤丸政人　　山口和幸
（日本電信電話公社　横須賀電気通信研究所）

１．はじめに

データ通信システムなどの大規模化・高度化に伴い、ソフトウェアの適用分野および規模が急激に増大している。この結果、各種のプロセッサおよび言語が用いられ、①ソフトウェア流通が困難、②開発環境構築への重複投資、③保守が困難、④要員訓練、要員流動が困難であるなどの問題を引起こしている。

これらの問題を解決するためには、①単一の標準言語により記述言語を統一し、②その言語を中心に開発支援系を構築することが最も現実的かつ有効である。また、この統一言語としては以下の理由により、ＩＳＯで標準化が進められているＡｄａ* が最適である。

（ｉ）ソフトウェア工学の最新の成果を取入れており、プログラム作成の生産性、信頼性が高い。

例えば、言語仕様の随所に、型の概念、データ抽象化等の最新のソフトウェア工学の成果が取り入れられており、信頼性の高いプログラムを高い生産性で作成できる。

（ii）システムプログラムから、科学技術計算や事務処理等の応用プログラムまで記述でき、さらに、パッケージ機能により応用分野向き各種機能が容易に定義できるので、種々の分野への適用が容易となっている。

（iii）標準化の徹底により、流通性の高いプログラムが作成できる。

例えば、コンパイラの検定システムが用意されており、方言の派生が規制され、異なるコンパイラ間でもプログラムの流通が可能となる。　また、機種に依存する部分は、他の部分と分離して記述しなければならないという規制が設けられている。　これにより、他機種への移植は主としてその部分の変更のみで実現でき、流通性の高いプログラムが作成できる。

（iv）今後のプログラミング言語の主流になると予想される。

（Ｖ）要求仕様にもとづいてＡｄａ総合プログラミング環境（ＡＰＳＥ）[4] として支援系も含めた環境づくりが進められている。

これらのことからＡｄａ処理系の開発を進め、標準言語仕様から並列処理記述機能などの一部の機能を除いたサブセット版言語仕様によるコンパイラと支援システムからなる処理系を実現した。

評価実験では、これまでシステム記述言語として用いてきたＳＹＳＬ[3] に比べ、目的プログラムのダイナミック・ステップ数が約０．９２に減少するほか、生産性が約２０％以上、信頼性が約４０％向上できるなどの効果を確認した。

本報告では、実現したＡｄａ処理系の言語機能、支援機能および評価結果について述べる。

２．言語機能の選定

サブセット言語機能の選定は以下の基準により行った。

（１）ＯＳ、通信制御プログラム、コンパイラ、アプリケーションプログラム等の広範囲な分野を記述できる。

（２）効率の良い目的プログラムを生成できる。

（３）ＯＳ記述、既存ソフトウェアとのプログラム結合を可能とするため、標準言語仕様の規定内で以下のシステム記述機能を充実する。

①プログラムデータのメモリ域指定

---

* Ａｄａは、米国政府、ＡＪＰＯの登録商標である

②サブプログラム* 呼出しなどにおけるレジスタコンベンションの指定
③パラメータパッシングにおけるレジスタ結合の指定

3．コンパイラの構成と実現法

3．1コンパイラの構成

実現したコンパイラのフェーズ構成を図１に示す。主な特徴及び処理概要を以下に述べる。

3．1．1フェーズ構成の特徴

（１）２レベルの中間言語

コンパイラを含めた各種ツールの外部流通をねらい、コンパイラと支援ツールの切口として、米国を中心として標準化が進められている木構造形式の高水準中間言語ＤＩＡＮＡ（中間言語Ⅰ）を採用した。また、他機種へのリターゲット化部分を少なくするため中間言語Ⅰの下に物理レジスタなどを意識しない機種独立で機械語生成が容易な四つ組形式の低水準中間言語（中間言語Ⅱ）を設定した。

（２）ポータブルなフェーズ分割

２つの水準の中間言語を基に、コンパイラのフェーズを、①ソースプログラムを構文・意味解析して中間言語Ⅰを出力するフロントエンドⅠ、②中間言語Ⅰを入力して中間言語Ⅰ上での最適化、中間言語変換を行うフロントエンドⅡ、③データの物理アドレスを割付け、コード生成を行うバックエンド、の３つに物理的に分割し、機種独立なフロントエンドⅠ・Ⅱについてはそのまま他機種へ移植可能とするとともにバックエンドの切替えによるリターゲット化を可能とした。

3．1．2処理概要

（１）フロントエンドⅠ

①構文解析

ソースプログラムに対し構文チェックを行い中間言語Ⅰの原型を作成する。構文解析では、構文テーブルを用いたテーブルドリブン方式を採用し処理の簡素化を図っている。　また意味解析でのネームサーチを高速化するためハッシュテーブルなどを作成している。

②意味解析

構文チェックが完了した中間言語Ⅰの原型をもとに意味解析を行い完全な中間言語Ⅰを作成する。このとき、文脈節**に指定されたパッケージなどの文脈単位の中間言語Ⅰの情報を、それが格納されているプログラムデータベース（後述）から読込み、文・式などの一意性を決定する。対応する文脈単位が未コンパイルの場合には、その旨のメッセージ情報を作成する。

（２）フロントエンドⅡ

定数の翻訳時計算など機種に依存しないソースレベルの最適化処理についてはリターゲットに際して変更が不要なため、フロントエンドⅡに位置付けた。また、このフェーズでは、木構造の中間言語Ⅰを、バックエンドでの機械語生成が容易でかつアクセス速度が速い線形な四つ組形式の中間言語Ⅱへ変換する。

（３）バックエンド

①文対応の機械語生成

中間言語Ⅱ対応に用意された機械語のスケルトンを展開し、実レジスタを割当てると共にメモリへのデータ接近形式を決定する。

このとき、命令部の大きさを見積り、命令部接近用レジスタの有効割当てを図るなど、レジスタの効率的管理を行う。

---

*　サブプログラムとは、手続きまたは関数を表す。

**文脈節は、そのサブプログラム単位の中で必要とする型やデータの情報がどのパッケージに含まれているかなどを指定するために用いられる。

② サブプログラムなどのライブラリ単位に対しては、レジスタの退避・回復などを行うプロローグ・エピローグ処理の機械語を付加し、目的プログラムを完成させる。

3．2主な機能の処理方式

（1）パッケージと分離コンパイル

パッケージは共通データなど複数のプログラムから参照される情報を一ヶ所にまとめ定義するのに便利であることから、他のプログラムから頻繁に参照される。また、これらの情報は、パッケージから分離して記述されたサブプログラム（分離コンパイル単位）からも参照される。

このためパッケージの参照時に何らかの方法でパッケージの仕様情報を取得する必要がある。この方法として、①必要となる毎にパッケージソースプログラムを逐次解析して情報取得を行う方法と、②予め、パッケージをコンパイルした時点で、これらの情報を中間言語としてプログラムデータベースに蓄えておき、パッケージ参照時にはプログラムデータベースから必要な情報を読込む方法がある。情報取得の高速化が図れること、支援系の効率的実現が図れること（4．2参照）からパッケージ情報をプログラムデータベースに蓄える方式を採用した（図2）。 さらに、プログラムデータベース内の個々の情報に対してはハッシュテーブルを用いた情報アクセスの高速化を図っている。

（2）オーバローディング[5] [6]

オーバローディングは、同じ名前を種々の用途（例えば整数加算としての既定義の”＋”記号を配列の加算にも用いるなど）に用いることを可能とするものであり、オーバローディングの解析には名前の一意性の決定が必要となる。 中間言語Ⅰの木を根から葉へ上向き（ボトムアップ）に、次に下向き（トップダウン）に評価する2パス定理を用いた。具体的解決の手順を以下に示す。

・手順1（ボトムアップ解析）：

対象サブプログラム（手続き、関数、演算）のパラメータをチェックして最終的に選ばれるべきサブプログラムの候補を選ぶ。

①可視性の規則により、対応するサブプログラム個数の観点で許容できるサブプログラムの集合Tを得る。

・手順2（トップダウン解析）：

関数、演算について、最終的に選ばれるべきサブプログラムを一つに絞る。

①集合Tの中から、サブプログラムの外側の文脈から要求される型（例えば代入文の左辺の型）とサブプログラムの結果の型（関数値の型、演算結果の型）を充たすサブプログラムを得る。

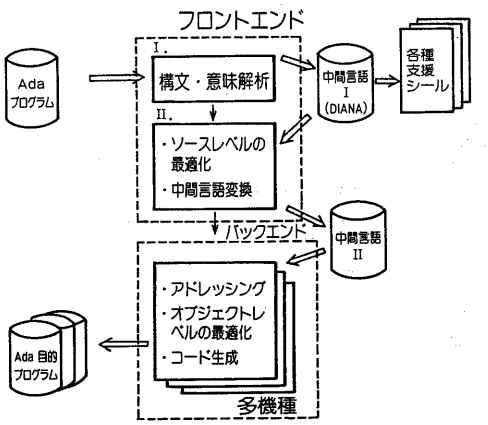

図1 コンパイラのフェーズ構成

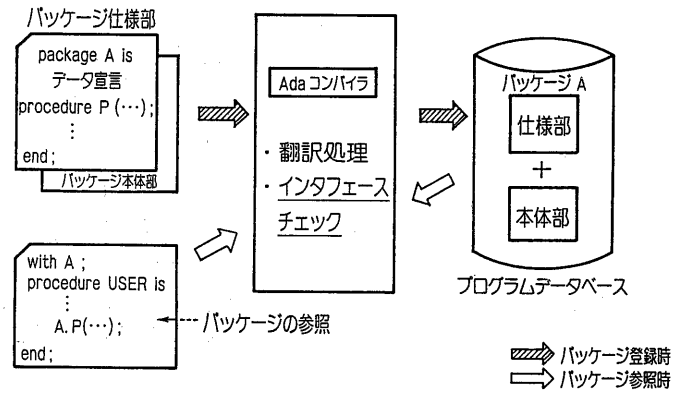

図2 パッケージ機能の処理

３．３最適化技法

システムプログラムからアプリケーションプログラムまでの広範囲なソフトウェアの記述を可能とするには、コンパイラが高性能な目的プログラムを生成することが必須である。このため以下に示す最適化処理を実現している。

（１）フロー解析による一般的な最適化

高級言語化による冗長なコーディングを検出して効率の良い目的プログラムを生成するため、コントロールフロー、データフローなどのフロー解析技術を利用し、文にまたがる共通式やループ内での不変式などの検出を行うとともにプログラム全体にわたる広域的なレジスタ割付けを行っている。

レジスタ割付けにおいては、データの値が代入されてからその値が最後に参照されるまでの区間、そのデータの値をレジスタにひきあげておきロード命令やストア命令を削除するなどの工夫をしている。

（２）外部手続きに関するレジスタ割付けの最適化

性能条件が厳しいＯＳ中核部を調査したところ、ＯＳプログラムの高級言語化のネックとなっている性能上の問題点として以下の点が明らかとなった。

①データ領域確保・解放のオーバヘッド
②アーギュメント渡しのオーバヘッド
③レジスタ退避・回復のオーバヘッド

コンパイラではパッケージ機能の特性を活かすことによりプログラム間にまたがる広域的なレジスタの一括管理を中心とした以下の最適化処理を実現した。

（ⅰ）プロローグ・エピローグ処理の効率化

パッケージ内の手続き間の呼び出しか否かを判定し、呼出し元が同一パッケージ内手続きであればプロローグ・エピローグ処理を行わないような目的プログラムを生成する。プロローグ・エピローグ処理の効率化の概念を図３に示す。

（ⅱ）データ領域をとらない目的プログラムの作成

パッケージ内で宣言されたデータの個数と使用できるレジスタの個数を見積り、可能であればすべてのデータ（または、手続きのデータのみ）をレジスタ上に割付ける。

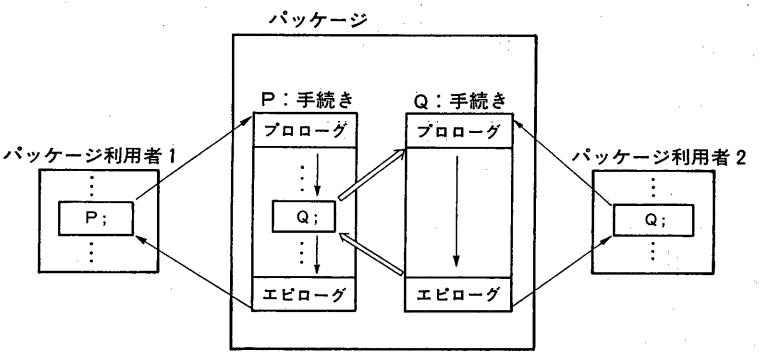

図３　プロローグ・エピローグ処理の効率化

(iii) 外部手続き間のレジスター括管理

パッケージで宣言されたデータとそのパッケージに含まれる手続きのデータすべてを対象とし、以下の広域的なレジスタの固定を行う。

①パッケージの仕様部で宣言されたデータについては、パッケージ内で共通的に参照されるので優先的なレジスタ固定を行う。

②手続きのデータについては@呼出し関係のある手続き間のデータ対して、手続きにまたがる広域的なレジスタ固定を行う。⑥その他のデータに対して、手続きに閉じたレジスタ固定を行う。

4．支援システムの機能と構成

ソフトウェアの信頼性、生産性の向上を図るため、設計・製造、テスト、デバッグ、保守のソフトウェアライフサイクル全体を支援する基本的な支援ツールを実現する（図4）とともにプログラムデータベースによる開発情報の一元管理化を図った。

4．1支援機能

実現した主な機能は、以下のとうりである。

（1）論理チェツカ[8]

プログラム・バグを早期に検出するため、データの値域を解析することにより、従来実行時でなければ検出できなかった、無限ループ、実行不能文などの検出を行う。

（2）シンボリックデバッガ

高級言語レベルでのデバッグを会話的に行うため、プログラム・フロートレース、割込み原因のシンボリック解析などのデバッグ機能を提供する。

（3）テスト網羅性測定ツール

内部仕様に基づくテストの完了度を評価するため、プログラムの内部構造を解析することによりプログラムのパスおよびエッジ* を抽出し、これをテスト網羅性の測定基準としたテストの進捗管理機能を提供する。

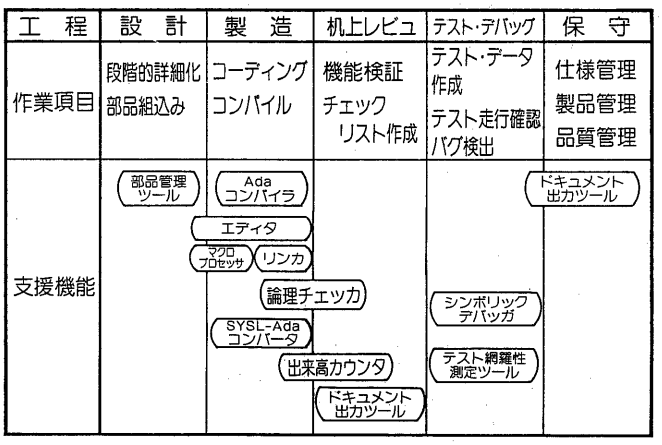

図4　DIPS　Ada支援環境

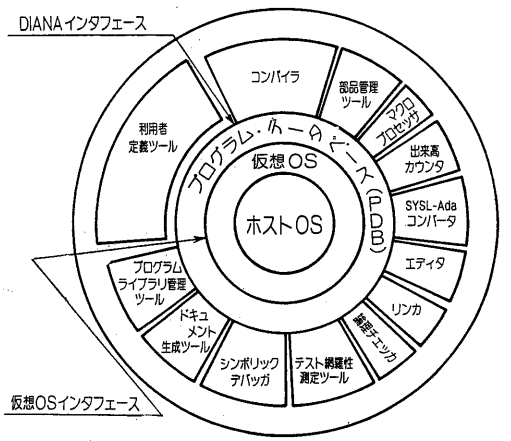

図5　DIPS　Adaシステム構成

---

* 途中に分岐のない実行文列をノードに、これらの実行文列間の制御の流れをアークに対応付けた有向グラフ上において、入口から出口に至る経路をパス、分岐箇所に対応するアークをエッジと定義する。

（４）ドキュメント出力ツール[9]

設計ドキュメントとソースプログラムとのズレを防ぐため、ソースプログラムからモジュール関連図、ＨＣＰチャート[10]などの各種保守ドキュメントを出力する。

（５）ＳＹＳＬ－Ａｄａコンバータ

既存のソフトウェア資産の流用および言語統一によるメンテナンスの一本化を図るため、ＳＹＳＬソースプログラムからＡｄａソースプログラムへの言語変換を行う。

４．２支援システムの構成

支援システムの構成を図５に示す。構成上の特徴を以下に述べる。

（１）ＯＳインタフェースの仮想化

従来の言語処理プログラムや各種支援ツール等は、ハードウェア情報やＯＳマクロなど、特定機種に依存する情報が散在する構成となっており、プログラムの移植作業に多大な労力を要している。ＤＩＰＳ Ａｄａ処理系では、他システムへの移植を容易とするため、ハードウェア、ＯＳ等に密接に関連するメモリ管理、ファイル管理などの基本的なＯＳマクロをプログラムから分離し、これらをＡｄａパッケージ機能を用いて作成することによりＯＳインタフェースを仮想化した。 ＯＳインタフェースの仮想化の概念を図６に示す。

具体的には、①ファイル入出力機能、②メモリ管理機能、③プログラム管理機能、④および制御表などの機種依存情報の仮想化機能などの仮想ＯＳインタフェース・ルーチンを実現した。 Ａｄａ標準入出力機能を実現する実行時ルーチンは、ファイル管理パッケージ内の順編成ファイル操作、直接編成ファイル操作を用いて構築した。

（２）プログラムデータベースの一元管理化

従来の支援ツールでは、ソースプログラムから処理に必要な制御フロー等の情報を抽出しており、個別に構文解析等の処理を行っている。 この構文解析等は、各支援ツールに共通的な処理であることから、これらを支援システムとして共通化することにより、ツール開発規模が削減できる。 このため、Ａｄａコンパイラが解析・生成した中間言語Ｉを各支援ツールで共通的に利用するとともにプログラムデータベースとして一元的に管理することにより、一貫した開発支援および効率的なシステム構成を可能とした。

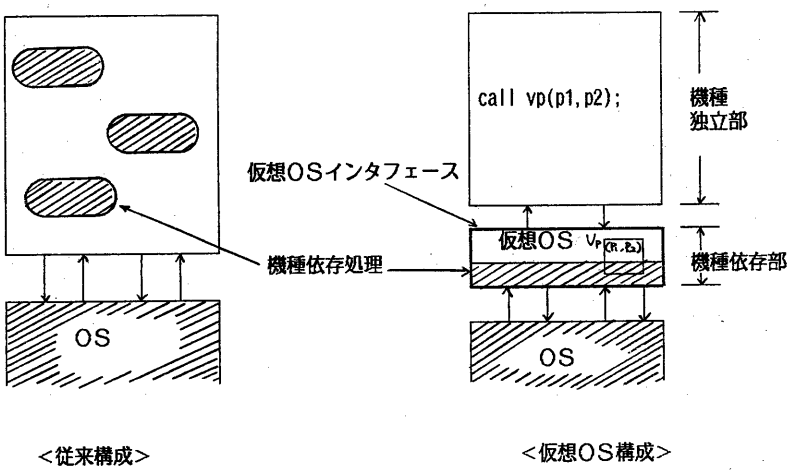

図６ ＯＳインタフェース仮想化の概念

5．評価

（１）性能

ＳＹＳＬで従来から適用してきた最適化処理に加え手続き間にわたるレジスタ管理、プロローグ、エピローグ処理の効率化など、Ａｄａ特有の最適化処理の効果を処理プログラムレベルのサンプルプログラムの記述実験で評価した。 その結果、ＡｄａはＳＹＳＬに比して、ダイナミックステップ数が０．９２倍、スタティックステップ数が１．０１倍、メモリ占有量が０．８７倍であることを確認した。

（２）信頼性・生産性

信頼性については、Ａｄａコンパイラ開発時（ＳＹＳＬ記述）のバグの机上分析により、ＤＢ工程で検出されたバグのうち、約４０％のバグはＡｄａコンパイラおよび支援システムにより、プログラム実行テスト以前に検出可能となることを確認した。

生産性については、今回、Ａｄａで記述したＡｄａ処理系の一部の支援ツールと従来のＳＹＳＬ記述の同種ツールの開発データを机上比較した。その結果、従来手法に基づく言語処理プログラムの開発工数に比べて約２４％削減できることを確認した。 この効果は、言語機能による効果に加えて、各種支援機能の効果により、テーブル設計書、インタフェース仕様書等の設計ドキュメント作成工数の削減、データ・コード設計工数の削減、テストデータ作成工数の削減および早期バグ検出に伴うデバグ工数の削減などによるものである。

6．おわりに

システムプログラムをはじめとする広範なソフトウエア記述に適用することをねらいとしてサブセット版コンパイラおよび支援システムからなるＡｄａ処理系を実現し基本ソフトウェアを中心に現在適用を進めている。

今後は、さらに、アプリケーションプログラムの記述への適用を図るとともに、コンパイラのフルセット化を行う。 さらに、マイクロプロセッサ用の処理系を開発し、移動通信装置、宅内機器等のソフトウェア作成に適用していく予定である。

参考文献

(1) American National Standard Institute , Inc.Reference Manual For The A Programming Language,ANSI/MIL-STD -1815A-1983.

(2) Tartan Laboratories Incorporated ,Draft Revised DIANA Reference Manual, Revision 2.1 ,1982 October 24 .

(3) 細谷他：" システム製造用言語ＳＹＳＬ" , 通研実報, 24,No. 1, PP.95,(昭50).

(4) U.S.Depatment of Defense:" Requirements for Ada Programming Support Environments, “stoneman” " , Febuary 1980.

(5)Perech,G.,Winsteratein,G.et al.:"Overlooading in ADA",Bericht,23,1979.

(6)Bauer,F.L,et al:"Computer Construction" ,2,1976,Spriger-verlag.

(7) 田中, 藤丸, 山口他:"中間言語に基づくＡｄａドキュメント自動生成法" , 情処学会, 59年前期全国大会予稿集 ,PP.541-542 ,(昭59) .

(8) 田中, 藤丸, 山口他:"Ａｄａコンパイラと論理検証ツールの最適構成について" , 信学会, 58年度部門全国大会予稿集 ,PP.574,(昭58).

(9) 田中, 藤丸, 山口他:"Ａｄａ言語システムの共通中間言語とその効率的実現法",情処学会, 57年後期全国大会予稿集 ,PP.359-360,(昭57).

(10)花田, 他:"コンパクト・チャートを用いたプログラム設計法",情処学論, 22,1,PP.44-50(昭56-01)