6. 逐次型推論マシン(SIM)：プログラミングおよびオペレーティング・システム

ICOT 研究所第3研究室長　横井俊夫
ICOT 研究所第3研究室長代理　内田俊一

アブストラクト　日本の FGCS（第5世代コンピュータ・システム）プロジェクトの最初の主要な成果として，パーソナル逐次型推論マシン（PSI）を現在，開発中である．ソフトウェア・システムを含めたシステム全体は，SIM と呼ばれている．以下，SIM のプログラミング・システムとオペレーティング・システム SIMPOS，主要な言語 ESP（Extended Self-Contained Prolog），開発ツール，および今迄の研究・開発経過を述べる．

SIMPOS の主要な研究課題は論理プログラミングをベースにしたプログラミング環境とシステム・プログラムの開発である．

SIMPOS の基本設計は，一様なフレームワーク（論理プログラミング）に基いたシステムのもとで，データベース機能および日本語の自然言語処理機能を有するスーパー・パーソナル・コンピュータを構築することである．（横井 et al. 1983a, b）．

1. まえがき

日本の FGCS プロジェクトの最初の主要な成果として，現在，SIM を開発中である．以下，SIMPOS の全般的な設計，主要言語 ESP，開発ツール，および今迄の開発経過を述べる．

SIMPOS の主要な研究課題は，以下の開発である．

・論理プログラミングによるシステム・プログラム
・論理プログラミングへのプログラミング環境

SIM は，FGCS ソフトウェア開発のパイロット・モデルである．SIM は高性能パーソナル・マシンで，FGCS プロジェクトの中期用の研究ツールとして使用される．

SIMPOS の基本設計方針は次の5つである．

・一様なフレームワークに基いたシステム設計

PROLOG 型論理プログラミングのみに基くフレームワークで，マシン・アーキテクチャ，言語システム，オペレーティング・システム，およびプログラミング・システムすべてをカバーする．

・パーソナル対話システム

SIM は，多くのスーパー・パーソナル・コンピュータ同様，パーソナルで高い対話機能を持ったシステムを想定している．

・データベース機能

PROLOG は，関係データベース・システムに順応しやすいデータベース機能を持っている．我々は，データベース機能を十分に駆使する新しいプログラミング・システムおよびオペレーティング・システムを開発する．

・ウィンドウ機能

高レベル対話処理を容易にするため，SIM はビット・マップ・ディスプレイとポインティング・デバイスを使用する．

・日本語処理

現在，コンピュータはすべて西洋の文化を基盤としている．しかしこれでは，他の文化圏の人がコンピュータを使うときは非常に不都合である．コンピュータは，すべての人が自国語で使用できるものでなければ

ならない．日本人にとっては日本語で使えるようにする．

SIMPOSは，プログラミング・システム（PS）とオペレーティング・システム(OS)で構成されている．OSは，カーネル，スーパバイザ，および入出力メディア・サブシステムで構成されている．PSは，エキスパートと呼ばれるサブシステムで構成されている．PSのサブシステムはユーザにより利用されるが，多数のサブシステムやプロセスを調整する必要がある．この作業は，コーディネータにより行なわれる．

その他のサブシステムを以下に列挙する．

ウィンドウ（OS）
ファイル（OS）
ネットワーク（OS）
デバッガ/インタプリタ（PS）
エディタ/トランスデューサ（PS）
ライブラリ（PS）等

2. 記述言語：ESP

2.1 言語概要

SIMPOSは，ESPと呼ばれるユーザ・プログラミング言語で作成される．(近山 et al. 1983)(近山 1984a)．ESPはSIMPOS作成用に特別に設計し開発されているが，各種アプリケーション，特に階層知識表現が必要なアプリケーションにも有用である．

PSIのマシン語であるKL0のほとんどすべての機能は，ESPで直接使用できる（高木 et al. 1983)．ESPはKL0のProlog型実行メカニズムに基づいているため，当然論理プログラミング言語機能を持っている．これらの中で重要な機能は，ユニフィケーションによるパラメータ受渡しおよびバックトラッキングによるAND-OR木のサーチ・メカニズムである．

論理プログラミング言語としての特長以外に，ESP言語は以下の特長を持っている．

・状態を持つオブジェクト
・オブジェクト・クラスと継承メカニズム
・マクロ展開

ESPの機能に関する詳細は，(近山 1984b)を参照されたい．

2.2 実現法

現在，ESPのオブジェクト指向機能はすべて，KL0の機能を使用して作成されている．ESPで作成されたプログラムはコンパイルされてKL0になる．オブジェクト指向呼出しは実行時サブルーチンの呼出しに変換され，これらのサブルーチンでメカニズムを実行する．オブジェクト指向呼出しは通常のKL0述語呼出しに比較し，3〜4倍遅くなっている．

ESPの実行速度を向上させるために組込み述語をいくつか用意するつもりである．

このようなファームウェア・サポートにより，オブジェクト指向呼出しの実行速度は通常のKL0述語呼出しよりやや遅い程度の速度になるものと思われる．

実行速度向上のためにさらに別の方法も採用されている．それは，ソース・プログラム・レベルでの最適化である（沢村 et al. 1984)．

3. オペレーティング・システム

SIMPOSオペレーティング・システムは，カーネル，スーパバイザ，および入出力メディア・サブシステムの3つの層で構成されている（服部，横井 1983)（服部 et al. 1984a, d, e)（高木 et al. 1984)．

3.1 カーネル

カーネルは，PSIハードウェアとスーパバイザ間のギャップを埋めるためにハードウェア資源を管理する（川上 et al. 1984)．プロセッサ管理はマルチプロセス環境を実現し，メモリ管理はメモリ・スペースを管理してゴミ集めを実行し，デバイス管理は入出力装置を制御する．

3.2 スーパバイザ

スーパバイザはオブジェクト蓄積場所，プロセス間通信，および実行環境といった実行のための基本機能を提供する（服部，横井 1984c)．これらの機能は，ユーザの選択に従い拡張および変更が可能である．

プールは任意のクラスのオブジェクトの入れ物であり，また自身がオブジェクトでもある．リストや配列はプールの一例である．オブジェクトはプールから取り出したり，入れたりすることができる（斎藤 et al. 1984)．

ディレクトリはオブジェクトのプールで，オブジェクトには名前が対応付けられる．オブジェクトはディレクトリ内の名前にバインドされ，その名前で検索できる．ディレクトリ内に別のディレクトリを含むことができるので，ディレクトリの木を構成できる．この場合，オブジェクトはパス名で識別される．

ストリームは，オブジェクトが流れるパイプである（島津 et al. 1984）．オブジェクトはストリームの一方から入り，他方から取り出される．ストリーム内にオブジェクトがない場合，取り出し操作はオブジェクトがストリームに入ってくるまでとめられる．ストリームは，プロセス間の同期と通信に使用される．

ストリームの上にチャネルを定義し，プロセス間のメッセージ通信を可能にする．メッセージはチャネルを介して送受信される．ポートは双方向通信用のメッセージ・ボックスで，チャネルにより別のポートと接続されている．ポートから送信されたメッセージは接続されているポートに到着し，そこで受信される．

プロセスはプログラムを実行するが，プログラムはプログラム・クラスのインスタンスである．プログラムのメイン・コールはインスタンス述語として定義され，プログラムインスタンスのスロットはそのプログラムに対し局所的なオブジェクトを保持する．

実行環境は，プログラム，ライブラリ，ワールド，およびユニバースで構成される（渡辺 et al. 1984）．これらは，プログラムの任意の箇所から参照できる．ワールドは一連のディレクトリで，各プロセスはワールドを作業ワールドとして保存する．ユニバースはシステム全体のディレクトリ木である．

3.3 入出力メディア・サブシステム

入出力メディア・サブシステムは，外部ワールドとのインタフェースを司る．このサブシステムは，ウィンドウ，ファイル，およびネットワークの3つのサブシステムで構成される．

3.3.1 ウィンドウ・サブシステム

ウィンドウ・サブシステムは，SIMの高水準マン・マシン・インタフェースをサポートする（辻 et al. 1984）（飯間 et al. 1984）．このサブシステムは単一の物理ディスプレイに複数の論理ディスプレイ（ウィンドウ）を生成し，これらに対するプリミティブな機能を提供する．エコーまたはカーソル制御などの機能はトランスデューサおよびコーディネータ・サブシステムによりサポートされる．

ウィンドウ・サブシステムでは，ウィンドウは階層を構成する．最も上位なウィンドウは論理画面で，通常のウィンドウは論理画面の下位画面である．各ウィンドウは，下位ウィンドウ（サブウィンドウ）を持つことができる．たとえば，エディタ・ウィンドウはコマンド・サブウィンドウ，テキスト・サブウィンドウなどを持つことができる．ウィンドウは物理画面上では矩形として表示され，サブウィンドウはその上位ウィンドウの内部になければならない．ウィンドウは画面上で重なり合うことができる．各ウィンドウは1つのプロセスにそのディスプレイとして専用に使用されるが，このディスプレイは，プロセスとマシンのユーザ間の交信用チャネルとなる．ウィンドウ上の出力は，ウィンドウ・サブシステムにより画面上の適当な位置に表示される．キーボードからの入力を受け取ると，ウィンドウ・サブシステムはその入力を送るべきウィンドウ（選択ウィンドウと呼ばれる）を決定する．ポインティングデバイスであるマウスは画面上の任意の箇所に移動でき，ウィンドウ・マネージャはマウス・クリックをマウスの位置に従って，選択ウィンドウまたはバックグラウンド・ウィンドウに送る．プロセスは，ウィンドウによりキーボードおよびマウス入力を読み取る．

3.3.2 ファイル・サブシステム

ファイル・サブシステムは，データおよびオブジェクトを長期的に記録する（服部，横井，1984b）（小松 et al. 1984）．

データ（レコード）の長期的記憶場所はファイルである．ファイルはディスク・ボリューム上に存在し，分散したディスク・ページで構成される．バイナリ・ファイル，テーブル（固定長レコード）・ファイル，およびヒープ（可変長レコード）・ファイルの3種類のファイルがある．レコードは格納位置またはインデックス・ファイルを介したキー（あるいはその両方）により識別される．バインダ・メカニズムは多くのデータ・ファイルやインデックス・ファイルをもつ仮想ファイルを作成できるように，サポートされる．関係データベース管理はこれらの機能に基づいて作成できる．

オブジェクトの長期的記憶はインスタンス・ファイルで，このファイルでは各オブジェクトがインスタンス・レコードとして記憶される．これは，ファイル・サブシステムの重要な機能の1つで，他のマシン上の通常のファイル・システムでは提供されていない．

ディレクトリ・ファイルは，インスタンス・レコードと名前を対応付けるファイルである．パーマネント・ディレクトリは，長期記憶としてディレクトリ・ファイル

を持っているディレクトリである．

パーマネント・ディレクトリに記録されると，パーマネント・オブジェクトはインスタンス・ファイルにインスタンス・レコードとして記憶され，パス名付きでディレクトリ・ファイルに入れられる．したがって，システムが再ブートされたときでも復元可能である．

3.3.3 ネットワーク・サブシステム

ネットワーク・サブシステムは，他のマシンと交信するために3種類のインタフェースを提供する（高山，服部1984）．

マシン間通信機能は，あるSIMと別のSIMまたは他のマシン間のデータ転送をサポートする．ネットワーク・サブシステムは通信を実現するために，ノード，ソケット，ケーブル，およびプラグなどのクラスを定義する．プロセス間通信機能により，異なったSIMノード上の2つのプロセスが，同じノードにある場合と同様に通信できる．リモート・チャネルが，他方のノード上の実チャネルを表わすために定義される．プロセスはリモート・チャネルにメッセージを送信でき，リモート・ノード上の別のプロセスは対応する実チャネルからそのメッセージを受信できる．

リモート・オブジェクト操作機能により，リモート・ノード上のオブジェクトを処理できる．ローカル・ノード上のリモート・オブジェクトはリモート・ノードのオブジェクトを表わし，実オブジェクトを操作するために通常のオブジェクトと同様な操作ができる．ネットワーク・サブシステムはこの機能をサポートし，SIMPOSをネットワーク・オペレーティング・システムにするつもりである．

4. プログラミング・システム

SIMPOSのプログラミング・システムは，エキスパート・プロセスの集合である．エキスパート・プロセスとは，独立の通信ウィンドウ(e-ウィンドウと呼ばれる)を持っているプロセスである．エキスパート・プロセスはユーザ要求に対し特定の処理を実行する．

この見方は，プログラミング・システムが単純なソフトウェア・ツールの集合であるとか，プログラム生産をサポートするためのプログラムの集合であるとかいう従来の見方とは異なっている．我々の見方によれば，使用可能なツールやプログラム間の情報を管理するための制御のオーバヘッドがなくなる．

ユーザの視点で考えると，ユーザはエキスパートのe-ウィンドウを通じてエキスパートを呼出し，制御し，終了することが可能である．自分の作業を実行するために，複雑なプロセス呼出し木を探索する必要はない．また，探索が間違っていたために自分の作業を壊してしまうことを心配する必要もない．

4.1 コーディネータ

SIMPOSでは，UNIXにおけるShellのような明示的な監視プロセスはない．しかし，コーディネータという名前のプロセスがある．コーディネータ自身はエキスパート・プロセスではなく，エキスパートの集合を管理するプロセスである（黒川，東条1984）．

上に述べたように，ユーザはウィンドウを通じエキスパートを直接制御すると考えることができるが，実際はコーディネータがそのウィンドウと対応するキー・コマンド・テーブルであるウィンドウ・インタフェースを通じ，ユーザの制御を支援している．

コーディネータの詳細は(黒川1984)で説明している．コーディネータの主要機能を以下に簡単に説明する．

- ウィンドウを通じ，ユーザのキー・コマンドをエキスパートに送る．
- システム・メニュを通じ，エキスパートを作成し，削除し，起動する．
- エキスパートから特殊コマンドを受け取り操作する．
- ホワイトボードを通じ，エキスパート間の通信を援助する．

ホワイトボードは黒板と同じであるが，エキスパートがここに他のエキスパートに対するメッセージを書き込むと，相手のエキスパートがユーザの命令によりそのメッセージを受け取る．

また，この通信問題は，他のエキスパートとの間に通信チャネルを設定する方法でも解決できる．しかしこの場合，ユーザがエキスパートの相手を決める前にエキスパート間にチャネルを設定しなければならない．通信が必要になる前に相手が誰であるかを知ることは容易ではない．

この方法を押し進めると，エキスパート数が増加するにつれそのコストが非常に高くなるであろうが，2つのエキスパート間すべてに通信チャネルを設定することであろう．しかし，それでもまだ問題は残る．ユーザは，

エキスパートにメッセージ書込みを依頼した後に相手を変更するかもしれない．マウス・クリックだけでは相手とメッセージの両方を示すことは難しいだろう．

ホワイトボードを使うことで，エキスパート間にほぼ完全な通信チャネルを設定できる．ユーザーは，あるエキスパートに対しメッセージ書込みを依頼した後に，相手のエキスパートを選択できる．この操作は，1つのマウス・クリックで実現できる．

各ユーザーは，エキスパートを作成するためにディレクトリを持っている．そのディレクトリには，エキスパート名およびエキスパートを作成するためのプログラム名が書かれている．ユーザーは，ディレクトリおよびコマンド・テーブルを任意に変更できる．

ユーザーはシステム共通ディレクトリ，つまりエキスパートの標準セットを継承したユーザ自身のディレクトリを持っている．

エキスパートは，そのウィンドウに対応されたキー・コマンド・テーブルのセットを持っている．しかし，コーディネータは，そのウィンドウがユーザーから change key command table コマンドを受け取ったときだけ，ウィンドウのキー・コマンド・テーブルを変更することをユーザーに許す．

この自由度は，最小の実行コストで達成される．このようにオーバヘッドが小さくユーザが最大限に制御できるのは，コーディネータの大きな成果である．

4.2 デバッガ/インタプリタ

このサブシステムはプログラムを解釈し，プログラムの流れに関する情報を提供する．デバッガ/インタプリタ・サブシステムの基本的な機能は，DEC-10 PROLOG (Brown et al. 1981) に準ずる．新規の機能を次に挙げる．

・手続きおよびクローズ・ボックス制御フロー・モデル
・インタプリティブおよびコンパイル・コード間の呼出し
・マルチ・ウィンドウ・ユーザ・インタフェース

DEC-10 PROLOG は，デバッガとして Box Control Flow Model を使用する．このモデルでは，各述語をデバッグ単位と見なす．この考えでは，各クローズは暗箱となり，ユニフィケーションが失敗したのが頭部なのか本体なのかが追跡できない．述語呼出しは，両方の場合とも失敗する．しかし，クローズの頭部が正しく選択できても本体が誤まりであることがよくある．Procedure and Clause Box Control Flow Model を使うと，頭部のユニフィケーションか本体のユニフィケーションのいずれが失敗するかをチェックできる（図1）．

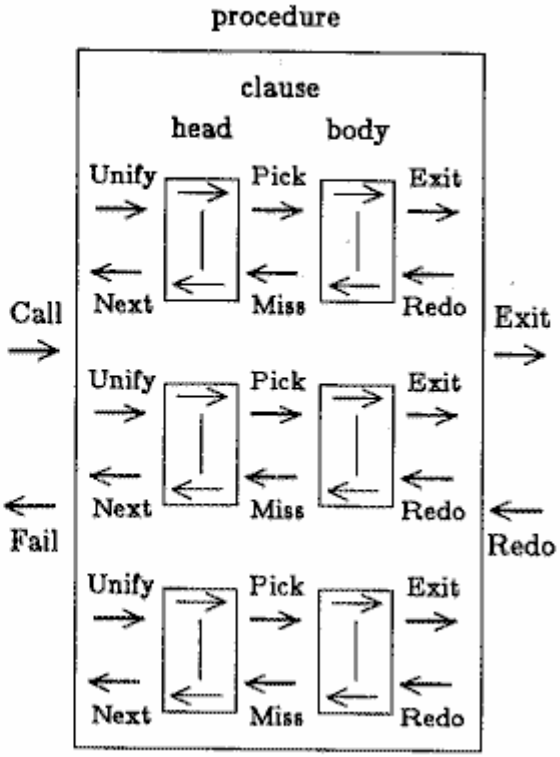

図1. インタプリティブコードに対する Procedure and Clause Box Control Flow Model

SIM では，インタプリティブおよびコンパイル・コードが互いに呼び合うことができる．しかし，デバッガはコンパイル・コードを追跡できない．デバッガはコンパイル・コードの呼出しを，組込み述語呼出し同様に扱う．インタプリティブ・コードがコンパイル・コードから呼び出された場合，追跡情報をインタプリティブ・コードに渡す方法はない．このような場合，デバッガは追跡情報なしで追跡を再開する．

SIM はビットマップ表示画面を持っている．デバッガはウィンドウ・サブシステムを使用するが，このサブシステムはマウスを使ったマルチ・ウィンドウ・ユーザ・インタフェースを提供している．ユーザはブレーク・ポイントで制御オプションの1つを選択するか，祖先またはスパイ・ポインタを調査するか，スロットの値を検査するか，またはライブラリ・サブシステムを使ってクラス定義を調べることができる．この情報はデバッガのサ

ブウィンドウに表示され，すべての選択はマウス・クリックを使って行なわれる．

4.3 エディタおよびトランスデューサ

エディタはプログラミング・システムの代表的な構成要素で，コンピュータ・システムを使用する上においては不可欠なソフトウェア・ツールである．テキストとは全く異なった抽象的な構造を操作するエディタもないわけではないが，以下では，テキストまたはテキストで表わされたデータを編集するエディタの説明に限定する．

テキスト表現でも通常，ネスト構造になっているので，SIM のエディタ（Edips と呼ばれる）は一般的に構造化テキストを操作するように設計されている．しかし，すべての構造に便利な汎用エディタが可能であるとは考えない．出来のよい汎用エディタとは，特殊な目的に便利で，しかも，それ程強力でなくても汎用に使用できるものである．この基準のもとで，Edips は ESP プログラムの編集に便利なように設計され，他の構造も操作できる．さらに，Edips は以下の機能を持っている．

- マクロ定義によるカスタム化
- コマンドの数が少なく，覚え易い．
- フェイルソフト機能

Edips に汎用性を持たせるため，ユーザが構文を定義できるようにした．他の汎用構造エディタは通常 BNF を使用しているが，我々はそれを採用していない．それは，通常の編集操作では構文木の枝刈りも木の探索もないからである．編集操作は，編集されるデータのテキスト表現と深く関係している．したがって，我々はユーザ定義かっこも許す演算子優先文法を採用した．演算子優先文法は簡単で，テキスト表現に良く対応する．

編集データのテキスト表現中のすべてのトークンは，以下の 6 つの範ちゅうに分類される．

- アトム
- 前置演算子
- 間置演算子
- 後置演算子
- 左かっこ
- 右かっこ

各演算子には優先順位がある．しかし，優先順位はテキスト構造と直接対応しない構造を生じるため，編集のためにはあまり多くの優先順位レベルを使用しない方がよい．ESP エディタの場合，2〜3 レベルが必要かつ十分である．以下のようなものがある．

- “:-”，“,”，“:” などの論理記号
- “+”，“—”，“*”，“/” などの関数記号

必要なら，以下を追加する．

- “<”，“>”，“=” などの述語記号

演算子優先順位文法の他に，トークン定義のために通常の正規表現も採用した．文字列であるテキストはまずオートマトンによりトークン列に変換され，次に構造に解析される．したがって，文法は 2 レベル文法である．しかし，両方のレベルとも簡単であるため文法は扱い易く，しかも，ほとんどすべての構造化プログラミング言語の構文定義に十分な表現力を持っている．

この文法用のパーサーやプリティ・プリンタはコンパイラ，インタプリタ，およびデバッガなどの他のプログラミング・ツールで使用できる方が望ましい．したがって，これらのツールがエディタとは別にユーティリティ（トランスデューサと呼ばれる）として用意されている．したがって，Edips はエディタ 中核部とトランスデューサで構成されている．

4.4 ライブラリ

ライブラリ・サブシステムは，SIM 上のすべてのクラスと述語を管理する．このサブシステムは，クラスの登録，プログラム・ファイルのロード，コンパイル，および継承分析によるクラス・オブジェクトの作成を制御する．

各クラスはクラス・ソース・ファイル，クラス・テンプレート・ファイル，クラス・オブジェクト・ファイルを 2 次記憶上に持っている．クラス・テンプレートおよびクラス・オブジェクトは主記憶にのみ存在するが，2 次記憶との間で退避・復元が行なわれる．

クラス・ソース・ファイルは，ユーザにより作成されたテキスト・ファイルである．クラス・ソース・ファイルは，クラス定義を 1 つだけ持っている．ソース・ファイル同様，テンプレート・ファイルもオブジェクト・ファイルも，クラス情報を 1 つだけ持っている．

クラス・テンプレートは単一のソース・ファイルから作成される．このテンプレートには，継承分析の情報を除いたそのクラスの全情報が入っている．そのクラスの述語は，テンプレート作成時にインタプリティブ・コードとして保持される．これらは，ユーザが要求するとコンパイルされる．コンパイル後，インタプリティブ・コ

ードとコンパイル・コードは両方とも保持される．テンプレートは，述語のコンパイル以前に退避・復元できる．

クラス・オブジェクトは，いくつかのクラス・テンプレートから作成される．クラス・オブジェクトでは，すべての継承が解析され解決されている．クラス・オブジェクトは，オブジェクト指向プログラムの実行可能イメージである．

ライブラリ・サブシステムのもう1つの機能は述語の管理である．このサブシステムはクラスの1つの述語を参照する機能を持っている．つまり，オブジェクト指向呼出し，およびコンパイル・コードからインタプリティブ・コードまたはその逆の呼出し機能を持っている．このメカニズムは間接参照により実現されている．述語の呼出しはすべて間接参照により行なわれる．インタプリティブ・コードが呼び出されると，その間接語がインタプリタの入口を指す．このメカニズムにより，インタプリティブおよびコンパイル・コードが混在している場合でも一様な呼出し方法が実行される．

オブジェクト指向呼出しの場合，実行時にどのメソッドを呼出すかを見付ける必要がある．この場合，ライブラリは，同じ名前でも別のクラスで定義されている述語を区別する必要がある．コンパイル・コードでは，すべての参照は処理され特定の述語の直接呼出しに変換されるが，インタプリティブ・コードの場合，ライブラリは実行時に述語を探索しなければならない．

コンパイラは，単にライブラリ・サブシステムのサブルーチンであり，インタプリティブ・コードから述語単位でコンパイルする．このプロセスは主記憶上でのみ行なわれる．コンパイル後，ライブラリにはインタプリティブ・コードとコンパイル・コードの両方が入る．ユーザは新しいクラス・オブジェクトの作成にどちらのコードを使用するか指定できる．テンプレート・ファイルはコンパイル後，自動的に再作成される．

4.5 例外処理とヘルプ・システム

一般的に，例外はソフトウェア・システムにおいて重要な概念の1つである．たとえば，次のようなものが例外に含まれる．ハードウェアが検出したゼロ除算，装置が検出した入出力エラー，ソフトウェアで検出したエラー，および複雑にネストされた手続き呼出し内のグローバル出口などの各種エラー，および作業中のユーザーに対するヘルプ機能．

従来のシステムでは，これらの例外は十分に対処されていない．これらは，各サブシステムまたはシステム・レベルで別に独立して扱われている．

SIMPOSでは，例外処理システムはすべての構成要素およびシステム・レベルに与えられている．我々の主目標および基本的枠組は次のとおりである．

・一様な枠組

例外は，ソフトウェア/ハードウェアの一部により生成され，通知される．この処理実行部はdetectorと呼ばれる．通知された例外は，例外処理プログラムを使ってプロセスにより処理されなければならない．この処理部分はhandlerと呼ばれる．

・無制限の例外登録

SIMPOSのように発展中のシステムでは，何種類の例外が使用されるか予想できない．したがって，何種類もの例外をできるだけ細かく分類して登録する方法が必要である．多重継承メカニズムで，上記目標がサポートされるはずである．

・柔軟な例外処理

同じ例外でも，発生する環境によりさまざまな意味をもつ．各例外の発生は，その意味に従って別々に処理しなければならない．

detector-handlerの方法は，文脈メカニズムの意味において強力である．detectorは環境を意識する必要がない．またhandlerも，ハンドラが文脈により選択されるのならば，文脈を意識する必要がない．

次のように実現される．

eventとsituationの2つの基本クラスが存在する．eventは，一般的な例外の名前である．eventには，2つの基本サブクラス，errorとhelpがある．errorには，警告，致命的エラー，および通常のエラーなど，いくつかのサブクラスがある．helpにも，一般的なhelpおよびキーボードhelpなど，いくつかのサブクラスがある．situationは，例外処理の文脈の名前である．可能性のある（または危険な）例外に対するハンドラは，situationに設定される．situationはスタックに似た構造を持ち，各プログラムはプロセスに対し必要なハンドラを設定し，プロセスが正常に終了すると不要なハンドラを削除することができる．

この方法における問題点の1つは，いくつかのハンドラが1つのeventに対して呼び出される場合があること

である．situation はハンドラを分類し，非決定モードでそれらのハンドラを適用するメカニズムを用意している．

ヘルプ機能は，最も広い意味で調査される．会話型のオンライン・マニュアル探索機能だけでなく，キーボード入力の補完メカニズムまたはスペル訂正機能も含まれている．

catch-throw または Lisp 言語における errset などのグローバルな出口も，この event-situation メカニズムを使用して実現されている．アプリケーション・プログラムに共通なコマンド・ループも提供されている．

5. ソフトウェア開発ツール

5.1 ESP クロス・システム

SIMPOS はすべて ESP で作成されている．ハードウェアが使用可能になる前に設計されコーディングされていたので，ソフトウェア開発用ツールのための ESP クロス・システムが必要であった．

ほとんどのプログラムは PROLOG で書かれているが，一部はメインフレーム・マシン上の PASCAL で書かれている．以下にプログラムを列挙する．

- ESP インタプリタ
- ESP クロス・コンパイラ（KL0 への）
- KL0 クロス・コンパイラ
- KL0 クロス連係編集プログラム
- 連係編集プログラムのデータベースを調べるための各種ユーティリティー・プログラム

5.2 実行時サポート・システム

ESP のオブジェクト指向呼出しメカニズムは，ESP コンパイラにより実行時サブルーチンへの呼出しへ変換することで実現されている．ESP の実行時サポート・システムは，このような実行時サブルーチンの集まりである．実行時サポート・システムは KL0 で直接記述されており，以下の機能を持っている．

- ESP の基本的なオブジェクト指向型呼出しメカニズム
- オブジェクト指向型呼出しのニーモニック追跡機能．この機能は手続きボックス制御フロー・モデルに基づいていて，各種の対話型制御（スキップ，再実行など）が可能である．
- ニーモニック形式でのオブジェクト・スロット値の検査．

上記機能に必要な入出力はすべて，コンソール・プロセッサの入出力装置を使用する組込み “read-console” および “display-console” により行なわれる．コンソール・プロセッサは主にハードウェアおよびファームウェア・デバッギング用に用意されたもので，PSI ハードウェアとともに使用可能状態になる．したがって，ESP プログラムのニーモニック・デバッグは，PSI のファームウェア完了直後に可能になった．

ESP のファームウェア実行サポートが用意されると，これは基本呼出しメカニズムの責任になる．実行時サポートは，ファームウェアにより生成される追跡例外を処理するパッケージになる．

5.3 IPL

SIMPOS のデバッグを最初に開始したときは，デバッグ対象プログラムのコンパイルおよびリンクをメインフレーム・マシン上で完全に済ませ，ネットワークを通じてミニコンピュータに転送し，PSI へダウン・ロードしておかなければならなかった．ネットワーク転送およびダウン・ロードは非常に時間がかかった．

SIMPOS のカーネルおよびスーパバイザの一部ができた後にかなり柔軟な初期プログラム・ローダが ESP で実現された．この IPL を使うと，デバッグ対象のプログラム・モジュールは別々にコンパイル，部分リンクされ，フレキシブル・ディスクに格納される．（上田 et al. 1984）．フレキシブル・ディスクからのロードおよび最終的なリンケージは，PSI 自身の上で IPL により行なわれる．これにより，デバッギングが大幅に短縮された．

5.4 システム・トレーサ

システム・トレーサも ESP で記述されたプログラムである．このプログラムは独立プロセスとして動作し，ニーモニック形式で別のプロセスの実行を追跡する．システム・トレーサは KL0 レベルの実行追跡用に開発された．実行時サポート・システムは ESP レベルの実行だけをサポートする（佐藤 et al. 1984）．

実行時サポート・システムとシステム・トレーサの 2 つのデバッギング・ツールを統合する作業が現在進行中である．

6. 簡単な歴史

SIMPOS の設計は 1982 年の秋に ICOT で着手され，1982 年の年度末に機能仕様書が作成された．1983 年 6

月，ICOT のメンバーを除いて約 20 人のメンバーで構成されるソフトウェア・グループが結成され詳細機能設計および開発にあたった．何度かの変更後，クラス仕様書が 1983 年の年度末に完成した．

これらと並行して，ESP の要求仕様が議論され，1983 年の夏までにまとめられた．その後，ESP の言語設計および開発に着手した．ESP サポート・システムは現在，開発システム上で動作可能である．SIMPOS はクラス仕様書から ESP でコーディングされ，1983 年 10 月から ESP シミュレータでクロス・デバッグされている．

最初の PSI は 1983 年 12 月に製造され，ファームウェアによるデバッグが 1984 年 2 月末に終った．PSI は 1984 年 3 月にソフトウェア・グループに使用可能となり，その他の PSI も順次に使用可能になった．

単一プロセス環境のサポートは 4 月に可能になり，PSI 上で簡単なプログラム・デバッグができるようになった．5 月に IPL が動作可能になり，PSI 上で直接，プログラムがリンクできるようになった．6 月にサポートされたマルチ・プロセス環境の機能を使い，各サブシステムは十分にデバッグできるようになった．入出力メディア・システムの主要部は 9 月に動作可能になった．プログラミング・システムは現在 PSI 上でデバッグ中である．

SIMPOS の仮バージョンは 10 月にでき上り，SIMPOS の第 1 バージョンは今年度末に完成の予定である．

7. 結　論

ICOT，三菱電機，日本電気，沖電気，松下電器，およびシャープからの約 40 名が SIMPOS の開発に従事している．彼らの努力により，論理プログラミングの強力さおよび汎用性が明確になった．SIMPOS の現在の状態は，FGCS'84 の会場および ICOT におけるデモで見ることができる．

SIMPOS の改良と機能拡充は他の研究活動とともに続けられる．SIM はこのプロジェクトの下部構造における主要な構成要素になるはずである．

〔参 考 資 料〕

Brown, D. L., Byrd, L., Pereira, F. C. N., Pereira, L. M., Warren, D. H. D. DEC システム-10, PROLOG ユーザーズ・マニュアル. Dept. AI., エジンバラ大学, p. 101, 1983.

Chikayama, T., Takagi, S., Sakai, K. パーソナル逐次推論マシン PSI—その言語システム—. 情報処理学会第 27 回大会講演論文集 1983. ICOT Technical Memorandum TM-0022, 1983.

Chikayama, T. ESP 解説書. ICOT Technical Report TR-044, 1984a.

Chikayama, T. ESP の機能. 第 5 世代コンピュータ国際会議 1984. ICOT Technical Memorandum TM-0055,1984b.

Hattori, T., Yokoi, T. SIM オペレーティング・システムの基本構造. New Generation Computing, vol. 1 no. 1, pp. 81～85, 1983. ICOT Technical Memorandum TM-018, 1983.

Hattori, T., Tsuji, J., Yokoi, T. SIMPOS：パーソナル Prolog マシン PSI のオペレーティングシステム. ICOT Technical Report TR-055, 1984a.

Hattori, T., Yokoi, T. SIMPOS ファイル・サブシステムの概念と機能. ICOT Techaical Report TR-059, 1984b.

Hattori, T., Yokoi, T. SIMPOS スーパバイザの概念と機能. ICOT Technical Report TR-056, 1984c.

Hattori, T., Kurokawa, T., Sakai, K., Tsuji, J., Chikayama, T., Takagi, S., Yokoi, T. 逐次型推論マシン PSI のオペレーティング・システム. ICOT Technical Memorandum TM-0065, 1984d および ICOT Technical Memorandum TM-0061, 1984.

Hattori, T., Tsuji, J., Uchida, S., Yokoi, T. SIMPOS オペレーティング・システムの概要. 情報処理学会第 29 回大会, 4E-1, 1984e.

Iima, Y., Nakazawa, O., Enomoto, S., Tsuji, J. SIMPOS のウィンドウ・サブシステム. 情報処理学会第 29 回大会, 4E-6, 1984.

Kawakami, T., Ueda, N., Horie, M., Hattori, T. SIMPOS の資源管理. 情報処理学会第 29 回大会, 4E-2, 1984.

Komatsu, M., Mano, T., Konagaya, A., Hattori, T. SIMPOS のファイル・サブシステム. 情報処理学会第 29 回大会, 4E-8, 1984.

Kurokawa, T., Tojo, S. コーディネータ, パーソナル逐次型推論マシン (PSI) のカーネル. ICOT Technical Report TR-061, 1984.

Saito, S., Watanabe, H., Shimazu, H., Yoshida, N., Hattori, T. SIMPOS の実行管理—プール—. 情報処理学会第 29 回大会, 4E-4, 1984.

Sato, Y., Watanabe, H., Hori, A., Ueda, N., Chikayama, T. SIMPOS のシステム・トレーサ. 情報処理学会第 29 回大会, 4E-10, 1984.

Sawamura, H., Takeshima, S., Kato, A. PROLOG 原始レベル最適化プログラム, 最適化方法のカタログ. ICOT Technical Report TR-047, 1984.

Shimazu H., Yoshida, N., Saito, S., Watanabe, H., Hattori, T. SIMPOS の実行管理—プロセスとストリーム—. 情報処理学会第 29 回大会, 4E-3, 1984.

Takagi, S., Chikayama, T., Hattori, T., Tsuji, J., Yokoi, T., Uchida, S., Kurokawa, T., Sakai, K. SIMPOS の総合設計. 第 2 回国際論理プログラミング会議の議事録. 1984. ICOT Technical Report TR-057, 1984.

Takagi, S., Chikayama, T., Yokota, M., Hattori, T. Prolog の拡張制御構造. 情報処理学会第 26 回大会, 4D-11, 1983.

Takayama, Y., Hattori, T. SIMPOS のネットワーク・サブシステム. 情報処理学会第 29 回大会, 4E-7, 1984.

Tsuji, J., Kurokawa, T., Tojo, S., Iima, Y., Nakazawa, O., Enomoto, S. パーソナル逐次型推論マシン (PSI) における対話管理. ICOT Technical Report TR-046, 1984.

Ueda, N., Tojo, S., Kurokawa, T. SIMPOS の IPL 方式. 情報処理学会第 29 回大会, 4E-9, 1984.

Watanabe, H., Shimazu, H., Yoshida, N., Saito, S., Hattori, T. SIMPOS の実行管理ーワールドー. 情報処理学会第 29 回大会, 4E-5, 1984.

Yokoi, T., Taguchi, A., Kurokawa, T., Hattori, T., Tsuji, J., Sakai, K. パーソナル逐次型推論マシン（SIM）のオペレーティング・システムの構造と設計概念. 情報処理学会第 26 回大会, 6D-8, 1983a.

Yokoi, T., Taguchi, A., Kurokawa, T., Hattori, T., Tsuji, J., Sakai, K. 論理プログラミング言語上のオペレーティング・システムの構造. 情報処理学会第 26 回大会, 6D-7, 1983b.