マルチプロセッサシステム向けの OS/omicron タスク管理の設計と実現†

並木 美太郎†† 鈴木 茂夫††† 岡野 裕之††
堀 素史†† 横関 隆††
中川 正樹†† 高橋 延匡††

パターン認識や人工知能などの研究では，並列計算による速度向上が有効な手段として注目されている．しかし，一般的な並列アルゴリズムの研究を行うための計算機環境は貧弱である．それは，ソフトウェア，特にOSをはじめとするシステムプログラムが，並列処理を対象に設計されていないことに起因する．近年では，共有メモリ型のアーキテクチャにUNIXの環境を適合させたMachなどのOSが登場し，興味深い並列処理環境を提供している．我々は，オンライン手書き文字認識や自然言語処理を応用とし，密接な関係にある中粒度かつ多量のプロセスと大量のプロセス間通信に有利な，マトリクススイッチ型アーキテクチャの開発を目指している．マトリクススイッチ型アーキテクチャのマルチプロセッサシステム上で稼働するOSを開発するため，次の特徴を有するOS/o (omicron) タスク管理の設計・開発を行った．(1)タスクフォースと呼ばれるプログラミングモデルにより，情報を共有する密な関係にあるタスク群を効率よく定義できる．(2)ポインタ通信と呼ばれる機構により，マトリクススイッチのメモリユニットのスイッチングを記述できる．(3)ユーザ拡張部と呼ばれる層を設け，問題向けのOSの機能を動的に拡張できる．

1. はじめに

我々は，オンライン手書き文字認識[1]や，レーザビームプリンタを用いた日本語文書出力システム[2),3)]など，日本語情報処理の研究・開発を行っている．手書き文字認識などのパターン認識では，ユーザの使用環境とともに計算機の性能が実用性を大きく左右する．その解決方法の一つとして並列処理が有望視されている．

しかし，現存するOSやプログラミング環境は，必ずしも並列処理の研究に向いていない．それは，従来のOSがマルチユーザ・マルチプログラミングを主体に設計されており，単一ジョブに対するマルチプロセスの機能が，OSの提供するマシンモデルに反映されていないことによる．特殊用途のマルチプロセッサは実用化されているが，種々の並列処理アルゴリズムの研究には，様々な並列処理モデルが採用され，これらシステムが提供するマシンモデルでは柔軟性に欠ける．

近年では，Machなど UNIX ベースのマルチプロセッサ用 OS が発表されている[4)~7)]．これらはマルチスレッドのプログラミングモデルを基本としており，Ethernet による粗結合または共有メモリによる密結合システムのようなバス結合アーキテクチャを反映している．

しかし，マトリクススイッチ型のアーキテクチャのように，アドレス空間を動的に再構成する必要のあるアーキテクチャでは，上記 OS の実行環境モデルでは，一つの論理アドレス空間内に複数のプロセスを置いた時，その空間内でプロセスの移動ができないなどの問題がある．

我々は日本語情報処理と並列処理向けの OS である OS/o (omicron) の研究・開発を行ってきた[8)~10)]．最終的には，マトリクススイッチ型のマルチプロセッサシステムを構築することを目標としている[11)~14)]．OS/o は，プロトタイプ的な第1版をへて[8),12)]，現在第2版が稼働している．以下の文章では，特に断らない限り OS/o 第2版を OS/o と呼ぶ．

本論文では，

(1)タスクフォースと呼ばれる並列処理のプログラミングモデルと，タスクフォースを効率よく実現するための実行環境

† Design and Implementation of the OS/omicron Task Management Suitable for Multi-Processor Systems by MITAROU NAMIKI (Department of Information Science, Faculty of Technology, Tokyo University of Agriculture and Technology), SIGEO SUZUKI (Canon Inc.), HIROYUKI OKANO, MOTOFUMI HORI, TAKASHI YOKOZEKI, MASAKI NAKAGAWA and NOBUMASA TAKAHASHI (Department of Information Science, Faculty of Technology, Tokyo University of Agriculture and Technology).

†† 東京農工大学工学部電子情報工学科
††† キヤノン(株)

(2)研究対象向けに OS の機能を動的に拡張するユーザ拡張部

を特徴とする OS/o タスク管理の設計と実現について述べる.

2. OS/o タスク管理の設計方針

2.1 並列処理に対する要求

我々は日本語情報処理をはじめとする応用研究を行っているが, そのいずれも計算機の性能がネックとなる問題である. 次に我々の研究と並列処理の関係を示す.

(1)囲碁対局システムの研究…最良手決定

(2)自然言語処理の研究…並列構文解析

(3)オンライン手書き文字認識の研究…信号処理, 並列辞書検索

これらの研究が持つ並列処理上の特徴は, 次のとおりである.

(1)並列検索などでは, 共有データを持ち, 相互に同期をとりあうタスクが多量に発生する.

(2)いくつかの処理フェーズごとに分割できる. ただ

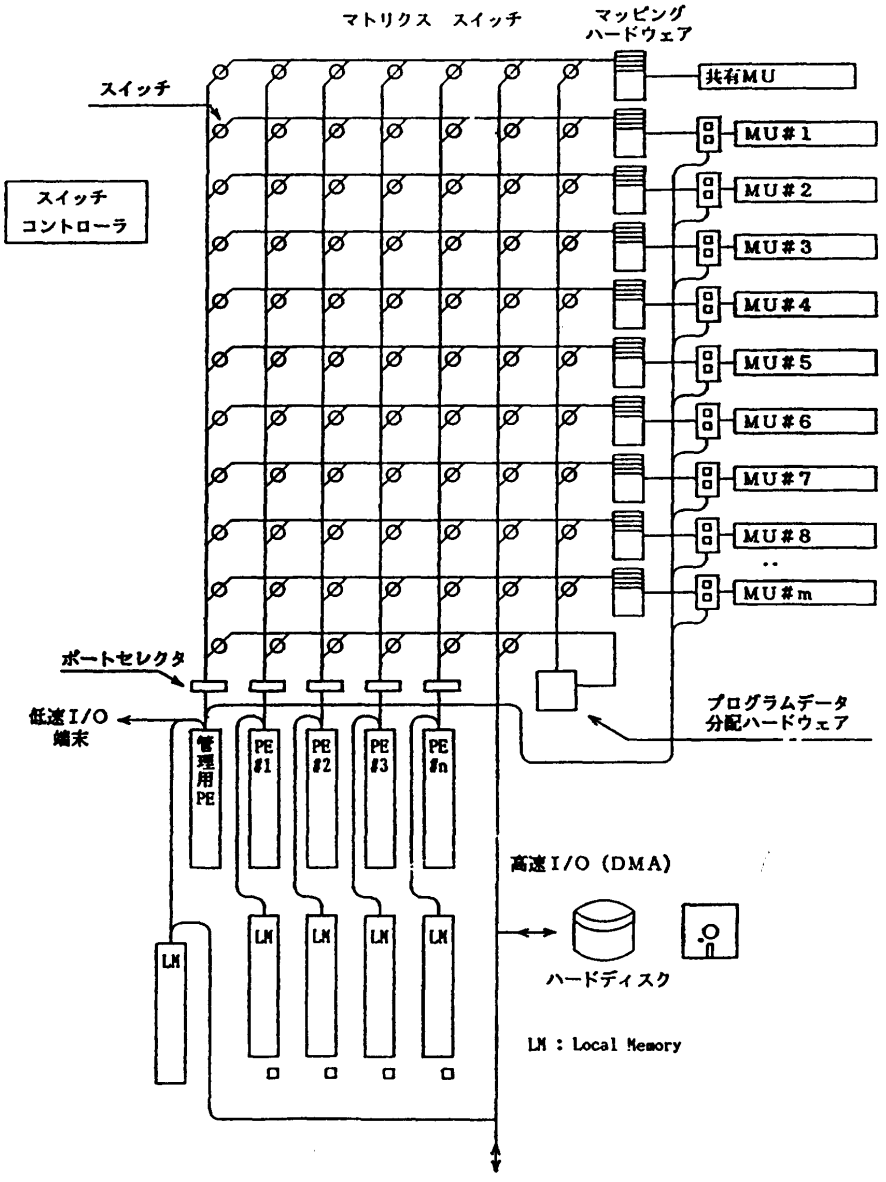

図 1 マトリクススイッチのアーキテクチャの構成例
Fig. 1 Target system (matrix-switch architecture).

し，フェーズ間で数十KBから数MBのデータ転送が行われる．

(3)上記二つの形態が同時に要求される．

(4)並列処理の粒度は，それほど小さくない．手続き型言語のサブルーチン単位程度である．

(5)種々の応用研究を行うために，OSが提供する機能に柔軟性が要求される．

2.2 マトリクススイッチ型のマルチプロセッサシステム

前節で述べた応用研究を行うためのアーキテクチャとして，**図1**に示すマトリクススイッチ型のアーキテクチャの実現を我々は計画している[13),14)]．このマトリクススイッチのアーキテクチャの特徴は，以下のとおりである．

(1)同一のデータや手続きを各MU (Memory Unit)に展開するためのバスとハードウェアを設定する．

(2)MUの一つを共有メモリにする．これを共有MUと呼ぶ．

(3)アドレス空間は実記憶方式を採用する．

実記憶系を採用した理由は，リアルタイムのアプリケーションの実行と性能設計を容易にするためである．実記憶系下で，MUの切り替え時にCPUのアドレス空間の配置に柔軟性を持たせるため，次のアドレス変換を行うマッピングハードウェアを用意する．

〈物理アドレス〉←
〈ポートセレクタから出力されるアドレス〉
＋〈変位〉

ポートセレクタとは，PEの出力するアドレスがどのMUに属しているかを決定する機構である．

入出力およびスイッチの管理は，管理用PEが行う．PE上のCPUは，設計当時もっとも広いアドレス空間を有していたMC 68000を採用した．

このようなマトリクススイッチ型のアーキテクチャには，次の利点がある．

(1)PE (Processor Element)とMUの接続を変えること（これをスイッチングと呼ぶ）で高速かつ大容量のPE間データ転送が可能となる．

(2)スイッチングにより，負荷分散のためのプロセス移動が可能となる．

(3)共有MU，または，MU間に同一のメモリイメージの分配機能を設けることにより，並列検索のような同一データに対する処理を高速化できる．図中のプログラムデータ分配ハードウェアにより，例えば，MU#2のある範囲（プログラムやデータ）をMU#5へCPUを介さずに複写することができる．

このシステムは概念設計が終了した段階である．この上で稼働するOSの設計と開発環境の実現のために，次章以降に述べるOS/oタスク管理を開発した．

2.3 タスク管理の設計方針

以上のことから，次に示す内容をOS/oタスク管理の設計方針とした．

(1)データを共有する密な関係にあるタスク群を定義でき，それらが効率よく稼働すること．

(2)MUのスイッチングを行うための機能をOSが提供すること．

(3)MUをスイッチした後に，アドレス空間を再構成できること．特に，実記憶系下でこの機能は不可欠である．

(4)種々の研究に対応するため，問題指向の機能をOSに容易に追加できること．

3. OS/o タスク管理の特徴

以上の問題と設計方針から，OS/oタスク管理に次の特徴を持たせた．

(1)タスクフォースによる並列プログラミングモデル

並列検索のように，密な関係にあるタスク群においては，高速かつ柔軟なタスク間通信が必要となる．また，このようなタスク群においては極力コンテキストスイッチのオーバヘッドを減らしたい．

そこで，データ領域など資源の一部を共有しあうタスクの集合の概念を，OS/oの並列プログラミングモデルとした[13),14)]．これをタスクフォースと呼ぶ．

(2)ポインタ通信機構によるスイッチング機能の提供

マトリクススイッチ型のマルチプロセッサでは，スイッチングによる高速・大容量のデータ転送が可能である．OSが提供するSVC (SuperVisor Call)で，MUのスイッチングを用いたタスク間通信を実現できなければならない．

OS/oタスク管理では，データ領域の論理アドレスを転送することにより，スイッチングに対応する機能を実現した．これをポインタ通信と呼ぶ．

(3)リエントラント・動的にリロケータブルな実行環境

マトリクススイッチ型のアーキテクチャでは，MUのスイッチングにより，動的にアドレス空間を再構成する必要性が生じる．そこで，OS/oのタスクの実行

環境は，動的にリロケータブルな実行環境とした．

(4)OS の動的な機能拡張が可能

マンマシンインタフェースの研究を行う場合，例えば透明タブレットのような新しいデバイスを接続したい要求が生じる．また，種々の応用研究を行うと，問題向けの OS の機能を追加したい．通常の OS では，デバイスハンドラを記述した後，OS 核の再リンクなどの作業が必要となる．

そこで，OS/o ではユーザ拡張部と呼ばれる層を設け，ユーザプログラムから OS の機能を動的に拡張する機構を実現した．

OS/o は，シングルプロセッサ上で稼働しているが，設計方針，プログラミングモデルとコンパイラはマトリクススイッチ型のマルチプロセッサを対象とし，シングルプロセッサとマルチプロセッサシステム上で同じプログラミングモデルと実行環境を持たせている．これにより，シングル／マルチプロセッサ上での統一性を保証している．

4. タスクとタスクフォース

4.1 タスクの実行環境

マトリクススイッチのアーキテクチャでは，スイッチングによって MU と PE の接続が変わる時に，MU を PE のアドレス空間のどこへ接続するかが問題になる．MU のアドレスを固定にする方法が簡単だが，接続方法に柔軟性がなくなる．我々のシステムでは，スイッチングと同時に，接続するアドレスを指定できるハードウェアを考えている（図1中のマッピングハードウェア）．スイッチングによる，CPU のアドレス空間の再構成の例を図 2 に示す．

このようなシステムでは，プログラムがどこへ配置されるかわからない．しかも，動的なリロケーションが手続き・データに対して起こりうる．

そこで，OS/o 上で稼働するプログラムは，リエントラント・動的にリロケータブルな実行環境を持つようにした．プログラムの実行の最小単位をタスクと呼ぶ．並列処理の粒度は，このタスクであり，手続き型言語のサブルーチン単位程度の大きさである．

リロケータビリティは，アドレシングをベースレジスタからの相対変位で行うことにより実現した．つまり，

〈CPU のアドレス空間〉←

〈対象に対する変位〉+〈ベースアドレス〉

によって手続きやデータのアクセスを行う．

タスクは以下の実行環境を有する．

(1)主記憶上の 4 領域

タスクは主記憶上に，手続き領域，静的データ領域，ヒープ領域，スタック領域の 4 領域を持つ．これら各領域へのアドレシングは，各領域のベースレジスタからの変位となっており（図 3），さらに各領域に対する共通な変位を与えるため，TFBR (Task Force Base Register) を設定した．言語 C のポインタは，TFBR からの変位となっている．このベースレジスタについては，ポインタ通信の項で述べる．

これらのベースレジスタにより，リエンタラビリティと動的なリロケータビリティを保証している．OS/o は MC 68000 を CPU として用いており，ベースレジスタは MC 68000 のアドレスレジスタ(A 1 から A 6) を利用し，レジスタ間接のアドレシングモードを用いてアクセスを行う．

(2)プロセッサのレジスタ

(3)セマフォ，メッセージ

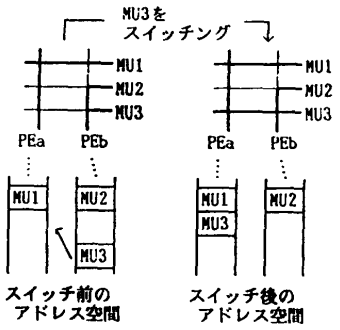

図 2 MU のスイッチとアドレス空間の再構成
Fig. 2 Re-configuring address space in switching memory units.

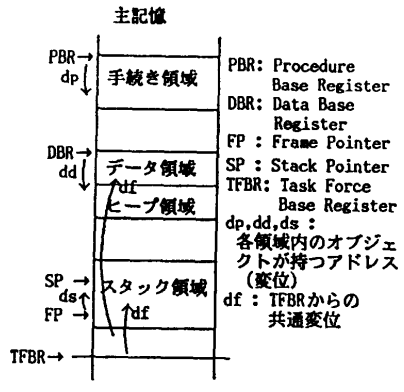

図 3 タスクの動的な実行環境
Fig. 3 The relocatable executing environment of tasks.

(4)カレントディレクトリ，オープンされたファイル

以上の四つの資源がタスクのコンテキストである．この実行環境に対するオブジェクトコードは，言語C処理系 CAT により標準的に生成される[15]．

4.2 タスクフォース

タスクは基本的には，個々の実行環境を持ち，独立して実行を進める．しかし，目的とする仕事によっては，複数のタスクがデータを共有したり，同期をとりながら処理を進める形態が考えられる．データの共有や通信機能は，OS の SVC として実現されるが，OS の呼出しにはコンテキストスイッチングが伴うため，呼出し回数が多いほど処理効率に大きな影響を与える．

そこで OS/*o* では，複数のタスクが共同して一つの仕事を行う形態を効率よく実現するために，実行環境の一部を共有するタスクの集合の概念を導入した．これをタスクフォースと呼ぶ[13),14)]．

タスクフォースは，タスクの次のコンテキストを共有するタスクの集合である（図 4）．

(1)手続き領域，静的データ領域，ヒープ領域
(2)カレントディレクトリ，オープンされたファイル

物理的にどのプロセッサに割り当てられるかは重要なことではない．データや情報の共有を通して強く関係付けられたタスクの集合が，タスクフォースである．

したがって，静的データ領域を共有データ領域とし，同期・排他制御にセマフォを利用することにより，柔軟かつ高速なデータ交換が可能となる．また，タスクのコンテキストスイッチのオーバヘッドが減少するのが利点である．

これらの各領域は，動的にリロケータブルなため，どのアドレスに配置してもよい．共有される領域はベースレジスタを同じ値にする．リエントラントかつ動的にリロケータブルなので，プロセス移動の際に，例えば手続き領域を異なる MU の別アドレスに複写してタスクフォース内のタスクを実行してもよい．

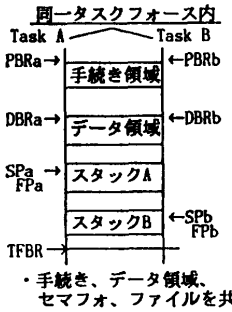

図 4 タスクフォース内のタスク
Fig. 4 Tasks in a taskforce.

情報を共有したスレッドや軽量化プロセスという概念がいくつかの OS で導入されているが[4)~7)]，我々のタスクフォースは，それらより早い時期の 1983 年にその概念を発表し[12),13)]，OS/*o* 第1版が翌年に完成している[8)]．

また，我々のタスクフォースがこれら軽量化プロセスなどと根本的に異なる点は，実行環境が動的にリロケータブルな点である．仮想空間の各ページを実記憶のどこへ写像するかは，メモリ管理機構の責任である．しかし，リロケータブルな実行環境でないと，一つの仮想空間内に二つのプロセスを配置した後では，各プロセスの位置を移動できない．つまり，仮想空間内のあるアドレスに固定されてしまい，プロセスを他の仮想空間へ移動する場合，ある特定のアドレスにしか配置できない．まして，同一仮想空間内でのプロセスの移動は不可能である．固定されたアドレス空間上でしか実行できないのでは，複数の MU からアドレス空間を構成し，それが動的に変化するマトリクススイッチのアーキテクチャを生かせない．

我々のタスクフォースは，アドレス空間内でどこへリロケーションしてもかまわない．リロケーションは，メモリの内容を複写またはスイッチの切り替えとベースレジスタの値を変更するだけで，特殊なハードウェアは不要である．マルチプロセッサシステムでのプロセス移動も，ロードアドレスに独立して行うことができる．

4.3 タスクフォースの結合

タスクフォースの定義から，異なるタスクフォースでは共有するデータを持たない．各領域に対するベースレジスタが互いに異なるために，アクセスできない．

しかし，各処理フェーズごとにタスクフォースを定義し，そのタスクフォース間でデータの共有ができると有用である．

OS/*o* タスク管理では，親子間のタスクフォースで，生成したタスクフォースと生成されたタスクフォースの間でデータを共有するか否かを，タスクフォースの生成期に指定できる．タスクフォース生成の SVC である __create_tf という SVC の引数は，

(1)データ領域を共有するか．
(2)オープンされているファイルやセマフォを継承す

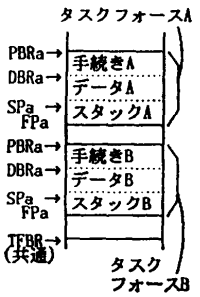

図 5 結合されたタスクフォース
Fig. 5 Binded taskforces.

るか.

を与える仕様となっている.

データ領域を共有する場合は，**図 5** に示すようにTFBR を一致させる．これにより，ポインタ値（アドレス）が相互に等価になり，相互の領域のアドレシングが可能となる．

TFBR を等価にすることから，これをタスクフォースの結合と呼ぶ．タスクフォースの結合という概念が重要な点は，二つのタスクフォースの間で共通のポインタ値を持たせることである．したがって，必ずしも共通の TFBR にする必要性はないが，OS やプログラミングモデルに制約を課することになる．これについては，ポインタ通信の項で述べる．

5. タスク間通信機構

各タスクは，独立で存在するのではなく，他のタスクとの同期・通信が必要となる．OS/*o* では，協調しあうタスクの集合の概念であるタスクフォースの内外で，通信に対する要求が異なる．タスク間通信の設計方針は，次のとおりである．

(1)タスクフォース内のタスク間では，高速なタスク間通信を実現する．

(2)異なるタスクフォース間では，パイプライン的な処理に対して高速なタスクフォース間通信を実現する必要がある．特にマトリクススイッチのスイッチングを OS の機能として実現できなければならない．

(3)同時にマトリクススイッチから粗結合型のシステムまで統一のとれたタスクフォース間通信を実現する．

このことから，OS/*o* では次のタスク間通信機構を用意した．

(1)タスクフォース内のタスク間では共有データによる通信を行い，排他制御はセマフォを用いる．

(2)タスクフォース間では，メッセージ通信とポインタ通信を用意した．特にポインタ通信により，OS はマトリクススイッチの MU のスイッチングを提供することが可能となり，粗結合システムまで統一したプログラミングモデルを実現している．

5.1 タスクフォース内のタスク間通信とセマフォ

タスクフォースは静的データ領域を共有し，高速な情報交換が可能である．ただし，何らかの手段で共有領域のアクセスを制御しなくてはならない．共有領域をアクセスする際の排他制御手段として，PV セマフォを用意した．

OS 核は，セマフォ作成の __create_sem SVC, PV 命令に対応する __P_op, __V_op SVC, およびセマフォを削除する __delete_sem SVC を提供する．

5.2 タスクフォース外のタスク間通信

5.2.1 メッセージ通信

タスクフォース内でのタスク間と異なり，別タスクフォースに属するタスク間では，基本的には共有のデータ領域を持たない．これらのタスク間での通信機能の一つとして，OS/*o* では，ランデブ型のメッセージ通信機能を実現した．

メッセージ通信経路は，通信相手（タスクフォース識別子），通信モード（送信／受信）を引数とする __create_mssg SVC によって生成される．

通信を行う二つのタスクフォースが，お互いに対応したモードで __create_mssg を呼び出した場合に，メッセージの経路が結合される．

__create_mssg SVC によりメッセージ通信経路が結ばれると，戻り値としてメッセージ識別子が返される．このメッセージ識別子により，メッセージの送受信を行う機能が __send_mssg, __receive_mssg SVC である．

5.2.2 ポインタ通信

例えば，二つのタスクフォースA，Bがあり，この間でのパイプライン処理を考える．二つのタスクフォース間を，前項のメッセージ通信により，パイプライン化する方法が考えられる．しかし，数 MB におよぶデータ転送の場合，メッセージ通信では複写や SVC のオーバヘッドが増加する．我々が目標としているアーキテクチャはマトリクススイッチであり，パイプライン的なタスクフォース間通信を MU のスイッチングで高速に行う．したがって，データの書かれた MU のアドレスを後段のタスクフォースへ転送し，スイッチングを行えばよい．詳細な手順は次のとおり

である.

(1)送信側タスクフォースと受信側タスクフォースを結合して生成する.

(2)送信側タスクフォースで転送すべき MU のアドレスを,TFBR の変位として求める.これを MU のユニットアドレスと呼ぶ.

(3)以下の手順のいずれか,または組合せにより受信側タスクフォースへユニットアドレスを転送する.

1)受信側タスクフォースの生成時に引数として受けわたす.

2)メッセージ通信で転送する.

3)共有メモリ領域で通知する.

(4)SVC により MU のスイッチングを行う.

(5)受信側タスクフォースでは,転送された MU のユニットアドレスを用いてデータの参照を行うことができる.なぜなら,タスクフォースの結合により送受信タスクフォースで,共通のアドレスが与えられているからである(**図 6**).また,送信側で MU 中にリスト構造を格納し,受信側でそのリストをアクセスすることもできる.なぜならば,結合されていることにより言語Cのポインタ(TFBR からの変位)は,共通の値を持つためである.

このように,MU のユニットアドレスを結合された受信側のタスクフォースへ転送し,ユニットアドレスを用いてデータをアクセスすることを,ポインタ通信と呼ぶ.ポインタ通信により,マトリクススイッチの MU のスイッチングを OS の機能として実現できる.

もし,スイッチングの際,ユニットアドレス上に別のタスクフォースが実行されている(例えば図6中のスイッチング先に別のタスクフォースが存在している)時は,リロケーションを行った後に空間を割り当てる.ユニットアドレス上にあったタスクフォースをリロケーションしてもよいし受信側タスクフォースを TFBR とともにリロケーションしてもよい.どちらをリロケーションするかは,OS のメモリ管理に依存する.

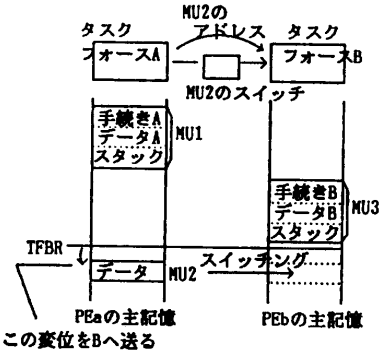

図 6 MU のスイッチングとポインタ通信
Fig. 6 Switching of memory units and pointer communication.

なお,結合されていないタスクフォース間でも,ポインタ通信は不可能ではない.スイッチングされた MU を受信側の TFBR を基底とするユニットアドレスに配置すればよい.ただし,結合されていないタスクフォースを同一のアドレス空間上に配置すると,共通のユニットアドレスを持つことができないほか,MU 中に格納されたポインタが変化してしまい,正常なアクセスができなくなる.結合されていないタスクフォース間でのポインタ通信は,

(1)同一空間上に送受信タスクフォースを配置できない.

(2)MU 中にポインタを格納できない.

などスケジューリングやプログラミングが制限されてしまう.

ポインタ通信はシングルプロセッサシステムでも効率的な方法であり,粗結合のシステムでも同一の通信モデルを提供できる.シングルプロセッサシステムでは,マトリクススイッチの MU のスイッチングがないと考えればよい.この場合,メッセージ通信と比較するとデータの複写がない点が有利である.粗結合システムにおいては,メッセージ通信によりデータの転送を行い,バッファのアドレスをユニットアドレスとして受信側タスクフォースへ転送すればよい.

以上に述べたように,ポインタ通信方式により,マトリクススイッチングアーキテクチャの MU のスイッチングを OS が提供できるだけでなく,種々のアーキテクチャに対して統一的なタスクフォース間通信を実現することができた.

基本的には,ポインタ通信は,マトリクススイッチに依存しないタスク間通信機構である.データを複写して転送するのではなく,データの格納されている場所(アドレス)とアクセス権を転送するというモデルである.マトリクススイッチの抽象化機構となっている.

なお,ポインタ通信の機構は,シングルプロセッサ上の版ではタスクフォース生成の際のタスクフォース結合とメモリプールからメモリを得るための SVC を用意し,MU のスイッチに相当する機能は,次の二つのライブラリとして実験的に実現した.

(1)BYTE *__get_mu(サイズ)
MU を得る.
(2)__send_mu(ユニットアドレス, サイズ)
送信側タスクフォースで MU を転送する.
(3)BYTE *__receive_mu (サイズを得る.)
受信側タスクフォースで MU のユニットアドレスとサイズを得る.

send/receive 系の SVC でアプリケーションプログラム側でバッファを用意する記述方式に対し, この方式はスイッチされた MU のユニットアドレスとメモリに対するアクセス権がわたされる方式である. マトリクススイッチアーキテクチャ上に実現する際は, このライブラリに相当する機能を SVC として提供する.

上記方式では, アプリケーションプログラムが MU のスイッチングを明示的に指定した. これとは別に, PE が MU に該当するアドレスをアクセスするたびにスイッチングを自動的に行うという方式が考えられる. これならば, 通常の共有メモリと同じスタイルでプログラミングできる. ただし, この方式でもメモリ保護の観点から, スイッチングと同時に MU へのアクセス権(接続する権利)を管理する必要がある. MU に対する接続権が定義可能でアクセスに応じて自動的にスイッチングを行うハードウェアと, 同一 MU に対するアクセスを減らすためのメモリ管理などを実現すれば有効な方式であると考えられる.

6. ユーザ拡張部

システムユーザが種々のシステムを構築する際, 応用研究に応じた機能を OS の核に追加したいことがある. 従来の OS では, OS が開放されていない場合が多く, OS の核に変更を加えるのが困難である. たとえ, 可能だとしても, これらの機能をコーディング, コンパイルした後に, 核を再リンクしなくてはならない.

各種応用研究に対応したシステムを構築したり, 種々の特殊なデバイスを容易に接続できるために, 我々は次の機能が OS に必要だと考えた.

(1)OS 核の再リンクなしに OS の機能を追加できること.
(2)システムユーザから, 機能の登録/削除を動的に行えること.
(3)ハンドラや例外処理を記述するために, 割込みを記述できること.
(4)種々の資源を操作するため, OS と同じ特権モードで追加されたプログラムが稼働すること.
(5)OS核と同様の呼出し手順で使用できること.

上記の要求を実現するため, OS/o では OS 核とユーザアプリケーション層の間にユーザ拡張部と呼ばれる層を導入した (図 7).

ユーザ拡張部は, ユーザプログラムから OS 核の __set_ue_task, __reset_ue_task SVC により, OS 核を再リンクすることなく, 動的に登録や削除を行うことができる.

こうして登録されたユーザ拡張部は, 応用プログラムからあたかも核であるかのごとく, SVC で呼び出される. OS 核の呼出しは MC 68000 の trap #4, trap #5, trap #7 命令で, ユーザ拡張部は trap #8 命令で行われる. 拡張部の呼出しは, 次の三つのパラメータを伴う.

(1)拡張部の番号
(2)拡張部の機能番号
(3)機能番号に対応する引数

拡張部の番号は, 拡張部で提供する機能の集合に与えられる番号である. 機能番号と引数が OS 核の SVC に匹敵する.

ユーザ拡張部へ登録するプログラムは, 通常のロードモジュールであり, 拡張部のための特異なプログラミングは不要である. さらに, 拡張部中で核の機能を制約なしに使用できる. ユーザプログラム, 拡張部, OS 核の処理の流れを図 8 に示す.

ユーザ拡張部は, 通常のユーザプログラミングと次

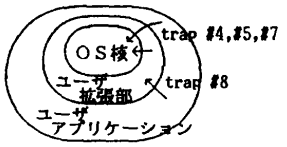

図 7 OS/o の構成とユーザ拡張部
Fig. 7 The structure of OS/o.

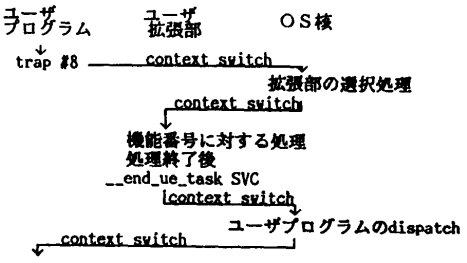

図 8 ユーザ拡張部の処理の流れと context switch
Fig. 8 The control flow calling user extend layer.

の点が異なっている．

(1)OS 核と同様に特権モードが実行される．

(2)デバイス割込みのような外部割込みを処理できる．

7. OS 核の実現

以上に述べた特徴を持つタスク管理を実現した．実現に関する方針は次のとおりである．

(1)マルチプロセッサシステムでの稼働を考慮し，OS もマルチスレッドの構造とする．

(2)移植性・保守性を考慮し，言語 C で記述する．

(3)割込み処理を特別に管理せず，タスクによる統一的な処理を行う．

OS/o の OS 核の規模は，現在，言語Cで記述した部分が約 25,000 行である．そのうち，タスク管理は言語Cで約 5,000 行，さらにアセンブリ言語で記述された部分（主にディスパッチャ）が約500行である．

7.1 OS 核の実現

OS/o では，OS 核も通常のユーザタスクと同様に，図3のような実行環境を持ち，それらの領域へのアクセスはベースレジスタを介して行われる．OS 核自身もタスクフォースの形態を持ち，割込み処理などもタスクとして管理される．OS 核の実行環境は，ユーザタスクと同様に TCB (Task Control Block) により管理される．

マルチタスクの環境下では，ユーザタスクが並列に OS 核を呼び出すことを考慮し，OS 核をすべてのユーザタスクから同時に利用できるタスク（これをカーネルタスクと呼ぶ）として扱う．実現方法としては，カーネルタスクの実行環境，およびそれを管理する TCB を，その時点で存在するすべてのユーザタスクの数だけ用意し，それらを**図9**に示すように，ユーザタスクの TCB 中のポインタにより管理する．そして，ユーザタスクが SVC により OS 核を呼び出した場合は，このカーネルタスクを対象として，ディスパッチャがプロセッサを割り当てる．なお，OS 核の表には，相互排除がなされている．

このような OS の構造により，OS 核は完全に並列動作が可能となっている．

7.2 割込み処理

OS/o では，割込みもタスクとして統一的に管理する．割込みを処理するタスクを割込み処理タスクと呼ぶ．割込み処理タスクは，OS 核とユーザ拡張部に存在する．OS 核には，ハードディスク装置，フロッピディスク装置などを対象とした割込み処理タスクが存在する．これらの割込み処理タスクは，OS の立ち上げ時 OS 核内で生成，実行される．生成された割込み処理タスクは，まず処理対象の外部割込みの確保を行う．これにより，指定されたタスクが外部割込み管理表に登録され，割込み処理タスクは各初期化を行った後，外部割込みの発生を待つ．外部割込みが発生すると，OS 核は登録されている割込み処理タスクをディスパッチする．ディスパッチされた割込み処理タスクは，I/O 領域とバッファのデータ転送を行う．この時，セマフォを用いて，バッファからデータを取り出すタスク（ユーザタスクから呼び出されたカーネルタスク）と同期をとる．

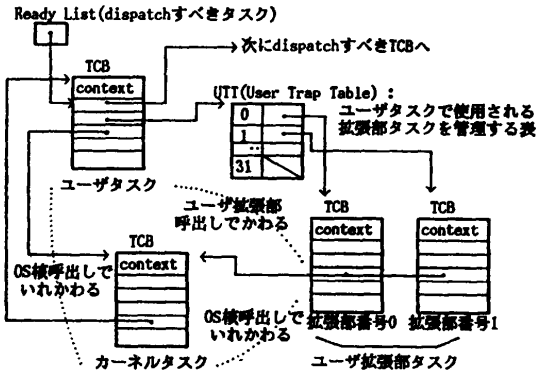

図 9 ユーザタスク，カーネルタスク，ユーザ拡張部タスクの関係
Fig. 9 The structure of task control blocks.

割込み処理のモデルは，OS 核とユーザ拡張部で共通になっている．ただし，外部割込み表の登録や割込み発生待ちは，OS 核内のハンドラでは関数呼出しで，ユーザ拡張部では SVC によって行われる．

7.3 ユーザ拡張部の実現

OS/o では，ユーザ拡張部に登録された各処理プログラムも，OS 核と同様にタスクとして扱う．これをユーザ拡張部タスクと呼ぶ．ユーザ拡張部タスクの TCB は，OS 核の場合と同様に，各ユーザタスクごとに存在し，それぞれが UTT (User Trap Table) により管理される (図9参照)．ユーザタスクが拡張部を呼び出した時，OS 核は UTT により対応する拡張部タスクをディスパッチする．さらに，その拡張部中で OS 核を呼び出した時は，カーネルタスクをディスパッチする．なお，拡張部タスクやカーネルタスクのスタック領域は，ユーザタスクのスタック領域を継承する．

8. OS/o 第2版タスク管理の性能解析

OS では，タスクのコンテキストスイッチの実行効率がシステムの性能を左右する．OS/o では，ユーザ拡張部，そして OS 核もタスクとして管理されているので，コンテキストスイッチの性能が重要な問題となる．

そこで，タスクのコンテキストスイッチに要する実行時間を測定した．対象システムは，日立 2050/32 システム (MC 68020 : 20 MHz) 上で動作する OS/o システムである．2050/32 システムには，OS の開発および移行を目的として，複数の OS を同時に動作させる多重 OS「江戸」[16] が実現されている．「江戸」上で動作する OS/o を対象として，一回の SVC に要するコンテキストスイッチングの実行時間を，多重 OS「江戸」のタイマを用いて測定した．

測定結果を**表1**に示す．OS 核のコンテキストスイッチングの処理時間は約 140 μs，ユーザ拡張部の呼出し処理に伴うコンテキストスイッチングの時間は，OS 核の呼出しの約3倍にあたる 410 μs という結果を得た．これは，ユーザ拡張部の処理が，2回コンテキストスイッチが発生することと，拡張部に対する表の管理などの操作が増加するためである．

ユーザ拡張部を経由した時，約 400 μs とコンテキストスイッチに関するオーバヘッドは大きい．しかし，ユーザ拡張部により容易に応用向けの OS 核を構築できる．性能が要求される場合は，ユーザ拡張部によりデバッグした後，核に組み込むことによりコンテキストスイッチのオーバヘッドを減らす方法が考えられる．

本タスク管理は，動的なリロケータビリティの実現のために，MC 68000 の CPU アーキテクチャをいかしたプログラミングを行っていない．また，言語Cで記述していることなどを考えると，OS 核のコンテキストスイッチに要する 140 μs は妥当なオーバヘッドだと考える．高速化に関しては，全面的にアセンブリ言語で記述する方法が考えられるほか，OS/o の実行環境を考慮したコンテキストスイッチ命令を有する CPU アーキテクチャを考察する必要がある．

表 1 コンテキストスイッチの処理時間
Table 1 Performance of OS/o kernel.

処理内容	処理時間 (μs)
OS 核の呼出し	140
ユーザ拡張部の呼出し	410
ユーザ拡張部経由	
P 命令	720
V 命令	690
OS 核の直接呼出し	
P 命令	310
V 命令	280

9. おわりに

本論文では，マルチプロセッサ，特にマトリクススイッチ型のアーキテクチャを考慮したタスク管理の設計と実現について述べた．本研究・開発の成果は，以下のとおりである．

(1) タスクフォースと呼ばれる並列処理モデルにより，データを共有しあうタスク群を定義できるようになった．

(2) 転送すべきデータのアドレスを転送するポインタ通信の機構により，マトリクススイッチの MU のスイッチングを記述できるようになっただけでなく，粗結合のネットワークシステムを含む，統一したタスク間通信機構を実現した．

(3) リエントラントかつ動的にリロケータブルなタスクの実行環境を設定したことにより，マトリクススイッチシステムの MU ユニットのスイッチング時のアドレス空間の再構成が容易になった．

(4) ユーザ拡張部により，問題向けの機能をユーザプログラムから動的に OS に追加できるようになった．

OS/o は，シングルプロセッサシステム上で稼働しており，プログラム開発に使用されている．現在，共有メモリ型のマルチプロセッサシステムへ移行中であるが，この共有メモリ型システムでの経験をふまえて，マトリクススイッチ型マルチプロセッサシステム版の OS/o を構築する予定である．

参考文献

1) Nakagawa, M., Aoki, K., Manabe, T., Kimura, S. and Takahashi, N.: On-line Recognition of Handwritten Japanese Characters in JOLIS-1, *Proc. of the 6th ICPR*, pp. 776-779 (1982).

2) Takahashi, N., Atoda, O., Manabe, T., Ikeda, Y., Itoh, Y. and Nakagawa, M.: 浄書 : Japanese Output Server with HOspitality, *Proc. of ICTP*, pp. 29-34 (1983).

3) 里山元章，中川正樹，高橋延匡：文書の論理構造を備えた日本語清書システム「浄書」の設計と実現，情報処理学会論文誌，Vol. 30, No. 9, pp. 1126-1134 (1989).

4) Tevanian, A.: Architecture-Independent Virtual Memory Management for Parallel and Distributed Environment: The Mach Approach, CMU-CS-88-106 (1987).
5) Cooper, E. C. and Draves, R. P.: C Threads, CMU-CS-88-154 (1988).
6) Ousterhout, J. E., Cherenson, A. R., Dougils, F., Nelson, N. and Welch, B. B.: The Sprite Network Operating System, *Computer*, Vol. 21, No. 2, pp. 23-36 (1988).
7) System Service Overview, Sun Microsystems (1988).
8) 高橋延匡，並木美太郎，武山潤一郎，中川正樹：OS/*o*のアーキテクチャと第1版の実現，情報処理学会 OS 研究会資料，24-11，pp. 65-70 (1984).
9) 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：OS/omicron における日本語プログラミング環境の実現，情報処理学会論文誌，Vol. 30, No. 1, pp. 2-11 (1989).
10) 鈴木茂夫，田中泰夫，岡野裕之，堀 素史，並木美太郎，高橋延匡：OS/omicron 第2版の実現と評価，情報処理学会 OS 研究会資料，42-1 (1989).
11) 武部桂史，鵜澤繁行，中川正樹，阿刀田央一，高橋延匡：MC 68000 アドレス空間の問題点と構成方式，情報処理学会 MC 研究会資料，19-1 (1981).
12) 高橋延匡，武山潤一郎，並木美太郎，中川正樹：MC 68000 用小型 OS：OS/omicron の開発，情報処理学会 OS 研究会資料，21-6 (1983).
13) 中川正樹，篠田佳博，藤森英明，高橋延匡：MC 68000 ユニ & マルチ・プロセッサ・システム用システム記述言語C処理系の開発，情報処理学会 OS 研究会資料，21-7 (1983).
14) 並木美太郎，高橋延匡：OS/omicron のマルチプロセッサ化の構想，信学 CPSY 研資，88-38，pp. 1-6 (1988).
15) 並木美太郎，屋代 寛，田中泰夫，篠田佳博，藤森英明，中川正樹，高橋延匡：OS/omicron 用システム記述言語C処理系 Cat のソフトウェア工学的見地からの方式設計，信学論（D），Vol. J 71-D, No. 4, pp. 652-660 (1988).
16) 岡野裕之，堀 素史，中川正樹，高橋延匡：多重 OS「江戸」設計と実現，情報処理学会論文誌，Vol. 30, No. 8, pp. 1012-1022 (1989).

(平成元年8月30日受付)
(平成2年4月17日採録)

並木美太郎（正会員）

昭和 59 年東京農工大学工学部数理情報卒業．昭和 61 年同大学院修士課程修了．同4月(株)日立製作所基礎研究所入社．昭和 63 年より東京農工大学工学部数理情報助手．平成元年4月より電子情報助手．並列処理，日本語情報処理のソフトウェア／ハードウェアアーキテクチャに興味を持ち，コンパイラ，オペレーティングシステムなどシステムプログラムの研究・開発に従事する．

鈴木 茂夫（正会員）

昭和 62 年東京農工大学工学部数理情報卒業．平成元年同大学院修士課程修了．同4月(株)キヤノンに入社．在学中，オペレーティングシステムのタスク管理の研究に従事．

岡野 裕之（正会員）

昭和 63 年東京農工大学工学部数理情報卒業．平成2年同大学院修士課程修了．同4月(株)日本 IBM に入社．在学中．仮想マシン，マルチプロセッサ用オペレーティングシステムの研究に従事．

堀 素史（正会員）

昭和 63 年東京農工大学工学部数理情報卒業．平成2年同大学院修士課程修了．同4月(株)富士ゼロックスに入社．在学中，仮想マシン，ウィンドウシステムの研究に従事．

横関 隆（正会員）

昭和 63 年東京農工大学工学部数理情報卒業．平成2年同大学院修士課程修了．同4月(株)ソニーに入社．在学中，追記型光ディスクを用いた世代管理ファイルシステム，マルチプロセッサ用オペレーティングシステムの研究に従事．

中川 正樹（正会員）

昭和 52 年東京大学理学部物理卒業．昭和 54 年同大学院修士課程修了．同在学中，英国 Essex 大学留学 (M. Sc. in Computer Studies)．昭和 54 年東京農工大学工学部数理情報助手，平成元年 1 月数理情報助教授，同 4 月電子情報助教授．オンライン手書き文字認識，日本語計算機システム，文書処理の研究に従事．理学博士．

高橋 延匡（正会員）

昭和 8 年生．昭和 32 年早稲田大学第一理工学部数学卒業．同年(株)日立製作所中央研究所入社．HITAC 5020 モニタ，TSS の開発に従事．昭和 52 年より東京農工大学工学部数理情報教授．平成元年電子情報教授．オペレーティングシステム，日本語情報処理，パターン認識の研究に従事．電子情報通信学会，ソフトウェア科学会，計量国語学会，ACM 各会員．理学博士．

知識型設計方法論に基づくインタフェース設計法の形式化と設計支援システムの構成†

木下哲男†† 岩根典之††
菅原研次††† 白鳥則郎††††

高度情報処理システムを実現するための重要な構成要素として，ヒューマンインタフェースが挙げられる．一般に，優れたヒューマンインタフェースを設計できるのは，豊富な経験を持つ熟練した設計者が設計を行った場合に限られることが多い．これは，利用者の要求をヒューマンインタフェースに反映させるための効果的な手段，あるいは熟練した設計者の持つ様々な設計知識を有効に利用するための手段が与えられていないことによる．このような問題を解決するために，本論文では，利用者の要求や熟練した設計者の設計知識を効果的に活用する知識型設計方法論の枠組みに基づいて，ヒューマンインタフェース設計のための知識モデル，およびそれを用いたヒューマンインタフェース設計方式を提案する．本方式では，ヒューマンインタフェース設計過程をシステムの機能設計過程とは独立にモデル化する．そして，その設計過程に含まれる各設計ステージが，利用者の要求や設計仕様などの知識で与えられる知識モデルとして表現され，また，各設計ステージを連結する設計プロセスが，熟練した設計者の持つ種々の設計知識で与えられる知識モデル間のマッピングとして表現される．こうして，要求仕様定義からヒューマンインタフェースの実現に至る設計過程が，マッピングを用いた知識モデルのインタラクティブな変換系列としてモデル化される．このようなモデル化によって，利用者の要求が効果的にヒューマンインタフェースに反映され，同時に熟練した設計者の知識を有効に活用した設計過程によりヒューマンインタフェースが半自動的に生成できる．さらに，本論文では，本方式に基づいて試作されたヒューマンインタフェース設計支援システムと設計例について述べ，その有効性を示す．

1. はじめに

コンピュータシステムの提供する機能とシステムの利用者の境界面となっているヒューマンインタフェース（以下，インタフェースと略記する）は，高度情報処理システムを実現するための重要な要素として認識されているが，優れたインタフェースを設計することは，次のような理由から極めて難しい問題となっている[12]．すなわち，従来のインタフェース設計は，目的とするシステムの機能の一部としてインタフェースの設計仕様が定義され，システムが提供する機能との密接な関連のもとに設計が行われてきた．これは，システムの機能設計を主体とするインタフェース構成手法であり，設計目標となるインタフェースに対して利用者の真の要求を反映させることが困難となる[2)~8)]．一方，近年，インタフェースを構成する部品要素とその直接的な操作機能を，利用者，あるいは設計者に提供することによってインタフェースを構築する手法が提唱されている[9)~11)]．これは，利用者の視点を重視してインタフェースを設計する一手法となっているが，種種の要求に応じて多数の部品を組み合わせるための方法論が確立されておらず，また利用者自身が設計を進める際の設計知識なども明示的に与えられていない．上述した2つの設計手法は，それぞれ異なる立場からインタフェースの構築を目指すものである．しかし，いずれの場合も，現状では，利用者の持つ様々な要求を，設計仕様，およびそれに基づいて実現されたインタフェースに十分反映させるための効果的な手段，また，熟練した設計者の経験に基づく設計知識を有効に利用する手段などがほとんど与えられていない．そのため，優れたインタフェースが構築できるのは，熟練した設計者が設計を行った場合に限られることが多い．したがって，インタフェース設計の高度化のための課題は，1)利用者の要求を設計目標となるインタフェースに効果的に反映させるための手段，および，2)熟練した設計者の持つ高度な設計知識を有効に利用するための手段，をどのように与えるか，という問題に帰着される．

インタフェース設計に関する上記の問題を解決するために，本論文では，知識型設計方法論と呼ぶ枠組み

† Formalization of the Interface Design Method and Construction of the Design Support System Based on the Knowledge-based Design Methodology by TETSUO KINOSHITA, NORIYUKI IWANE (Systems Laboratory, OKI Electric Industry Co., Ltd.), KENJI SUGAWARA (Faculty of Information Engineering, Chiba Institute of Technology) and NORIO SHIRATORI (Research Institute of Electrical Communication, Tohoku University).

†† 沖電気工業(株)総合システム研究所
††† 千葉工業大学情報工学科
†††† 東北大学電気通信研究所

に基づいたインタフェースの設計支援方式を提案する．本方式では，インタフェース設計のための知識モデルを提案し，これに基づいて利用者の要求知識，インタフェースの要求仕様，および設計仕様が明確に表現される．また，本方式では，複数の設計ステージとそれを連結する設計プロセスによって構成されるインタフェースの設計過程が，システムの機能設計とは独立した設計タスクとしてモデル化される．そして，各設計ステージが上記知識モデルとして表現され，また，各プロセスが，熟練した設計者の持つ種々の設計知識によって定義される知識モデル間のマッピングとして表現される．こうして，要求仕様定義からインタフェースの実現に至る設計過程は，マッピングによって連結された知識モデルの変換過程として形式化される．このようなモデル化により，利用者の種々の要求が，知識モデルを媒体として設計対象のインタフェースに十分反映され，同時に，熟練した設計者の持つ設計知識が，知識モデル間のマッピングとして設計過程の中で有効に利用される．さらに，本方式に基づいてインタフェース設計者の設計作業をインタラクティブに支援することによって，利用者の要求に適合したインタフェースが柔軟に，かつ半自動的に生成できる．以下，2章では本方式の基礎となる知識型設計方法論の概要を述べる．3章では，はじめにインタフェース設計のための知識モデルを提案し，次にそれを利用した知識型設計方法論に基づくインタフェース設計方式を提案する．そして，4章では本方式に基づくインタフェース設計支援システムを試作し，設計例を通してその有効性を示す．

2. 知識型設計方法論

高度な設計支援システムの実現を目指して多くの研究が行われている．しかしながら，熟練した設計者の持つ種々の設計知識を効果的に利用するための方法論を与える研究は極めて限られている[12)~21)]．著者らによって提案された知識型設計方法論[1)]は，設計型タスクで利用される種々の知識を設計過程において統一的かつ効果的に利用するための枠組みである．

一般にシステム設計過程は，**図1**に示すように，多くの設計ステージと設計プロセスから構成される．知識型設計方法論では，以下の2つの基本要素によって設計過程をモデル化する．すなわち，

［A］ 設計ステージを知識モデルとして表現する．

［B］ 設計プロセスをマッピング（写像）として表現する．

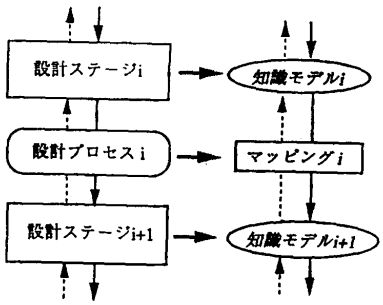

図1 知識型設計方法論の枠組み
Fig. 1 Framework of the knowledge-based design methodology.

ここで，知識モデルは，利用者要求／設計仕様／設計部品情報／設計状態情報など，設計対象や設計問題によって与えられる知識（問題依存型知識）によって定義される．また，マッピングは，その設計ドメインの経験的知識／設計規則／基本原理などの設計知識（領域依存型知識）によって，1つの知識モデル上，あるいは2つの知識モデル間において定義される．こうしたモデル化により知識型設計方法論に基づく設計過程は，マッピングによって連結された知識モデルの変換系列として形式化される．また，本設計方法論では，設計者がマッピングの過程にインタラクティブに関与することにより，柔軟性の高い設計形態を実現する．

3. 知識型設計方法論に基づくインタフェース設計

3.1 ジェネリックファンクションとジェネリックオブジェクト

高度インタフェースの実現においては，まず，利用者の持つ要求（要求タスク）を抽出し，それを設計対象となるインタフェースに効果的に反映させることが重要である．そこで，本方式では，インタフェースに関する要求を次の3種類の知識（インタフェースの基本要求知識）を導入し，これに基づいて利用者の要求を抽出・整理する．

（1） インタラクション：利用者とシステムがインタフェースを介して行う様々な相互作用（操作，応答）に関する知識．

（2） プレゼンテーション：利用者がインタフェース上で直接的／間接的に操作／観測が可能な対象の形態や構造に関する知識．

（3） アクティベーション：利用者に提供されるサービス機能の実体となるシステムの内部機能モジ