ソフトウェア工学　56－4
（１９８７．９．９）

画面ドライバと設計支援システムへの適用

大竹　勤*　　中村　能章**　　中川　透**
＊ＮＴＴ　　＊＊ＮＴＴ　研究所

ユーザインタフェースは，一般に，ビジュアルな表現とユーザのアクションを組合せた複雑な構成となっているため，プロトタイピングを繰返すことによってユーザの要求を最大限満たす仕様を設定することが望ましい．また，インタフェース設計では，ユーザのアクションを高級言語で記述するよりも，設計対象そのものをビジュアルかつインタラクティブに構成する方が便利である．本論文では，サービス・フロー制御と会話的入出力を統合した設計モデルに基づいて画面を単位としてサービス・フローを制御する画面ドライバ（ＳＯＤＯＭ）およびＳＯＤＯＭを用いてユーザインタフェースの設計が画面上でインタラクティブにできる設計支援システムの試作概要について報告する．これらを使用することにより，プロトタイピングの手順が簡略になった．

A Window Manager, SODOM and Its Application to Design Supporting Tool

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

A good user interface design has need to make prototype that satisfies user's requirements. Because, visual presentation on a screen and user's actions are very complicated. In this paper, an idea of the service sequence control machine, called "SODOM", based on a window control technique are proposed.

To design the user interface visualy and interactively is better than to describe by using high level language. A design supporting tool based on SODOM gives interactive design means of user interface. With this tool, the prototype of a service system can be made easily and fastly.

1．まえがき

ユーザインタフェースは，直接ユーザに接する部分であり，ユーザの操作特性あるいは要望を十分に採入れて設計する必要がある．設計者は，プロトタイピングを繰返してユーザの要求に沿った仕様に近づけることが望ましい．このため，ユーザインタフェースのプロトタイピングは，サービス開発上の重要な課題のひとつとなっている．

プロトタイピングを容易にし，インタフェース設計を簡易にするため，ユーザインタフェース管理システム（ＵＩＭＳ）の検討が進められている．従来のＵＩＭＳは，最終的にはインタフェース設計言語として提案されてきた．これらの設計言語は，あくまでインタフェース機能を仮想化したユーザのアクションを記述する高級言語指向の設計となっている．[1),2)] ユーザインタフェースは，ビジュアルな情報とインタラクティブなアクションを組合せたインタフェースである．ビジュアルな情報は，設計において，アクションの記述よりユーザフレンドリである．このため，設計を効率的に行うには，ビジュアルな設計言語あるいは設計環境を提供することが望ましい．[3)]

筆者らは，プロトタイピングの容易なＵＩＭＳを提供するため，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を提案した．ここでは，ユーザのアクションだけでなく，サービス・フローの制御まで一連のテーブル群としてデータベース化している．このため，本設計モデルに従ったプログラムを構築すれば，データベースを構築することによってサービスのユーザインタフェースのプロトタイピングが効率的にできる．また，サービス・フロー，入出力などのデータベース化を実際のサービス画面などと対応付けることにより，ビジュアルな設計環境が構築できる．

ここでは，最初に，本設計モデルに従ったプログラムとして，画面をひとつの単位としてサービス・フローを制御するＳＯＤＯＭ(Screen Oriented Driver Of service Management) の構成について述べる．次に，ＳＯＤＯＭを用いてユーザインタフェースのプロトタイピング，プログラム作成を支援する設計支援システムの概要を述べる．筆者らは，設計モデルの妥当性を確認するため，設計支援システムをＳＯＤＯＭを用いて試作した．本試作の概要も述べる．

2．設計モデル

設計モデルは，以下のようになっている．

(1) 入力と状態の組合せで表現されたサービス・フローを推移面で制御する．

(2) 入出力は，すべて会話面で制御する．

(3) 個々の入出力動作は，文法面で抽象化する．

(4) 処理は，ライブラリで実行するものとし，ライブラリの実行管理を機能面で行う．

(5) 会話面と機能面は，独立面を構成し，非同期で動作する．

設計モデルを図２．１に示す．

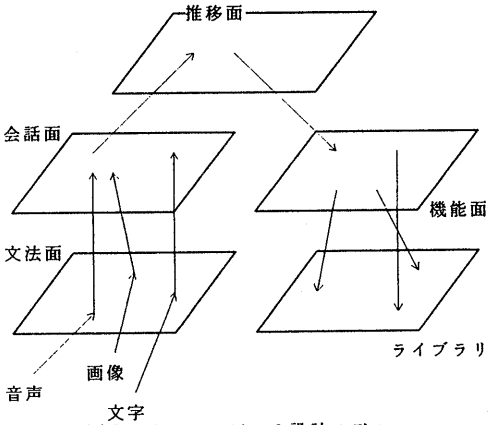

図２．１　サービスの設計モデル

これらの階層化された面のうち，推移面，会話面，機能面は，状態遷移を管理する面であり，図２．２に示すようなプロセス構成となっている．ここで必要な設計は，状態を推移させる関数δ_d（推移関数），入力パラメータを実行関数δ_pに合わせて最適化する関数　$\rho_a{}^d$，$\rho_a{}^p$（分配関数）および機能を専用化するテーブル構成である．一方，文法面と機能面は，会話面と機能面に対して機能要素を提供するから，汎用的なインタフェースが規定できればよい．

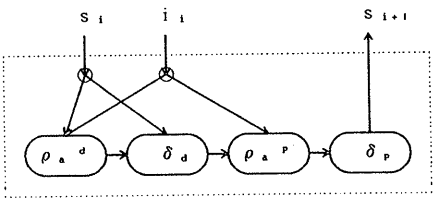

図２．２　状態関数の構成

駆動モデルを，図２．３のように構成し，入出力として、ユーザの入力，デフォールト，状態の引継ぎ，結果出力を規定した．これらはそれぞれの面で必要となり，機能面と会話面は，非同期な独立プロセスであるから，独立プロセスに対して並列実行制御が必要となる．

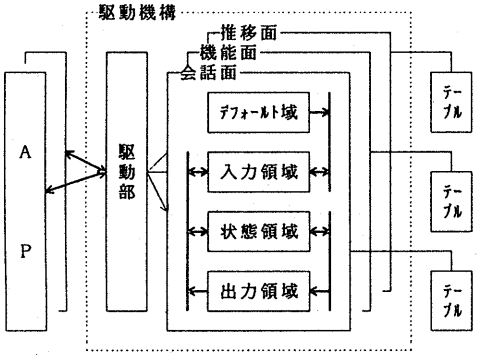

図２．３　ユーザインタフェースの駆動機構

３．ＳＯＤＯＭ

プロトタイピングを簡易化するため，前章に示した設計モデルに従ってＳＯＤＯＭを試作した．ここでは，ＳＯＤＯＭの試作概要について述べる．

３．１　ＳＯＤＯＭの概要

ＳＯＤＯＭは，前章の設計モデルに基づいたユーザインタフェースの駆動機能を持ち，外部モジュールとして規定されたサービス機能を駆動するコマンドの実行管理モニタである．ここでは，サービスの実行情報はすべてテーブル化する．ＳＯＤＯＭは，テーブル情報とサービス状態に応じて処理を起動・実行する．ＳＯＤＯＭの概要を図３．１に示す．

３．１　フロー制御の単位

サービス・フローの制御には，ユーザインタフェースを記述し易い制御単位を規定する必要がある．ここでは，ユーザの操作で最も煩雑なコマンド入力操作に基づいて考察する．

テレタイパ，ワークステーション（ＷＳ）などのコマンド入力は，形式の差はあるが，入力促進プロンプトに対してコマンドを投入する．受付け側は，コマンド投入によって，処理を分岐・実行していく．処理のの終了により，処理結果を出力して次の入力促進に移行する．プログラムの状態を図３．２に示す．

ここでは，ユーザに対して連続的に出力される処理結果出力，プロンプト出力の２つの関係を考察する．

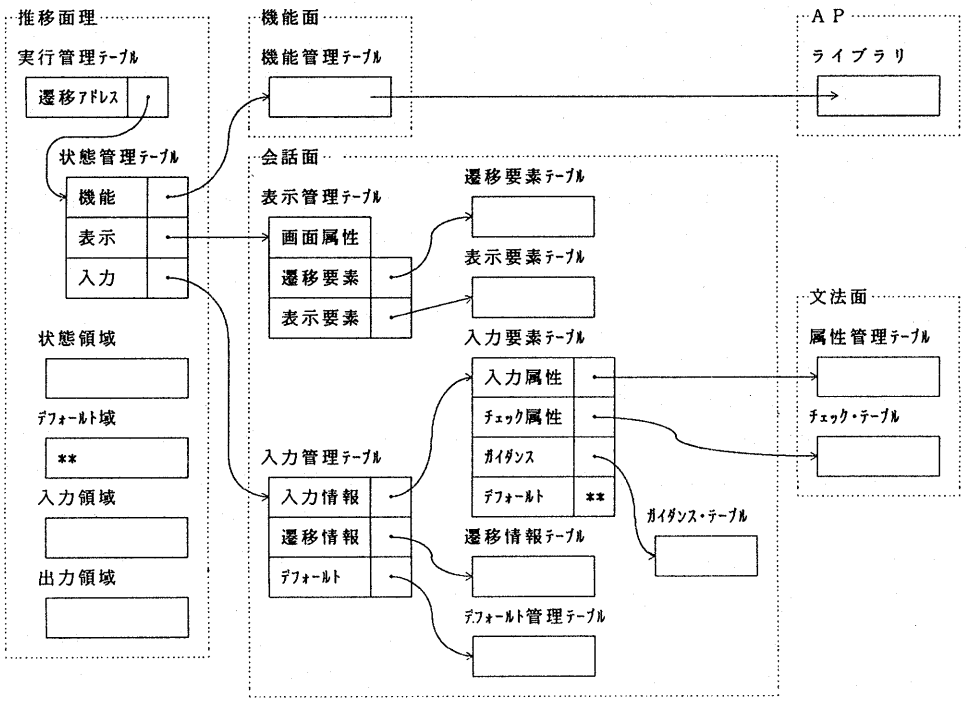

図３．１　画面ドライバの概要

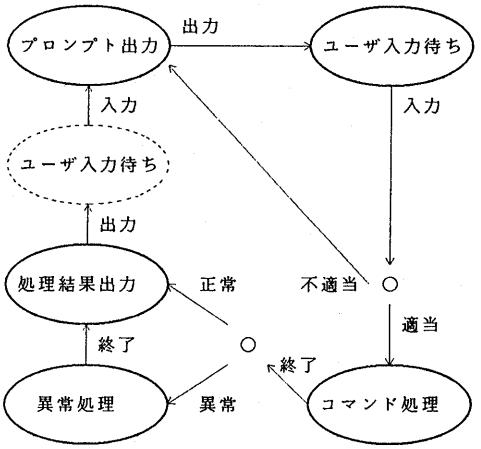

図３．２　プログラムの状態遷移

テレタイパなどは，出力がライン単位となるため，履歴を利用して状態の遷移はスムーズに行われる．一方，ＷＳなどは，画面単位の遷移となるため，遷移の契機が必要となる．一般には，メニュー選択などのコマンド投入を遷移の契機とするので，処理結果出力とプロンプト出力の２つのフェーズはユーザの入力待ちによって分離されている．

一方，アイコン，ファクション・キーなどでプロンプト出力できる方法がある．この場合，処理結果出力とプロンプト出力は重合せて出力されていると考えられる．分離する方法は，メニューなどの出力をひとつの処理と考えると，”処理結果出力＝プロンプト出力”と置換えた，画面の重合せで表現できる．また，プロンプト出力は，ユーザ入力とともに機械を発側とした会話系を構成する．このため，状態遷移は，”処理→処理結果出力＝プロンプト出力＝ユーザ入力待ち”で表せることが分かる．

ここで，ユーザが意識するインタフェースは，（処理結果出力＝プロンプト出力＝ユーザ入力待ち）である．これらは，画面を通してビジュアルにユーザに提示される．すなわち，ユーザからは，処理は画面に応じて遷移するように見える．サービスの状態遷移を図３．３に示す．

ここでは，サービス・フローを制御するための管理単位として画面を採上げ，画面を制御する処理系について述べる．サービス・フローは，画面単位に制御すると規定し，さらに，機械の自由度が低いため，話し手の要求と聞き手の応答で構成される会話の要求側を

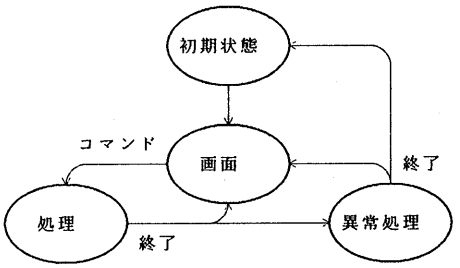

図３．３　サービスの状態遷移

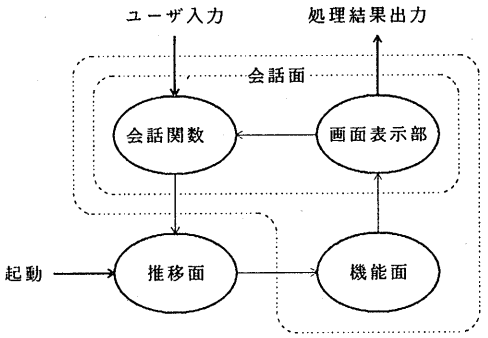

図３．４　推移面の機能構成

ユーザではなく機械とした．ここでは，機械は応答によって次の動作指示をユーザに要求する．このため，ＳＯＤＯＭの機能構成は，図３．４に示すようになる．

３．２　推移面の構成

分配関数ρ_a^dは，会話面の入力から遷移情報を抽出する．一般に，遷移情報は複数の情報から構成され，これらは木構造的な意味関係$t^{ij}=t(s^i, j)$（ｊはネスト数）が存在する．このため，分配関数ρ_a^dは，複数の情報から木構造に基づいて遷移情報を絞込み関数$\rho_a^d{}_{pp}$，ある情報について遷移を特定する関数$\rho_a^d{}_p$の積で与えられる．分配関数ρ_a^dは，以下のようになる．

$$\rho_a^d(s^i, i^i) = \Pi \rho_{adp}(\rho_{adpp}(t^{ij}, i^i)). \quad \cdots\cdots (1)$$

推移関数δ_dは，分配関数で得られた遷移表のアドレスに遷移する．また，機能面の関数が直前の関数と独立の場合，別の遷移表に遷移すると同時にタスクを分岐する．分岐された遷移表に従って実行されたタスクは任意の時点で終了できるが，タスクの消滅はユーザの指示による．このため，推移関数δ_dには，タスク管理機能が必要となる．ここでは，タスクの初期化を含め，機能面にタスク管理機能を設定し，各タスク

は起動時と消滅時に必ずこの機能を起動することにした．

分配関数$\rho_{a,p}$は，機能面のプログラム・インタフェースへのマッピングを行う．ここでは，関数を汎用化するため，インタフェースの構造を制限した．パラメータは，それぞれがＩＤを持ち，パラメータ・リストにはＩＤ，長さ，値の格納してある位置のオフセットだけを持つ．値がリストなどの場合は，値の格納域にこの形式のパラメータ・リストを格納する．インタフェースの構造を図３．５に示す．

部分関数δ_pは，遷移表にある機能を起動する．タスク管理機能は，推移関数が意識するひとつの機能であって，部分関数はこれを機能として包含しない．

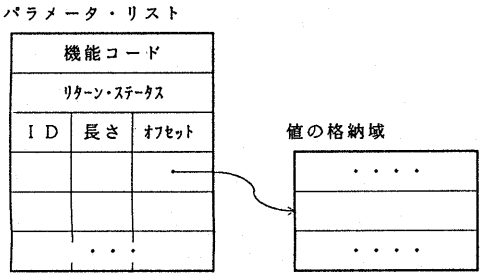

図３．５　インタフェースの構成

３．３　会話面の構成

会話面は，推移面と同じ機能構成となっている．このため，会話面の状態管理方法の説明は省略する．ここでは，会話面で独立面を構成する処理要素と状態遷移について述べる．

会話面は，基本的に画面表示と画面入力で構成する．

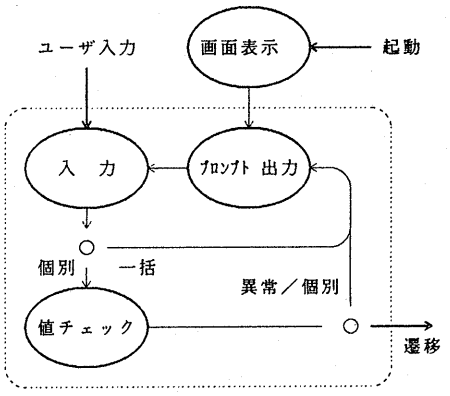

図３．６　会話関数の構成

画面入力は，プロンプト出力，入力，値チェック機能を提供する．これらの動作状態の概要を図３．６に示す．

入出力は，音声，キー入力など，ユーザのアクションに応じて構成される文法面で仮想化される．画面表示，プロンプト出力は，文法面のメディア対応の入出力関数によって結果を表示する．出力は個別に制御できるため，ＳＯＤＯＭの管理する出力情報のＩＤ，出力データでインタフェースを構成する．出力情報のメディア，出力条件などは，ＣＣＩＴＴ勧告ＭＨＳで用いられているものを仕様とし，出力情報のＩＤで対応付けられる表示管理テーブルに前以って記述しておく．出力インタフェースの構造を図３．７に示す．

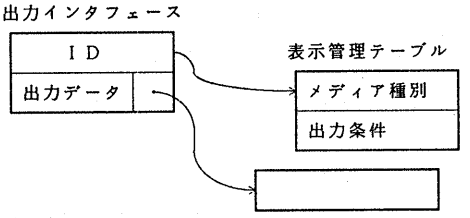

図３．７　出力インタフェースの構成

一方，入力は，例えば音声などのように，個々の情報に対する区切りが明確でない場合には，個別に入出力制御できない．このため，個別入力，一括入力の両方が適宜利用できるインタフェースが望ましい．このため，出力インタフェースを組合せた複合インタフェースを構成した．

また，画面の遷移契機と遷移情報を分離し，両方を同等な入力情報として扱った．このため，アイコン，ファンクション・キーなど遷移契機と遷移情報が同時に入力される場合を別々に入ってくる場合と同一手順で扱える．

値チェックの契機は，入力形式に応じて一括入力と個別の逐次入力の２つがある．このため，会話面には文法面からの入力を一括してチェックする方法と個別に逐次チェックする２つの制御機構が必要となる．さらに，１つの画面における入力を一括して行う場合が考えられる．このため，複数の入力値の一括チェックを文法面からの入力と画面遷移の２つの契機で行なえるようにした．ＳＯＤＯＭは，これを起動パラメータとして実行する．

値チェックの方法は，入力の値そのもののチェックと入力属性のチェックの２通りがある．値自体のチェ

ックは，連続量，離散量のそれぞれに対するチェックを考える必要がある．ここでは，マウスなどからの入力座標は，文法面でコマンドに変換され，離散量になっていることを想定している．このため，連続領域が不連続に定義される場合はない．したがって，チェックの方法は以下の２通りと考えられる．

(1) 離散量は，テーブルにあるコード列と直接マッチングする．

(2) 連続量は，上下限でチェックする．

入力属性のチェックは，入出力メディアに含まれる記号列のチェックとなる．基本的には，コード列のテーブル・マッチングであるが，連続量として扱えるものもある．これは，勧告化された属性によって様々な形式をもつため，出力で定義した属性についてチェックを個別に実現した．

３．４　機能面の構成

機能面は，画面で規定される機能を複数の関数を用いて実現する．ＳＯＤＯＭは，機能面の構成には関与しない．ただし，個々の関数は，状態を遷移させる状態関数であり，入力を含めた状態遷移が規定できる．このため，ＳＯＤＯＭは，推移面を機能面の制御機構として包含することとした．

これにより，ライブラリ化されたオブジェクト，ロードモジュールなどを任意に組合せてサービスが実現できる．このとき，ＳＯＤＯＭは，個々のライブラリのもつ遷移状態が仕様上で明確であれば，これをライブラリ毎に独立に管理する．すなわち，状態管理は，機能フローとマクロ・フローの組合せで状態を制御する．機能面の状態管理の概念を図３．８に示す．

ライブラリの状態管理は，実行状態が動的に変化する場合を除いて実行時に管理することは無駄が多い．このため，設計時に個々のライブラリの状態をチェックする．[4] 各ライブラリの状態推移が同一形式でテーブル記述されている場合，機能面の状態推移をテーブル化した段階で，マクロの発行シーケンスを自動的にチェックすることができる．この場合，状態だけでなく，入出力条件も同様にチェックできるため，機能設計の設計効率の向上が期待できる．

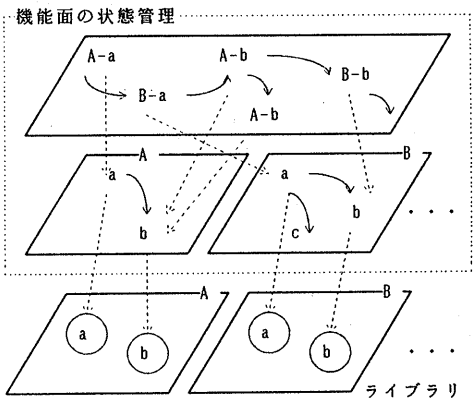

図３．８　機能面の状態管理の概念

４．設計支援システムへの適用

設計支援システムは，設計モデルのユーザインタフェースであって，テーブル設計を容易にするインタラクティブなツールである．ＳＯＤＯＭは，ＡＰの駆動部を含む設計モデルであり，テーブル設計によりインタラクティブなユーザインタフェースのプロトタイピングを可能とする．ＳＯＤＯＭで設計支援システムを構成することは，設計モデルの適用性を評価する上で意味がある．このため，ＳＯＤＯＭを用いたプログラムの設計支援ツールをＳＯＤＯＭの上に構成した．以下は，ＳＯＤＯＭの適用方法について述べる．

４．１　設計支援システムの概要

設計支援システムは，ＳＯＤＯＭの実行に必要な処理，会話機構を定義したテーブル群を生成するツールである．具体的には，図３．１に示す構成のテーブル群をインタラクティブに生成する．設計支援システムの画面の構成を図４．１に示す．

設計支援システムは，画面対応に機能面を設定するが，本来テーブル群の生成を目的とするため，提供機能は限定される．提供機能を以下に示す．

(1) テーブル生成．

(2) テーブル群の結合．

(3) エディタ．

ここでは，実行効率を犠牲にし，テーブル生成機能をテーブル・ドリブンに実現する．これにより，ほとんどの画面はテーブル生成機能で実現できる．

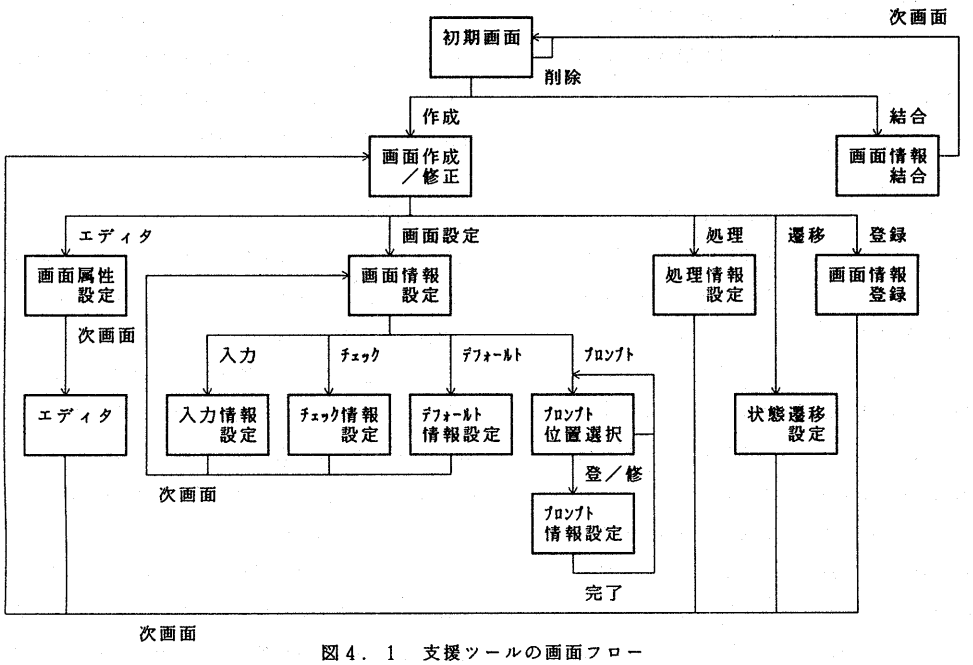

図４．１　支援ツールの画面フロー

４．２　適用方法

筆者らは，ＳＯＤＯＭ上に設計支援ツールを構築した．ここでは，支援ツールの構成概要について述べる．

４．３．１　支援ツールの試作概要

支援ツールをＳＯＤＯＭで構成する場合，サービス実現に必要な機能の製造，駆動テーブルの作成が必要となる．このため，ＳＯＤＯＭ，テーブル生成，テーブル群の結合，エディタなどの機能を先に用意した．テーブル生成は，簡易なツール用のテーブルをコーディングし，簡易ツールを用いてインタラクティブに実現した．支援ツールの構成手順を図４．２に示す．

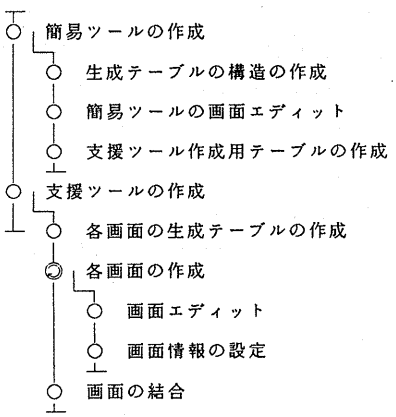

図４．２　推移面の構成手順

４．３．２　エディタの構成

画面を単位とした機能構成方法について考察するため，画面エディタを採上げ，機能の実現方法について述べる．なお，本試作では，エディタをＳＯＤＯＭの基本機能として内蔵する構成とした．ＳＯＤＯＭでエディタを構成する場合の例を図４．３に示す．

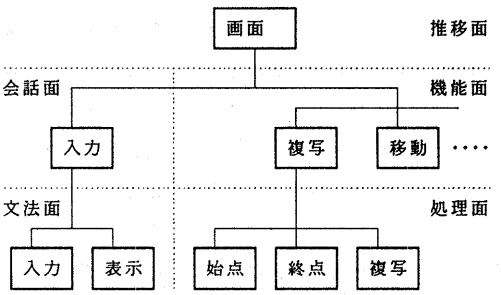

図４．３　エディタの機能構成

エディタでは，画面に入力ブロックを構成する．この入力ブロックの範囲であれば，複写，移動などのコマンドを指示する入力がない限り，文法面で入力を画面に展開する．文法面では，入力を画面にバッファリングし，コマンドによって入力を会話面に引継ぐ．会話面は，画面の遷移契機だけを抽出し，後は文法面の入力を機能面に引継ぐ．入力の引継ぎは，推移面を通して実行するが，ここでは，画面から溢れた入力・表示データを画面表示するだけの制御を行う．本試作で

は，数行をまとめてスクロールすることを前提に設計した．機能面では，各コマンドの状態を引継ぎながら，コマンド要素をライブラリから起動する．

5．考察

ＳＯＤＯＭの狙いは，インタラクティブなユーザインタフェースを含むシステムの機能設計のテンプレートを作成し，かつユーザインタフェースを効率化的に設計できる環境を設定することにある．これらの狙いは，本方式に基づいた設計を長期に亘って計測し，従来の設計時間との比較をしなければ，定量的に評価することは難しい．ここでは，ＳＯＤＯＭによる設計支援ツールの設計時間で簡単に評価する．

構成手順から分かるように，簡易ツールの作成ではテーブル群を手作業でコーディングした．一方，設計支援ツールの作成では簡易ツールを用いてインタラクティブに設計した．この作業で１画面の作成に掛かった時間を表５．１に示す．

表５．１　画面作成時間

	簡易ツール	設計支援ツール
設計時間	３時間	３０分

これらの時間は，画面設計終了後の単にテーブルを作成するため時間である．したがって，設計時間の差は，テーブル構造の仮想化による設計インタフェースの簡易化が大きな理由と考えられる．設計支援ツールがエディタを用いてインタラクティブに仕様を決定できることを考えると，画面および画面フローの設計を含めた工程では，さらに差が広がると考えられる．

高級言語を用いたインタフェース設計は，一般に，すべての制御機能がＡＰにあって，画面などの取扱いから設計する必要がある．このため，簡易ツールの作成以上に時間が掛かることが予想される．この点で，ＳＯＤＯＭは当初の狙いを達成している．

設計モデルで会話面と機能面を完全に分離したため，機能設計とユーザインタフェースの設計を独立に扱うことができる．このため，ユーザの要求を広汎に満たす機能モデルに基づいたモジュールが多数存在すれば，ＳＯＤＯＭを用いてプロトタイピングが可能となる．また，タイマにより遅延を掛けるモジュールによるユーザインタフェースの画面シミュレートも可能となる．これらのプロトタイピングは，モジュールの機能の範囲でユーザが自由にインタフェースを設計できる利点がある．このため，ＳＯＤＯＭと設計支援ツールは，設計に有効なツールと考えられる．

6．あとがき

ＳＯＤＯＭの構成により，サービスの実行制御機構の設計が容易になる見通しを得た．一方，以下のような問題点が明らかになった．

(1) ユーザとの会話を処理と完全に切離したため，インタフェースの木目細かな制御が難しい．
(2) 機能面の一様な構成方法が提示されていない．
(3) 概念的に２次元でない情報を画面の範囲で扱うための変換が明らかでない．

現在，仮想的な機能面をセンタに設定し，端末側のＳＯＤＯＭからアクセスするオンライン検索システムを試作している．今後は，エンドーエンド，センターエンド間通信を一元的に制御できる機能面を構成しつつ，上記の問題点を解決していく予定である．

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)守屋；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)
2)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)
3)岡崎；アダプタビリティを持つ端末ソフトウェア，情報処理学会シンポジウム論文集(1986)
4)小林；抽象方構成子概念に基づくソフトウェア部品体系を用いたソフトウェア構築方式，情報処理学会研究会86-SW-47(1986)