ソフトウェア工学　56－3
（１９８７．９．９）

ユーザインタフェースの設計モデル

中村　能章＊　　大竹　勤＊＊　　中川　透＊
＋ＮＴＴ　研究所　　＋＋ＮＴＴ

サービスの使い勝手は，一般に，サービス・フローを含めた総合的なユーザインタフェースで評価される．個々の入出力およびその組合せは，サービスの全体の流れの中で評価される．この観点からすると，ＵＩＭＳでプロトタイピングを提供するには，サービス・フロー制御と入出力の組合せを統合的に扱える設計モデルが必要になる．

本研究では，状態機械に基づいてＵＩＭＳの構成を考察する．まず，サービスをプロセスの集合に分割し，分割されたプロセスの入力とシーケンス管理の要求条件を検討する．同時に，個々の入力とプロセスの関係について考察し，ユーザインタフェースの設計が設計モデルを駆動する種々の関数に置換えられることを明らかにした．

A Model of User Interface

Yoshiaki NAKAMURA ＋, Tsutomu OHTAKE ＋＋ and Toru NAKAGAWA ＋
＋ＮＴＴ Laboratories　　＋＋ＮＴＴ
1-2356 Take Yokosuka-city Kanagawa Pref. 238-03, Japan

User-friendliness of a system is generally evaluated through its user interface design that is composed of service sequence and user's actions. As the service sequence is the main factor of the user interface design, the input and output of each stage should be considered in relation to the service sequence. A design model that describes actions and sequence uniformly plays an important part in the design of prototyping a service system used User Interface Management System (UIMS).

In this paper, the UIMS is considered as an automaton. A service sequence is divided several processes, and requirements to the input and output, and to the sequence management of these process is considered in relation to transition of state function. The basic design of user interface allowed to replace the description of service sequence to a set of state function is proped here.

1．まえがき

ユーザインタフェースは，ユーザの入力情報をアプリケーション（ＡＰ）の入力形式に自由に変換できない，ＡＰの様々なサービス要求に柔軟に対応できるフロー制御ができないなどの問題があり，ＡＰに応じて設計する手法が一般的であった．ユーザインタフェース管理システム（ＵＩＭＳ）は，ＡＰ設計で大きなウェイトを占めるユーザインタフェースの設計・製造を効率的にすることを目的としている．このため，プロトタイピングの容易な設計モデル，機能構成が提案されている．[1),2)]

従来のＵＩＭＳは，入出力関数を組合せ，ある場面での一塊りの入出力を表現できる仕様をＵＩＭＳの設計モデル，言語仕様として提案していた．このため，ユーザインタフェースの重要な要素のひとつであるサービス・フローは，プロトタイピングあるいはＡＰの製造において，ＵＩＭＳの言語仕様に従ってコーディングする必要があった．[2),3)]

サービスの使い勝手は，時間的な評価を除けば，サービス・フローを含む総合的なユーザインタフェースで評価される．個々の入出力，入出力の組合せは，あくまでサービス全体の流れの中で評価される．このため，試行錯誤的なプロトタイピングが特に重要となる．このプロトタイピングが容易にできるＵＩＭＳを提供するには，サービス・フローの制御と入出力の組合せを統合的に扱える設計モデルが必要になる．

ここでは，ソフトウェアの挙動を状態の関係で表現する機械モデルに従って設計モデルを検討する．このため，サービスをプロセスの集合に分割し，プロセスを入力とサービス・フローを管理する観点で分析し，要求条件を明らかにする．同時に，個々の入出力とプロセスの関係を考察し，サービス・フローの制御と入出力の組合せを統合的に管理できる設計モデルを構成する．

最初に，サービスをプロセスの集合と捉え，設計モデルのベースとなる設計環境を提示する．次に，サービス・フローが表現できるＵＩＭＳの設計モデルを提案する．最後に，設計モデルを実現するプログラムの設計モデルについて考察する．

2．サービス設計の基本モデル

サービスは，いくつかのプロセスを組合わせたアプリケーション・プログラムとして実現される．プロセスは，プロセス間の順序に無関係な独立プロセスと順序に依存した逐次プロセスに分けられる．サービスは，２種類のプロセスが混在して提供されることが一般的である．このため，まず２つのプロセスの関係を規定し，個々のプロセスをモデル化することにより，サービスの設計モデルを構成する．

2．1　サービスの構成

サービスをプロセス p^j の空でない有限な集合Ｐとする．このとき，サービスは，他のプロセスと任意の順序に組合せて実行できる独立プロセス $p_i{}^j$ の集合 P_i と組合せの順序に拘束のある逐次プロセス $p_d{}^j$ の組合せ P_d によって構成される．

このとき，P_i と P_d の間に順序性があると，P_i の独立性が失われる．これより，P_d 自体は独立プロセスである．これより，サービスは，逐次プロセスを組合せた独立プロセスの集合として以下のように表すことができる．

$$P = \Sigma\,(\Pi\, p_d{}^i)^j \qquad \cdots\cdots (1)$$

独立プロセスと逐次プロセスの組合せは，一般に，さら複雑で，単純に分離できない．しかし，このモデルを適用することにより，ひとつのプロセスを複数のプロセスの構造体として構成することができる．このため，モデルの最小単位である逐次プロセスをひとつのサービスとして考える．これにより，サービスに対する議論を適用すれば，逐次プロセスを独立プロセスの集合で表すことができる．この議論を再帰的に適用すれば，サービスは，独立プロセスの集合で表せる面（独立面）と逐次プロセスの集合で表せる面（従属面）の２つの面を交互に組合せた多階層構造で表現できる．サービスの階層構成を図２．１に示す．

この議論では，サービスは，ある条件下で独立なプロセスに分割できることを示している．このとき，独立面から従属面を見ると，概念上面を分離しているだけで等価と考えられる．一方，従属面から独立面を見ると，２つの面の間には明らかに何らかの写像が存在する．以下は，従属面から独立面への写像関係につい

て述べる．

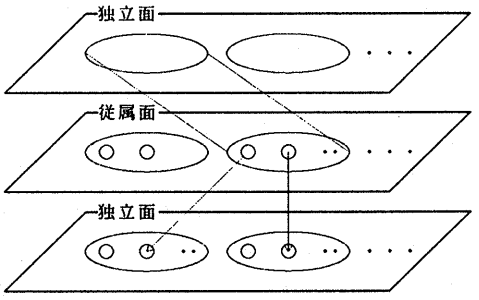

図２．１　サービスの階層構成

２．２　各階層の機械モデル

本節では，考察の前提となる，独立面と従属面におけるそれぞれのプロセスの機械モデルを仮定する．

独立プロセスが時刻 t_i に状態 s_i にあるとき，プロセスは，入力 i_i に応じて次の状態 s_{i+1} に遷移する．独立面では，直前のプロセスとは無関係，すなわち時刻 t_i に無関係に，状態間の関係で同一面上のプロセスの関係が表現できるから，入力，状態，状態遷移によってサービスを表すことができる．このため，有限状態機械で独立面のプロセス関係を表す．有限状態機械は，以下に示す５項組（Ｓ，Ｉ，δ，s_0，Ｆ）で表される．[1]

入力　　$I = \{ i : 有限,\ I \neq \phi \}$，

状態　　$S = \{ s : 有限,\ S \neq \phi,$

$\exists \delta\ (S \times I \rightarrow S) \}$，

初期状態 $s_0 \in S$，

終了状態 $F = \{ F \neq \phi,\ F \subset S \}$．　　・・・・ (2)

関数 δ は，状態を遷移させる状態関数を表す．

従属面は，独立面と同様な状態モデルに，状態間の順序関係が拘束条件として追加される．このため，従属面は，独立面の特異な例と捉えることができる．

順序性の概念は，半群の概念を用いて数学的に拡張できる．ここでは，拡張された有限状態機械で独立面のプロセス関係を表す．有限状態機械は，以下に示す５項組（Ｓ，Θ，τ，s_0，Ｆ）で表される．
ルを以下に示す．

入力半群 $\Theta = \{ \theta : \exists \rho\ (I \times I \rightarrow \Theta),$

$\exists \theta_e\ (\theta_e\ \theta = \theta) \}$，

状態　　$S = \{ s : S \neq \phi,$

$\exists \tau\ (S \times \Theta \rightarrow S) \}$，

初期状態 $s_0 \in S$，

終了状態 $F = \{ F \neq \phi,\ F \subset S \}$．　　・・・・ (3)

ここで，関数 ρ は入力の集合から自由半群を生成する生成関数，関数 τ は状態を遷移させる状態関数を表す．それぞれのモデルにある状態関数 δ，τ は，以下の関係を満たしている．

$\tau\ (s_i,\ i^i\ i^j) =$

$\delta\ (\delta\ (s_i,\ i^i),\ i^j)$　　・・・・ (4)

以下は，これらのモデルを前提に設計に必要な機能について考察する．

３．サービス設計モデル

本章では，前節で提示された基本モデルに基づき，サービスの設計に用いる機能モデルを構成する．前節の(2)～(4)式は，状態関数の積が表現できる関数が存在すれば，入力半群Θを生成する関数 ρ（生成関数）を設定することにより，従属面が表現できることを表している．(4)式は，プロセスの逐次実行の一般的な表現であって，状態関数 τ は単にモジュール表現の問題として常に存在するといえる．したがって，サービスは，独立面，従属面を問わず，入力を状態と対応付け規定することにより，表現することができる．

ここでは，まず状態遷移と入力についてそれぞれ考察する．これらの結果を基本モデルに適用し，従属面をベースに独立面と従属面を統一的に扱うサービス設計モデルを検討する．

３．１　状態遷移

状態関数は，状態を遷移させるプロセスを表現する．状態が何ら仕事をしないとすると，明らかに状態関数が仕事をすることになる．ある状態に対して複数の状態遷移が考えられるならば，状態関数は，複数の仕事を行う可能性がある．プログラムの実現可能性において，種々の機能が一元的に表す関数表現は困難である．このため，離散的な状態を与える関数の構成について考察する．

ある区間を連続的な関数として表現できない場合，区分的な連続領域で定義できる関数を組合せてひとつ

の関数を表すことが一般的である．ここでは，離散的なプロセスの状態を区分的に結合した，状態関数を考える．

関数は，区分的に扱う場合，適用域を定義域より割出し，定義された関数を適用する．このため，定義域を割出す関数（推移関数），定義域に応じた処理を実現する関数（部分関数）の２つの関数が必要となる．この２つの関数は，時系列的に発生する事象であって，それぞれの作業の順序性は一意である．ここで，部分関数をδ_p，推移関数をδ_d，状態関数δは以下のようになる．

$$\delta = (\Sigma \delta_p{}^i) \times \delta_d . \quad \cdots\cdots (5)$$

ここで，部分関数$\delta_p{}^i$によって生成される遷移状態をS_{it}とすると，状態関数の構成は図３．１に示すようになる．

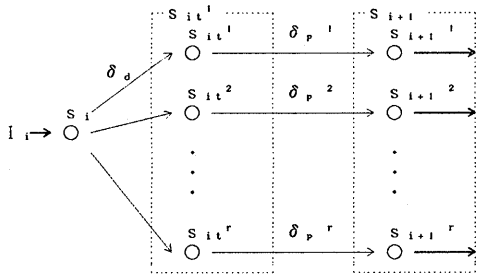

図３．１　状態関数の構成

状態関数は，それ自身をひとつのサービスと考えると，従属面を構成することが分かる．したがって，状態関数は，遷移状態s_{it}から状態s_{i+1}，状態s_iから遷移状態s_{it}への遷移を規定する逐次プロセスの集合と見なすことができる．

推移関数δ_dは，入力に応じて遷移状態s_{it}を設定する．いわゆる本来の状態遷移関数である．一方，部分関数δ_pは，自己の仕事によって変化した状態を表す状態関数である．

状態遷移は，一定の条件に従って行われる．このため，推移関数δ_dは，部分関数δ_pに対して可換でない．一方，部分関数δ_pは，関数が独立であれば，任意の組合せが考えられる．このため，推移関数δ_dは逐次プロセス，部分関数δ_pは独立プロセスと考えられる．

ここで，関数相互の関係は状態によってのみ規定されると仮定する．これにより，相互関係は状態という入力に置換えられるから，個々のプロセスは，プロセスの規定した入力と状態によって任意に規定できる．このため，関数の独立性は常に保証できる．

これらの議論から，状態関数は，逐次プロセスと独立プロセスを組合せ，逐次プロセスの集合を見せていることが分かる．すなわち，遷移状態s_{it}から状態s_{i+1}への遷移を規定する独立面を推移関数δ_dによって従属面に写像している．

独立面の推移関数δ_dは，状態の発生は任意であるから，状態s_iと入力I_iを対象にプロセスを構成すればよい．一方，従属面のそれは，状態s_i，s_{i+1}の順序性を考慮したプロセスとする必要がある．このため，独立面と従属面の推移関数δ_dは，それぞれ開放型，フィードバック型の制御プロセスということができる．それぞれの制御プロセスの概念を図３．２に示す．

実用面への適用を考えると，独立面は，ライブラリ，システム関数などを一般的に表すと考えられる．これに対し，従属面は，通常のプログラムの機能仕様を一般的に表すと考えられる．ユーザインタフェースは，独立した機能ブロックあるいはプログラムを制御する．以下は，フィードバック型を前提に考察する．

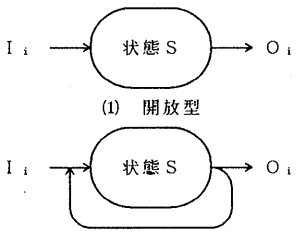

図３．２　推移関数δ_dの機構

３．２　入力関数

従属面の入力としては，以下に示す２つの場合が考えられる．

①　部分関数δ_pに応じた入力を規定する．

②　入力半群として入力を規定する．

部分関数δ_pに応じた入力は，部分関数対応の会話入力を規定する．一方，入力半群は，部分関数を組合せた一括入力を規定する．基本モデルによれば，部分関数の順序性を入力の組合せと対応付けると，２つの場合の結果は等価になる．したがって，部分関数が規

定された場合，入力半群の構成，すなわち入力半群の生成関数を規定すれば，サービス・フローを含めたユーザインタフェースが決定できることが分かる．ここでは，入力半群の生成関数（入力関数）について考察する．

入力半群の生成関数をρ，入力半群を連結によって与える部分的な生成関数をρ_p（部分生成関数：$\Pi I \rightarrow \Theta_p$，$\Pi \Theta_p \rightarrow \Theta$）とする．このとき，部分生成関数$\rho_p$は，独立プロセスと逐次プロセスのそれぞれを含むから，以下に示す関係を満足する．

$$\rho = \Sigma \left(\Pi \rho_p{}^i \right)^j . \quad \cdots\cdots (6)$$

(4)式の関係は，再帰的に適用できるから，部分生成関数ρ_pは多階層に定義できる．このため，独立面，従属面のそれぞれに部分生成関数ρ_pによるプロセス実行が定義できる．以下，部分生成関数の意味について考察する．

部分生成関数の積で与えられる部分$\Pi \rho_p{}^i$は，明らかに会話などで得られる状態を与える．一方，和ρは，任意の組合せを与えるから，一括入力に対応付けられる．ここでは，現実社会の環境に基づいてこれらの意味につい考える．

会話は，本質的には抽象的な概念あるいは意味の伝達を目的とする．人間の会話は，話し手にとっては順序立っているが，聞き手にとっては必ずしも順序立っていない概念の遣取りで成立っている．このため，聞き手にとっては，任意の順に送られてくる概念を自分で再構成する必要がある．ここでは，伝送順序は意味を持たないから，(4)式の和で与えられる部分に相当すると考えられる．

一方，伝達する概念を符号化・復号化する上で必須となるのは，信号方式，プロトコルといった文法がある．これらは，内容，順序を一定の制約の下で運用する必要があるため，(4)式の積の部分で与えられると考えられる．

応用的には，積$\Pi \rho_p{}^i$は，文字，音声，画像など各種メディア対応に規定される処理から，抽象概念を取出す操作となる．和ρは，フェーズに応じた会話を管理する．ここでは，積$\Pi \rho_p{}^i$を文法関数，和ρを会話関数と呼び，別々の階層として構成する．入力関数の構成を図３．３に示す．

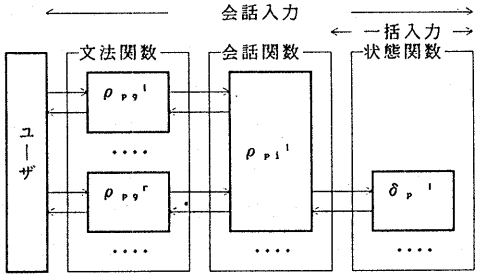

図３．３　入力関数の構成

(4)式の状態は，入力半群Θによって生成された入力の集合を示す．会話関数の本来の持つべき概念の再構築は，入力とは独立に従属面の構成に応じて規定する必要がある．ここでは，部分生成関数ρ_pで与えられる入力集合と逐次プロセスの状態により，部分関数δ_aの入力を与える分配関数ρ_aを考える．(4)式は，生成関数の逆関数として，分配関数が存在することを前提としている．このため，分配関数ρ_aは，入力関数の設定に必須の機能といえる．生成関数ρと分配関数ρ_aの関係を以下に示す．

$$\rho = \Sigma \rho_a{}^r \quad \cdots\cdots (7)$$

これにより，入力関数として考えるべき機能はすべて抽出された．

３．３　サービスの設計モデル

前節では，状態関数と入力関数のそれぞれの構成について述べた．ここでは，状態関数と入力関数を組合せた設計モデルを提案する．前節までに得られた考察の結果を以下に示す．

(1) サービス・フローは，入力と状態の組合せで表現できる．

(2) 入力は，状態に応じた値の集合で，会話関数ρによって構成される．

(3) ユーザの個々の入出力動作は，文法関数$\rho_p{}^i$によって抽象化される．

(4) 状態関数δは，推移関数δ_aと部分関数$\delta_p{}^i$の積で与えられる．

(5) 部分関数$\delta_p{}^i$の入力は，分配関数ρ_aによって決定される．

ここでは，入力が会話関数ρによって一括入力に変換できるから，入力動作とプロセスの実行を分けて考えることができた．しかし，従属面でプロセスを実行す

るために，会話関数の逆関数となる分配関数ρ_aを定義する必要性が提示された．以下では，状態関数に分配関数ρ_aを適用する方法について考察する．

状態関数δは，推移関数δ_dと部分関数δ_pで構成される従属面で，会話関数ρからの一括入力よって動作する．このため，分配関数ρ_aをそれぞれの関数について考える必要がある．したがって，状態関数δは，以下のように表される．

$$s_{i+1} = \delta(s_i, i_i) = \delta_p(\delta_d, \rho_a^p(\delta_d, i_i)) \times \rho_a^p(\delta_d, i_i) \times \delta_d(s_i, \rho_a^d(s_i, i_i)) \times \rho_a^d(s_i, i_i). \quad \cdots\cdots (8)$$

ここで，ρ_a^jはそれぞれの部分関数の部分生成関数である．(8)式の状態関数の構成を図３．５に示す．

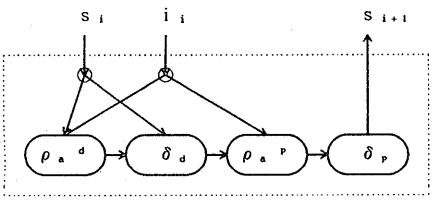

図３．５　状態関数の構成

従属面は，逐次プロセスの積で表される　．この場合，部分関数δ_pは，さらに小さな部分関数δ_p^2の順序付きの集合に分割できる．また，分配関数ρ_a^pも同様に，部分分配関数ρ_a^{pi}に分割できる．これより，従属面のプロセスδ_pは，以下のように表すことができる．

$$s_{i+1} = \Pi(\delta_p^i \times \rho_a^{pi}). \quad \cdots\cdots (9)$$

部分分配関数と部分関数の関係を図３．６に示す．

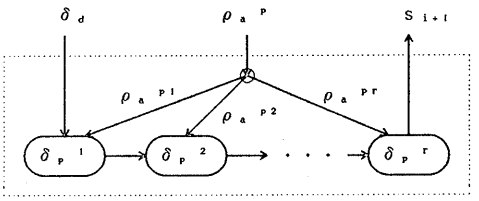

図３．６　部分分配関数と部分関数の関係

以上の考察により，(5)～(8)式の関数を決定すると，個々の入出力動作だけでなく，サービス・フローを含む総合的なユーザインタフェースが設計できることが分かった．次章では，(5)～(8)式を実現するための設計条件について考察する．

４．サービスの駆動モデル

前章では，サービス設計の一般モデルを提案した．設計モデルは，サービス設計自体の仕様を記述するモデルとすることにより，入出力動作だけでなく，これらの動作間の関連を含めたユーザインタフェース指向のサービス記述言語あるいはサービス設計支援ツールが実現可能となる．一方，駆動機構とは対象とする目的が異なるため，駆動機構の設計モデルとして直接適用することはできない．すなわち，設計モデルは状態遷移を表現することを目的とするが，駆動モデルは遷移時の状態の引継ぎを表現することを目的とする．このため，同様の要素が異なる意味をもっている．本節では，設計モデルと駆動モデルの各要素の関係について考察し，駆動機構の設計モデルを提案する．

４．１　基本モデル

設計モデルでは，状態関数δ_iを従属面として分析した．このとき，状態関数δ_iは，独立面を構成する部分関数δ_p^iを含んでいた．このため，δ_p^i以外の他の部分関数δ_d，ρ_a^d，ρ_a^pは，独立面を駆動するための駆動関数と考えられる．一方，独立面と従属面は等価と考えているから，従属面の駆動関数は独立面の駆動関数と等価である．駆動関数の基本型を図４．１に示す．

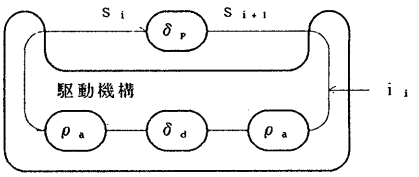

図４．１　駆動関数の基本型

４．２　階層構造

従属面は，入力の順序付き組合せで規定できることを示した．この場合，状態に応じた会話関数が独立に定義できれば，独立面と同様な駆動機構で表現できる．すなわち，会話関数と部分関数を独立に扱える環境を整えることにより，入出力とサービス・フローを統合的に駆動することができる．

部分関数と会話関数を独立に取扱う場合，これらの状態を管理する従属面，すなわちサービス・フローを制御する面が必要になる．ここでは，推移関数，分配

関数を組合せた部分関数と会話関数を起動する機構として面を規定する．これを推移面と呼ぶ．

一方，会話関数は，文法関数を駆動するから，文法関数群の駆動面となる．一連の会話を構成するため，推移面と同様の構成をとると考えられる．部分関数も同様である．さらに低機能な部分関数の組合せとして実現できるから，同様な駆動面を構成する．ここでは，それぞれを会話面，機能面と呼ぶ．

この議論は，再帰的に下位の文法関数，部分関数にも適用できる．一方，ユーザインタフェースを考える場合，これらは取扱いメディアに応じて構成する低位の関数である．ここでは，これらをそれぞれ文法面，処理面と呼び，一括して扱うこととする．このように階層化した駆動モデルを図４．２に示す．

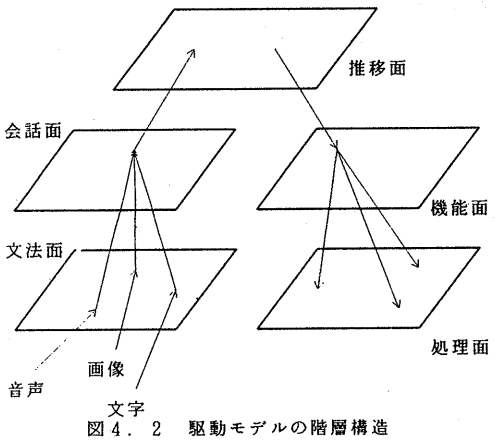

図４．２　駆動モデルの階層構造

４．３　駆動モデルの構成

駆動モデルを階層化した場合，推移面，会話面，機能面は，それぞれ下位の面を駆動するから，同様な駆動機構をもつと考えられる．ここでは，この駆動機構の設計条件について考察する．

推移面は，会話面と機能面のサービス・フローを管理する．推移面の入力としては入力集合と状態が考えられる．これらは，それぞれ会話面，機能面の状態関数の結果である．これより，駆動関数は，独立面を構成する部分的な面の出力を入力として部分的な面間の状態遷移を管理する関数と定義できる．部分的な面からの出力をo^iとすると，駆動関数は以下のように表せる．

$$s_{i+1} = \delta\ (s_i,\ \Sigma o^i) \qquad \cdots\cdots (10)$$

ここでは，状態s_iは，駆動モデルの状態であって，設計モデルの状態ではない．これより，推移面には，サービスと自己の２つの状態管理が必要なことが分かる．

会話面と機能面は互いに従属的に遷移するから，出力o^iが同時に発生することはない．この場合，(10)式は，(5)～(8)式の従属面を表す関数と等価になる．一方，出力の到着が無秩序，たとえばユーザの要求が同時に複数入ってくる場合を考えると，複数の独立面を並列に駆動するから，(10)式は以下のように書替えかれる．

$$s_{i+1} = \Sigma\ \delta\ (s_i,\ o^i) \qquad \cdots\cdots (11)$$

これは，並列実行すべきプロセスが起動要求に応じて独立に管理できることを示している．また，状態の組合せは，実行状態の和で与えられることを示している．これは，プロセスの生成・消滅が任意にできることを暗示する．

設計モデルでは，プロセスを入力の組合せとして規定した．そこでは，遷移契機としての入力に着目していた．ここでは，状態管理情報としての入力について考えるため，入力要素について述べる．

外部からの入力は，一般に，直前の関数の結果によって決定される入力，外部からの割込みなど，他の要因によって直前の事象とは無関係に決定される入力の２つが混在している．これらの入力は，駆動モデルの現在の状態に無関係な要因に支配されている．したがって，外部入力は，状態に無関係な既知の要素として扱う．

外部入力としては他に，デフォールトのような要素が考えられる．デフォールトは，状態に応じて一定のアルゴリズムで設定できるため，状態に関係した入力といえる．

状態に関係する要素として，サービスの状態，出力などのプロセス間の引継ぎ情報がある．これらは，出力o^iとして一括して扱える．

入力は，外部入力，デフォールト，引継ぎ情報の３つを組合せたものと言える．これらは，(6)式によって一括入力に変換され，分配関数で状態に応じた入力に再配分される．

これらの考察より得られる駆動モデルの設計条件は以下のようになる．

(1) 各面は，同一駆動機構で設計する．

(2) 各面は，他の面からのメッセージの流入で起動されるデータフローマシンを構成する．

(3) 各面は，他の面とオブジェクト間通信で起動，データ転送を行う．

(4) 会話面，機能面は，従属面毎に独立に設定する．

(5) 入力は，外部入力，デフォールト，引継ぎ情報の3つの組合せとする．

5．あとがき

本研究では，ユーザインタフェースがサービス・フローの中で評価されるとの視点から，サービス・フロー制御機構をもつユーザインタフェースの設計モデルを提案した．これより，ユーザインタフェースの設計は，設計モデルを駆動する種々の関数の設計に置換えられた．

一般に，特性の規定された関数は，線形結合のように係数行列を用いた変換で個々の応用へ適用する．プログラムは，同様な操作をテーブルによるフロー制御で実現する．これより，設計モデルにある条件の関数と関数を駆動するテーブルの設計でユーザインタフェースが実現できることが分かる．

このモデルによれば，サービスの設計は，状態，状態遷移，状態毎の会話，プログラムの入出力条件を設定するテーブル設計となる．これらの要素は，プログラム上の関数を統合した概念である．このため，ユーザにとっての親和性の高い設計言語，あるいは設計用のエディタの提供が可能となる．さらに，設計，駆動の概念が統一されているため，設計がプロトタイピングと等価になり，ユーザインタフェース設計の効率がよくなると考えられる．

設計モデルについては，有効性について検証するため，ユーザインタフェースの設計環境を構築する必要がある．現在，モデルに基づいた設計環境を試作しているので別途報告する．ここでは，ユーザインタフェース設計に最低限必要となる機能を明らかにするため，プログラム設計の方向からユーザの行動を限定した．今後は，本モデルとユーザの行動モデルを対応付け，本モデル限界および機能拡張について検討する．

謝　辞

本研究にあたり，御指導頂いた結城開発部長，ＨＩ研杉山視覚部長，安田画像部長に感謝します．また，有益な助言を頂いた遠藤主幹員，堀口主幹技師，岸本主幹員，片岡主幹技師に感謝します．

参考文献

1) Pfaff, G. E.; User Interface Management System, Springer-Verlag, 1985

2) 守屋他；ユーザインタフェース管理システムの方式とその試作，情報処理学会研究会87-CAD-25，1987

3) 戸村他；言語設計システム「つくばね」のユーザインタフェース記述言語SDRLの言語仕様とその定義方法，情報処理学会研究会87-SW-52，1987

4) Arbib, M. A.; Theories of Abstract Automata, Prentice-Hall, 1969