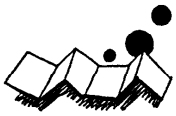
Vol. 27 No. 3　　情　報　処　理　　Mar. 1986

解　説

Ada の適用例（その 2）



通信システムの Ada による統合的開発事例†

荒木 啓二郎†† 牛島 和夫††

1. はじめに

プログラミング言語 Ada は，組み込み型の計算機システム用に開発されたにもかかわらず汎用の高水準言語であり，1970 年代のプログラミング言語ならびにソフトウェア工学の研究成果を反映しているといわれている．ちなみに Ada 基準文法書[20] から Ada の主要な目標ないし特徴を表すキーワードを拾ってみると（意味・内容に重なりがあるものもあるけれども）以下のようなものが挙げられよう．

- 大規模実時間システムのプログラミング
- 読みやすさ
- プログラムの信頼性
- 保守性
- データ型
- モジュール化
- タスク処理
- 情報隠蔽
- 分割コンパイル
- ライブラリ管理
- ソフトウェアの部品化
- プログラムの移植
- 内部表現の指定
- システム依存機能へのアクセス
- 例外処理
- 汎用体

われわれは，これらの目標ないし特徴を持つ Ada が，実際のソフトウェアシステムの開発においていかなる効果をもたらすかに関心を抱いてきた．そこで，Adaを用いることを前提としたシステム開発法に関する研究と，実際の利用経験に基づく Ada の評価とを行うことにした．われわれの関心は主として，システム開発過程と Ada との係わりにあり，例として開発するシステム自体は次のような条件を満たすものであれば，ある程度どういうものでも構わなかった．すなわち，第一に，われわれは Ada の重要な機能であるタスクに最も興味を持っていたために，分散型のシステムを対象として，その開発においてタスクが果たす役割を調べられること．第二に，ある程度実用規模に近いシステムであること．幸いにして，通信システムの専門家からのご教示をいただく機会を得たので，われわれは本稿で述べるように通信システムを例として取り上げて，Ada を用いて実際に開発してみた[1),2),3),17)]．

本稿では，通信システム開発の過程を述べた後で，われわれの経験に基づいてソフトウェアシステム開発の各段階における Ada の適用性を評価し解説する．2 章では，分散型システムの例として通信システムを対象とした場合のソフトウェア開発過程と Ada が果たす役割とについて述べて，われわれが今回採用したシステム開発方法を紹介する．3 章では，通信システム開発の事例研究として，対象とした通信システムの概要，Ada によるプログラム作成，通信システムの実装について述べる．4 章では，われわれの経験に基づいて Ada の評価を試みる．最後に 5 章で，まとめを述べる．

2. ソフトウェア開発ライフサイクルと Ada

ソフトウェア開発のライフサイクルは，通常，

- 要求定義
- 仕様記述
- 設計
- プログラミング
- テスト
- 運用，保守

という段階に分けられる．

われわれは，ソフトウェア開発ライフサイクルにおけるほとんどすべての段階で Ada を効果的に用いることができるのではないかとの予想の下に，通信シス

† Integrated Development of Communications Systems Using Ada—A Case Study— by Keijiro ARAKI and Kazuo USHIJIMA (Department of Computer Science and Communication Engineering, Kyushu University).

†† 九州大学工学部情報工学科

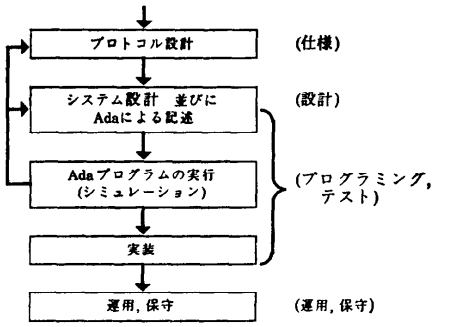


図-1 Ada を用いた通信ソフトウェアの開発過程

テムを例にとり実際にその開発を行ってこの予想を確かめることにした．われわれが今回採用した開発過程を図-1に示す．右側の括弧内には，上記のソフトウェア開発ライフサイクルとの対応を示す．

通信システムが通信を行う際のデータ形式や手順を定めた通信規約（プロトコル）を，作成する通信システムに対する要求仕様と見なすことにする．今回われわれは，すでに状態遷移表の形で与えられたプロトコルを用いた．

まず最初に，与えられたプロトコルを Ada で記述する．このように，プロトコルの記述にプログラミング言語を用いる方法は，従来から考えられているし[16]，最近では FDT (Formal Description Technique) と呼ばれるプロトコルの形式的記述技法として研究されている[15]．

パッケージ，並列タスク，仕様と本体の分離，ランデブによるタスク間通信などの Ada の主要な機能は，システム設計の際に便利な道具立てを提供してくれる．ことに，通信システムのような分散型システムの設計においては，タスクの使用が大変有用であろう．Ada を用いた設計については，他の文献でも詳しく議論されているのでそちらの方も参照されるとよかろう（たとえば[6),7)]）．

Ada で記述されたプロトコルは，上述のように通信システムに対する仕様と見なすことができる．一方，それはもちろんプログラムでもあるわけで，計算機上で実行可能である．すなわち，Ada で記述したプロトコルは，実行可能な仕様と見なすことができる．実行可能な仕様を計算機上で実行させてみることによって，システムへの理解が深まり，また，仕様定義の妥当性の確認もできる．これは，システム開発におけるいわゆるプロトタイピングということができる．分散型システムの開発においては，特にこのプロトタイピングが重要な役割を果たす[10]．Ada は，タスクを含む種々の機能を備えた高水準言語であるために，ここで述べたようにプロトコルを記述する仕様言語としても用いることができる．

通信プロトコルを記述した Ada プログラムというのは，通信システム全体を記述した単一のプログラムであり，この時点では一台の計算機上で実行される．すなわち，通信システムのシミュレーションを行っているということでもある．この段階において，通信システム全体ならびに各部分についての動作確認を行うことができる．この結果，プログラム中の誤りを発見したり，システム設計に対してフィードバックを与えたりすることもあろう．

上述のプロトタイプないしはシミュレーションを，最終的には，目的とする通信システムの実現へと展開せねばならない．目的とする通信システムは，複数台の計算機から構成される分散型システムである．現在のところ，単一の Ada プログラムが分散型システム中の複数の計算機上で分散されて実行されるようなシステムは，研究の段階であり，実際に利用することはできない．したがって，通信システムの完成にあたっては，システム全体を記述した単一の Ada プログラムを分割して，通信システムを構成する各計算機上でそれぞれ独立した Ada プログラムが実行され，それらが通信媒体を介して通信するようにしなければならない．それを行うためには，システムに依存したプログラムを書かねばならぬことが多いと予想される．Ada では，ハードウェアや処理系に依存した事柄も陽に記述できるので，このようなシステム依存事項を通信媒体に直接アクセスする部分に封じ込めておけば，プロトタイプやシミュレーションで用いた Ada プログラムの大部分がそのまま最終システム内でも生き残る．これによって，プロトタイピングやシミュレーションから，最終システムの完成まで一貫して Ada を使用することができる．

上述の開発法は，ホスト／ターゲット法[19] と呼ばれるものの範疇に属するものである．その開発過程を図-2 のようにまとめてみた．ホスト計算機上の Ada 処理系の下で，対象となる通信システムの設計，記述，シミュレーションによるテストを行った後に，実システム上への実装を行って，最終システムの完成に到るという次第である．前述のように，ホスト計算機上では通信システム全体を記述した単一の Ada プログラムが実行される．一方，ターゲットは分散型のシ

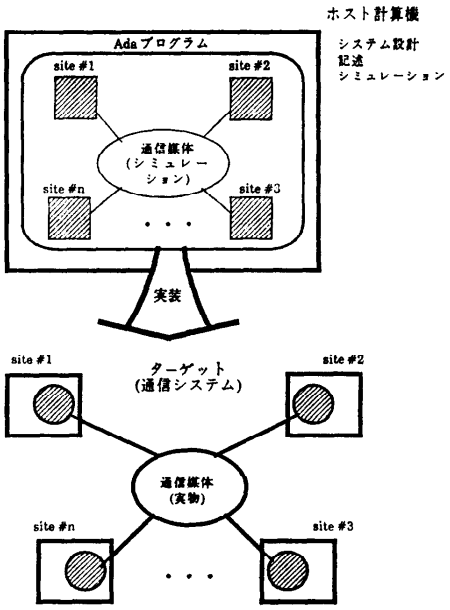


図-2 ホスト/ターゲット法によるシステム開発

ステムであり，それを構成する各計算機上で，それぞれ独立した Ada プログラムが個別に実行される．別別の計算機上で実行されるこれらの Ada プログラム間の通信では，タスク間のランデブを直接用いることはできない．したがって，計算機間を結ぶ通信媒体をアクセスすることによって通信を行うようにプログラムを修正せねばならない．

また，保守性ということが，Ada の主要な目標の一つであるので，システムの記述・実現に Ada を用いることが保守段階においても有効であることが期待される．

この方法では，本章の冒頭に掲げたシステム開発のライフサイクルのほとんど全段階で一貫して Ada を用いる．そこでは，仕様記述，設計，プログラミングという段階の境界が判然としないものになっている．Ada をシステム開発に用いることになれば，従来のものとは異なるシステム開発ライフサイクルを考える必要がでてくるであろう．

3. 通信システム開発事例

3.1 通信システムの概要

今回，例として開発した通信システムのモデルを**図-3** に示す．1台のプリンタサーバと複数台のワークステーションとから構成される．本通信システムは，ワークステーション上で作成したデータファイルを通

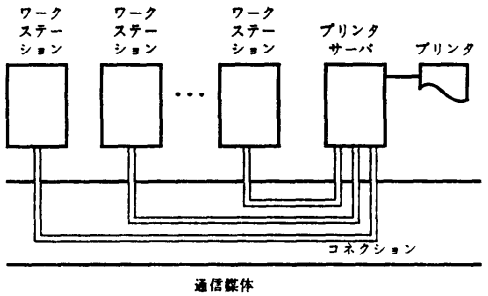


図-3 通信システムモデルの構成

信媒体を介してプリンタサーバに転送して，データファイルの出力を行うという機能を提供する．プリンタサーバは同時に複数のワークステーションと通信を行う．各ワークステーションとプリンタサーバとの間には1本のコネクションが存在する．ワークステーションは他のワークステーションとはなんの相互作用もなく独立に動作する．ワークステーションにおける1個のファイルが，ユーザの指示する転送の単位であり，かつ，プリンタサーバにおける出力の単位である．

ワークステーションとプリンタサーバとの間の通信を規定するプロトコルの階層構造と，それに基づくシステム構成モデルとを**図-4** に示す．図中最下位層のデータリンク層は，既存のものを用いることにした．したがって，通信システムの中でバルク転送層* の部分と応用層の部分とが行う処理を，今回 Ada で実現することとなった．ワークステーション側の応用層は，ユーザインタフェースをつかさどり，プリンタサーバ側の応用層は，転送されたファイルの管理をつかさどる．

バルク転送層が本通信システムの主要部分である．この層で行う処理の概要を**表-1** に示す．この層では，転送中のデータの紛失や制御コマンドの紛失に対する障害対策を行う．また，輻輳対策としてフロー制御を行う．本稿の主題は，Ada を用いた通信システムの開発であるので，プロトコルの詳細については省略する．

3.2 Ada によるプログラミング

上述の通信システムを九州大学工学部情報工学科の教育用計算機システム上に実現することを目標とし，米国国防総省の検定を通過した Ada 処理系[9] が稼働する日本データ・ゼネラル社の ECLIPSE MV/10000

* 本稿では，「バルク転送」という言葉を OSI などで呼ばれている「ファイル転送」と同様の意味で用いている．

表-1　バルク転送層で行う処理の概略

処理項目	処理の内容
データファイルの転送	1. 転送の開始/終了の同期 2. 再送制御 3. フロー制御 4. 転送途中の打ち切り
制御コマンドの転送	1. 出力の要求と応答の転送 2. プリンタサーバの状態/ファイルの有無の問い合わせとその応答の転送 3. 制御コマンドの監視
プロトコル状態のリセット	1. リセットの要求とその応答 2. プロトコル状態のリセット 3. 応用層へのリセット通知

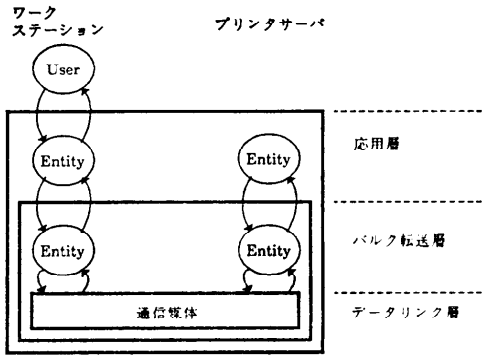


図-4　システム構成モデルとプロトコル階層

をホスト計算機として利用した．今回開発した通信システムは，1台のプリンタサーバと3台のワークステーションとから構成される．

記述した Ada プログラムの構成を図-5 に示す．図中，矢印は，制御コマンド，応答，転送データなどの通信システム内におけるデータの流れを表す．これらのデータは，Ada におけるタスク間通信機構であるランデブによって授受が行われる．ランデブにおけるデータの移動を図-5 で用いた記法を使って分類すると図-6 のようになる．

プログラミングに際しては，図-4 における各層内のエンティティをモジュール化の単位とすることを設計方針とした．これによって，階層化アーキテクチャ[18]を持つ通信システムを，実物との対応が良くとれたプログラムとして自然に記述できる．その結果，理解しやすいプログラムとなって，保守も容易になることが期待される．

各エンティティは論理的に互いに並行して動作可能であるので，Ada のタスクとして記述した．各エンティティはプロトコルで定められた動作をする一種の有限状態機械とみなせる[4),5)]ので，状態遷移表として与えられたプロトコルを Ada プログラム中でもそのまま表の形式で表現して，表駆動によって動作するタスクとして記述した．ただし，プリンタサーバ側のバルク転送層では，同時に複数台のワークステーションを相手にしなければならないために，処理が複雑にな

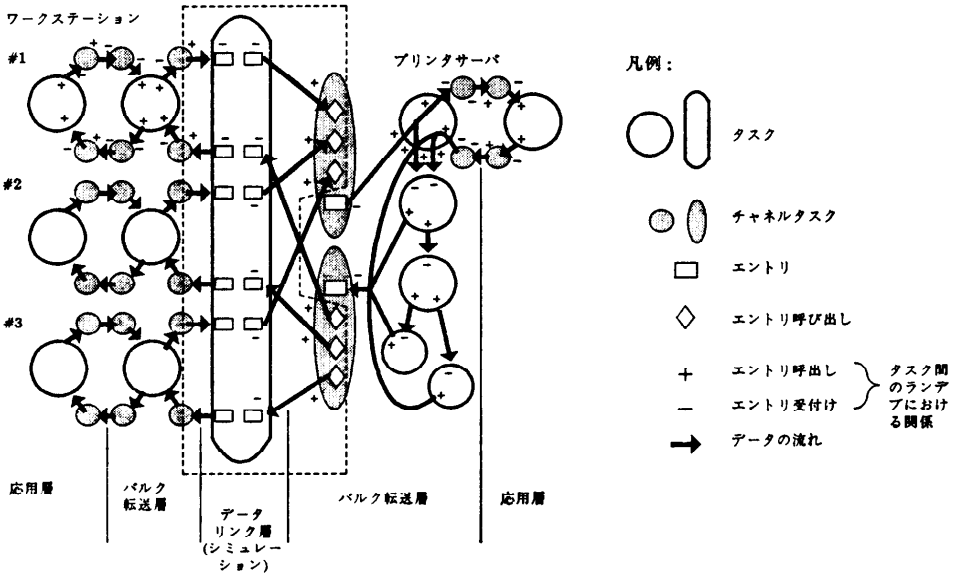


図-5　Ada プログラムの構成

accept ent(x : **in**...) **do**...

（a） 呼び出し側から受け付け側へ (in モード)

accept ent(x : **out**...) **do**...

（b） 受け付け側から呼び出し側へ (out モード)

accept ent(x : **in out**...) **do**...
あるいは
accept ent(x : **in**... ; y : **out**...) **do**...

（c） 双方向 (in out モード，または，in モードと out モードとをともに持つ)

図-6 ランデブにおけるデータの移動

```
type type_station is (WS_1, WS_2, WS_3);

generic
      name : type_station;
package workstation is
      .
      .
      .
end workstation;
      .
      .
      .
with workstation;
package workstation_1 is new workstation(name => WS_1);
package workstation_2 is new workstation(name => WS_2);
package workstation_3 is new workstation(name => WS_3);
```

図-7 ワークステーションの汎用体宣言と汎用体具体化

```
task data_link is
   entry to_workstation (type_station) (data : type_data);
   entry from_workstation (type_station) (data : out type_data);
   entry to_printer_server (type_station) (data : type_data);
   entry from_printer_server (type_station) (data : out type_data);
end data_link;
```

図-8 データリンク層をシミュレートするタスクの仕様

った．このため，この層のエンティティは一つのタスクとはせずに，機能によってさらにサブモジュールに分解し，それらのサブモジュールをタスクとして記述した．

各層の間には，チャネルタスクと名付けたタスクを設けて各層のエンティティタスク間のインタフェースを明確にするとともに，エンティティタスク同士が直接同期・通信を行わなくともよいようにした．このようなタスクは，文献 7) の分類に従えば，バッファタスク (buffer task) あるいは転送タスク (transport task) に相当する．

以下には，Ada が提供する機能を用いた具体例をいくつか挙げる．

（1） 汎用体によるワークステーションの記述

3台のワークステーションは，自分自身の識別とコネクションの識別を除けば，まったく同じ振る舞いをする．これらのワークステーションを記述するのに Ada の汎用体を用いた．**図-7** に，汎用体宣言（部分）と汎用体具体化とを示す．

ワークステーションは前述のようにいくつかのタスクからなっている．したがって，図-5 の Ada プログラムには同じ動作をするタスクが3個ずつ存在することになる．Ada では，タスクの配列が利用できるので，われわれもこれを用いることを一応考えはした．しかしながら，タスクの配列を用いた場合には，ワークステーションの識別を配列の添字によって行うことになる．そうすると，添字がなんらかの理由によって誤った値となって，本来相互作用のまったくない筈のワークステーション間で不正な相互アクセスが起こる危険性が存在する．また，実システム上への実装においては，配列要素を個別に取り出して別々の計算機上に割り当てなければならないけれども，それをどうやって行えばよいのか今のところわれわれには不明である．汎用体を用いれば，ワークステーションごとに具体化した各パッケージを別々の計算機上に割り当てることは容易である．

（2） タスクによるデータリンク層のシミュレーション

図-5 の Ada プログラムでは，図-3 のデータリンク層をタスクによってシミュレートしている．このタスクの仕様を**図-8** に示す．このタスク仕様は3本のコネクションに対するアクセスの方法を掲げている．タスク本体では，ワークステーション側とプリンタサーバ側との間に設定される3本のコネクションを通して行われるメッセージの転送を，単純なバッファリングによってシミュレートしている．その際に，選択待ち (selective wait) を用いている．

このように下位層を Ada プログラムでシミュレートすることによって，システムのテストの際に下位層の障害の影響を考慮する必要がなくなり，対象としている層のテストに専念できる．また，下位層で発生した障害に対する処理をテストするために，下位層をシミュレートする Ada タスク本体中で各種の障害を模擬的に発生させるように細工することもできる．

```
begin
   loop

      RECEIVE_ONE_DATA :
         loop
            for WS in type_station loop
               select
                  data_link.from_workstation (WS) (data);
                  exit RECEIVE_ONE_DATA;
               else
                  null;
               end select;
            end loop;
         end loop RECEIVE_ONE_DATA;

      accept pass (temp : out type_data) do
         temp := data;
      end pass;

   end loop;
end;
```

図-9 データリンク層からのデータをプリンタサーバのバルク転送層へ渡すチャネルタスクの本体における処理

```
task channel_from_data_link_to_workstation is
   entry receive (data : in out type_data; event : type_event);
end channel_from_data_link_to_workstation;

task body channel_from_data_link_to_workstation is
begin
   loop

      accept receive (data : in out type_data; event : type_event) do
         start := time;
         catch := start;

         loop
            select
               data_link.from_workstation (data.name) (data);
               catch := time;
            or
               delay time_limit - (catch - start);
               data.event := reset;
            end select;
            exit when data.event = next(event)
                     or data.event = reset;
         end loop;
      end receive;
   end loop;

end channel_from_data_link_to_workstation;
```

図-10 時限エントリ呼び出しを用いたタイムアウト処理

（3） チャネルタスクにおけるポーリング

データリンク層からのデータをプリンタサーバ側のバルク転送層へ渡すチャネルタスクは，3本のコネクションとの間のデータの流れを1本のデータの流れにまとめている．このチャネルタスクの本体を**図-9**に示す．各ワークステーションからデータが届いているかどうかを，即時エントリ呼び出し (conditional entry call) を用いて順に調べて，いずれかのワークステーションからのデータを受け取ると直ちにプリンタサーバのバルク転送層に引き渡す．いわゆるポーリングを行っているのであるけれども，ここでは単純に3台のワークステーションを毎回同じ順序で調べるようにしている．

表-2 Ada プログラムの大きさ

	ソースプログラムの行数
ワークステーション	約 1,100
プリンタサーバ	約 1,500
データリンク層のシミュレーション	約 120
その他	約 300
合 計	約 3,000

Ada を用いたポーリング手法については，Welsh と Lister[21]による考察がある．ただし，ポーリングを行うと，効率の低下を招いたり，プログラムが複雑になったりするので，ポーリングは行わないようにするのが良いとの指摘もある[7]．われわれは，Ada 利用の経験が浅く，そこまでは考えが及ばなかったのであるけれども，そのようなことを含む Ada プログラミング技法については現在研究を進めているところである．

（4） タイムアウト処理

データリンク層からのデータをワークステーション側のバルク転送層へ渡すチャネルタスクを**図-10** に示す．ワークステーションのバルク転送層が receive エントリを呼び出して，プリンタサーバからのデータを要求すると，このチャネルタスクはデータリンク層の受信用エントリ (data_link. from_workstation (data. name)) を呼び出して，受信データを受け取る．ある時間待っても受信データが届かなければタイムアウトとなるけれども，ここでは，時限エントリ呼び出し (timed entry call) を用いてタイムアウト処理を記述している．

参考までに，本章で述べた Ada プログラムのソーステキストの大きさを**表-2** に掲げる．ワークステーションの大きさは，汎用体宣言の大きさ，すなわち，1台分の大きさである．「その他」には，共通なデータ型の宣言などが含まれる．

3.3 Ada プログラムの実行

上述の Ada プログラムを，ホスト計算機である MV/10000 上で実行した．これは，2章で述べたように，一面ではプロトタイピングと見なすことができ，与えられた問題に対する理解がこの段階で深まった．また，システムの動きを実際に確認することによって，プロトコル設計の妥当性を確認したり，プロトコルにおける曖昧な箇所を指摘したりできた．

この段階は，通信システム完成に至る前のテスト段階であるともいえる．ホスト計算機上での実行ではあるけれども，通信システム全体の動作を観察すること

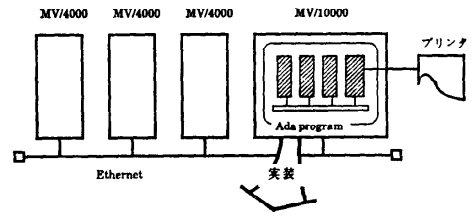
により，システム設計の不都合やプログラミングの誤りを発見できた．テストの際には，データの流れに着目し，タスク間のランデブによってどのようなデータがどこからどこに移動するかを表示させた．また，タスクについては，各タスクのコーディングが終わった時点で，それの本来の通信相手の身代わりを用意することによってそのタスクの動作環境を整えた上で，個別にテストを行った．

3.4 通信ソフトウェアの実装

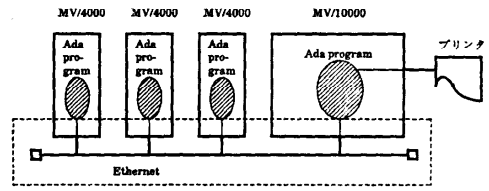
本通信システムを九州大学工学部情報工学科に設置されている教育用計算機システム上に実装した．本計算機システムでは，図-11 に示すように，4台の計算機が Ethernet で接続されている．その中の1台である MV/10000 をホスト計算機として利用し，その上で Ada プログラムの作成を行った．実装後は，この MV/10000 がプリンタサーバとなり，他の3台の MV/4000 がワークステーションとなる．

実装に当たっては，2章で述べた方法に従い，データリンク層をシミュレートするタスクを実物のネットワーク上のデータリンク層である Ethernet の機能で置き換え，その上で，それぞれのワークステーションならびにプリンタサーバの部分を個々のネットワークノード計算機に割り当てるという方針を立てた．ところが，本学科では，Ethernet で接続された計算機システムを X. 25 パケット層の運用の下に利用している．このために，本稿で述べた通信システムが X. 25 の運用と並行して直接 Ethernet にアクセスすることは事実上不可能である．よって，データリンク層の機能を，現在運用中の X. 25 パケット層の機能で代用することにした．われわれの関心は，通信システムそのものというよりは，通信システムの開発方法にあるので，本稿で述べた開発法によるシステム実現の可能性が確認できればよいとの判断のもとに X. 25 を利用することにしたわけである．効率低下にはこの際目をつぶることにした．

実装手順をもう少し詳しく述べる．図-11(a) の MV/10000 上では，通信システムを記述した Ada プログラムがシミュレーションとして動作している．斜線を施した部分がそれぞれのネットワークノード計算機の動作を記述した部分である．図-11(b) に示すように，それらの部分を取り出して個別にコンパイル・リンクし，それらをそのまま対応する計算機上に割り当てる．ただし，(a)では，通信媒体は，Ada タスクでシミュレートしていたけれども，(b)では実物の通信媒体を使用する．このため，通信媒体にアクセスする部分を適宜書き直す必要がある．これは，図-5 に示すように，データリンク層とパルク転送層との間のチャネルタスクの破線内の部分に限定しているために，その変更は容易であり，他の部分（破線の外側）は変更の必要がない．



(a) シミュレーション



(b) 完成した実システム

図-11 通信システムの実装

4. 使用経験に基づく Ada の評価

前章で述べた通信システムの開発における Ada 利用の経験を基に，Ada に関する評価を試みる．ただし，ここで述べることは，われわれの限られた経験によるものであることをあらかじめお断わりしておく．

4.1 言語の機能

3.2 節で述べた記述という面では，一応満足している．本稿の冒頭に列挙したような Ada の豊富な機能が，システムの設計や記述やテストにおいて役に立った．中でもタスクの利用が本質的であるとわれわれは考える．実際のところ，Ada の一番の特徴はタスクとパッケージであると言ってよかろう[12]．パッケージに関しては本特集の他の解説に譲ることにして，ここでは詳しくは述べない．

タスクを用いることによって，Ada プログラムの構成を実物の通信システムの構成とよく対応できるようにすることができた．すなわち，設計の際に考えた「事物」を素直に Ada のタスクとして記述できるのである．Ada では，「事物」に即したいわゆるオブジェクト指向の設計ないしはプログラミングを，タスクを

用いることによって容易に提供あるいは支援していると言える[6),7)].

タスク間の通信・同期の機構であるランデブでは，即時エントリ呼び出し，時限エントリ呼び出し，選択待ちを用いることができる．これによって，モニタ概念に基づく並行プログラミング言語を用いる場合と比較して，状況に応じた柔軟なプログラミングが行える[17)]．たとえば，Concurrent Euclid[13)] では，モニタエントリを呼び出したプロセスは，そのモニタエントリの受け付け待ちの行列に一旦入ると，待たされている途中で呼び出しを取り消して待ち行列から抜け出すことはできない．一方，Ada では，時限エントリ呼び出しを用いることによって，ある時間待たされてもエントリ呼び出しが受け付けられなければ，その呼び出しを取り消して待ち行列から抜け出しなにか他のことを行うことができる．

accept 文中に含ませる実行文によって，ランデブの同期期間の長さを調整することができることも，柔軟なプログラミングにとって重宝である．われわれの Ada プログラムでは，accept 文中で行う処理をできるだけ少なくすることによって，タスク間の同期期間を短くするようにした．

ランデブにおいて各タスクがエントリを呼び出す側とエントリを受け付ける側とのどちらになるかによって，プログラムの構成や効率が大きく変わってくることがある．われわれの場合では，通信システム内の層間に設けるのはバッファタスクにするか転送タスクにするかということや，3章で紹介したポーリングの使用についての忠告が，その例である．このような問題に関する Ada プログラミング技法の考案と評価が行われる必要がある．

厳密な型付けや例外がプログラミングにおける誤りを除去するのに大いに有効であったことは，言うまでもない．例外が伝播すると，デバッグの際にその例外がどこで発生したのかを特定しにくくなるので，例外が発生した枠 (frame) 内で処理して，例外の名前と枠の名前を出力することにした．われわれの場合で最も多く発生した例外は constraint_error であったけれども，そのうちの多くはプログラミング時の不注意によるものであった．

Ada プログラムにおける仕様は，情報隠蔽やデータの保護やインタフェースの明示という点で大いに有効である．しかしながら，たとえば手続きの仕様では，それの名前とパラメタの名前と型とを掲げるに過ぎない．このため，その手続きの使い方ないし呼び出し方は，仕様を見れば分かるけれども，その手続きがなにをするのかということに関しては，本質的になにも表していない．この意味の欠如とも言うべきことは，Ada の目標であるソフトウェアの部品化や保守性などに対する一つの短所であろう．これを補うために，ある程度の意味を Ada プログラムの仕様中に形式的に記述する方法も研究されている[14)]．Ada プログラムの意味記述は，今後の大きな課題の一つである．

4.2 Ada 処理系

われわれは検定を通過した Ada コンパイラを使用した．しかしながら，現状では，検定を通過しているからといっても，Ada 基準文法書を読んで期待される通りには必ずしも機能しないコンパイラもあるようなので，実際の利用においては注意せねばならない．というのは，Ada 基準文法書の 13 章に書かれている内部表現節と処理系依存機能にかかわる事柄については，その使用に際して，事前の細心なる調査を Ada 処理系と使用計算機とについて行う必要があるからである．

3.4 節に述べた通信システムの実装においては，これらの処理系に依存する事柄が問題となった．ところで，Ada が元来目標とする組み込み型計算機システム用のプログラムにおいては，当然のことながらこれらの事柄がかかわる場合が頻繁に起きると予想される．Ada 基準文法書の 13 章に書かれた機能を活用すれば，Ada だけを用いて，利用する処理系あるいはターゲット計算機システムに依存する部分と独立な部分とを明確に分離できて，抽象化やモジュール化を損なうことがないであろうと期待した．しかしながら，われわれが利用した Ada 処理系では，オペレーティングシステムとの関係からの制約などから処理系依存機能の一部が十分に提供されていないために，Ada だけでは解決できない実現上の問題が生じた．たとえば，レコード内部表現節 (record representation clause) を用いてレコード要素の位置や大きさを細かく指定できると思っていたところが，よく調べてみるとわれわれが使用した処理系では境界調節節 (alignment clause) を書いても無効であることなどが分かり，われわれの意図する内部表現を Ada だけで実現することができなかった．結局のところ，アセンブリ言語を使う場面がでてきてしまった．

また，Ada プログラムの実行時の効率については，改善の余地は大きい[8)]．検定を通過した Ada コンパ

イラは，1984年あたりから増え出したけれども，効率において満足できるものはまだ少ないようである[11]．われわれは豊富な機能を持つ Ada の処理系を作成した方々に敬意を表すると同時に，実用上十分な効率を満足するように改良されることを期待する．

4.3 ソフトウェア開発環境

今回の通信システムの開発は，いわゆるホスト/ターゲット法で行った．われわれの場合は，幸いにも，ホスト計算機とターゲット計算機とが同一機種であったために，通信システムの実装に関しては**4.2節**で述べたこと以外には大きな問題はなかった．しかしながら，ホスト計算機とターゲット計算機とは互いに異なる機種となる場合も多いであろう．その場合には，クロスコンパイラないしマルチターゲットのコードジェネレータが必須である．さらに，テスト計算機上にターゲット計算機の環境を構築して，テスト・デバッグを行える環境を用意する必要がある．これらは，Stoneman[19]のなかでも要求されている．したがって，その要求を満足する環境を利用できる状況であれば，われわれがここに書くことはなにもなくなるということになるのかも知れない．

われわれは，通信システムを構成するターゲット計算機群の環境をホスト計算機上にタスクを用いて Ada の言語仕様の枠内で（不十分ながら）記述した．今回は，例題として通信システムだけを対象としてホスト計算機上で開発を行ったので，一般的な分散型システムに対する開発支援環境をホスト計算機上に構築することはしなかった．分散型システムの開発を支援するテスト支援ツールやプログラム解析ツールなどが統合された環境は今後の大きな課題である．

コンパイルやリンクに要する時間は開発環境のなかで大きな位置を占めると言えるので，コンパイラやリンカに対する高速化への要求がなくなることはないだろう．また，Ada では，分割コンパイルが特徴の一つである．このために，ライブラリ管理やバージョン管理やコンパイル順序に関して利用者への便宜を図る環境が要求される．

5. おわりに

分散型システムの例として取り上げた通信システムの開発において，そのライフサイクルのほとんどすべての段階で Ada が有用であることを実際に確認できた．ことに，Ada がタスク処理機能を有するということが本質的に重要であると考える．

また，われわれは，実際のシステム開発において Ada がどれくらい有用であるかに興味があった．そして，それまでに Ada を実際に利用した経験はなかった．とにかく実際にやってみようということでで き上がったのが**3節**で述べた Ada プログラムである．したがって，このプログラムには，いろいろと不備なところがある．現在，システム設計や効率の観点からプログラムの見直しを行っている．併せて他の問題にも Ada を適用し，利用経験を基に，Ada プログラミングの種々の技法やシステム設計における指針などについての研究を行っているところである．

Ada 処理系が基準文法書の 13 章の機能を十分に満足し，Ada プログラムの実行時の効率が実用上満足できるものとなり，Stoneman を満足する開発環境が利用可能になれば，Ada の適用性はさらに向上するであろう．

今回報告した例に留まらず，他の多くの問題に Ada を適用することにより，経験を蓄積して（特に分散型の）ソフトウェアの開発において，その仕様記述から設計，実現，保守に至るまで一貫して Ada を用いる方法ならびに開発支援環境について研究を進めることをわれわれの今後の課題としたい．

謝辞 3章で述べた Ada プログラミングと実装についてそれぞれご尽力いただいた本学大学院修士課程樽本尚明氏（現在沖電気）と甲斐郷子氏，ならびにご協力いただいた日本データ・ゼネラルの関係諸氏に謝意を表する．三菱電機水野忠則氏にはプロトコルについてご教示いただいたことを感謝する．ただし，通信システムに関する本稿の記述の中におかしな所があるとすれば，それはひとえに著者の不勉強によるものである．また，本稿に関して有益なご助言を賜わった査読者の方々にも感謝する．なお，本研究は一部，文部省科学研究費総合研究(A)「並列処理機械とその言語に関する基礎的研究」の補助を受けた．

参 考 文 献

1) 荒木，牛島：Ada による分散型システムの開発について(I)—方針と概要—，情報処理学会第31回全国大会論文集，pp. 321-322 (1985).
2) 荒木，甲斐，樽本，牛島：Ada による分散型システムの開発について(II)—通信ソフトウェアにおける事例—，情報処理学会第31回全国大会論文集，pp. 323-324 (1985).
3) Araki, K. and Ushijima, K.: Integrated Method of Network Software Development with Ada, Proc. Pacific Computer Communi-

cation Symposium (1985).
4) Bochmann, G. V.: Finite State Description of Communication Protocols, Computer Networks, Vol. 2, pp. 362-372 (1978).
5) Bochmann, G. V.: A General Transition Model for Protocols and Communication Services, IEEE Trans. on Communications, Vol. COM-28, No. 4, pp. 643-650 (1980).
6) Booch, G.: Software Engineering with Ada, Benjamin Cummings (1982).
7) Buhr, R. J. A.: System Design with Ada, Prentice-Hall (1984).
8) 大黒, 荒木, 牛島: 使用経験に基づく Ada 処理系の評価, 昭和60年度電気関係学会九州支部連合会大会論文集 (1985).
9) Data General Corp.: Ada Development Environment (ADE) (AOS/VS) User's Manual 093-000373-01 (1984).
10) Duncan, A. G.: Prototyping in Ada: A Case Study, ACM SIGSOFT Software Engineering Notes, Vol. 7, No. 5, pp. 54-60 (1982).
11) 石井: 出そろった検定済み Ada コンパイラ, 日経エレクトロニクス, 1985-5-6号, No. 368, pp. 151-166 (1985).
12) 疋田, 徳田: パッケージとタスクの機能を備えた新言語 Ada の概要, 日経エレクトロニクス, 1981-12-21号, No. 280, pp. 130-162 (1981).
13) Holt, R. C.: Concurrent Euclid, the Unix System, and Tunis, Addison-Wesley (1983).
14) Luckham, D. C. and von Henke, F. W.: An Overview of Anna, a Specification Language for Ada, IEEE Software, Vol. 2, No. 2, pp. 9-22 (1985).
15) 水野: プロトコルの形式記述とコンフォーマンス試験, 情報処理, Vol. 26, No. 4, pp. 402-427 (1985).
16) Sunshine, C. A. (ed.): Communication Protocol Modeling, ARTECH HOUSE (1981).
17) 樽本, 荒木, 牛島: 高水準並列プログラミング言語を用いた通信システムの開発法について, 電子通信学会技術研究報告, EC 84-61 (1985).
18) 勅使河原: 開放型システム間相互接続 (OSI) の参照モデル, 情報処理, Vol. 26, No. 4, pp. 299-309 (1985).
19) United States Department of Defense: Requirements for Ada Programming Support Environments —"Stoneman" (1980).
20) United States Department of Defense: Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A (1983).
21) Welsh, J. and Lister, A.: A Comparative Study of Task Communication in Ada, Software Practice and Experience, Vol. 11, pp. 257-290 (1981).

(昭和60年11月8日受付)