ユーザインタフェース管理システムにおける対話の設計

3H-6

石井久恵　清水徹　稲葉豊

三菱電機東部コンピュータシステム(株)

1. はじめに

近年、ユーザインタフェースに関する研究が、米国を中心にさかんに行われている。特にGraphical User Interface(以下GUI)の分野で活発な活動が展開され、目覚ましい成果をあげてきている。

一方、ユーザインタフェース部をアプリケーション･プログラム本来の処理部分から分離して開発できるシステムをUser Interface Management System(UIMS)と呼んで、さまざまなアプローチによるシステムが開発されてきている。

エンジニアリング･ワークステーション上の対話型ソフトウエアのほとんどは、GUIを用いている。GUIを用いた対話型ソフトウエアは、ユーザにとって親しみやすいものとなるが、システム側の対話制御は複雑になることが知られている。UIMSは複雑になりがちな対話制御を、効率的に開発するために有効である。

今回、プロダクション･システム(以下PS)をベースにしたUIMS[1]を用いて、アプリケーション･プログラムのユーザインタフェース部の開発を行った。

本論文は、その開発経験から本UIMSを用いた対話定義の記述に関する一方法について報告する。

2. 本UIMSの概要

2.1 本UIMSの構成

本UIMSの構成を図1に示す。

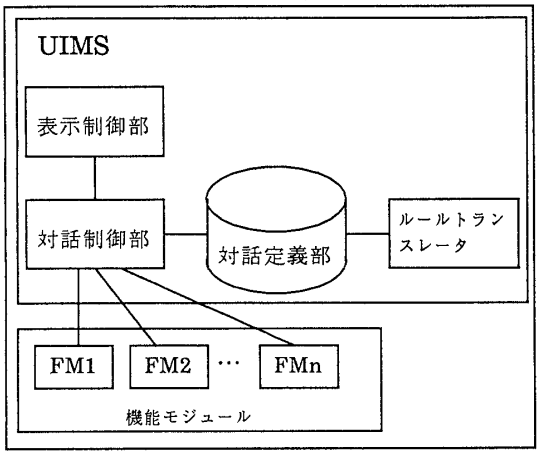


図1　本UIMSの構成

表示制御部は、入出力デバイスや入出力技法の管理を行う。具体的には、ユーザの入力をトークンに変換して対話制御部に渡し、対話制御部からの表示要求に応える[2]。

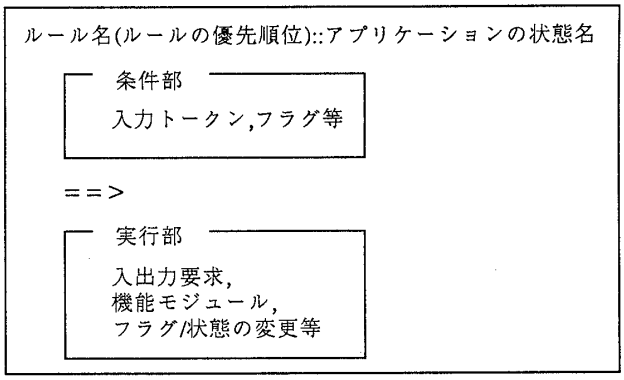
対話制御部は、表示制御部からの入力トークンを解釈し、対話定義部の設定に従って機能モジュールの起動や表示制御部への表示要求を行う。

機能モジュールは、アプリケーション･プログラム本来の処理部分である。

2.2 対話定義

ユーザとの対話をどのように進めていくかは、アプリケーション･プログラムの制御部分に関係する。本UIMSでは、この部分の対話定義をルール形式の記述で行う。記述ルールはルールトランスレータによって実行に適したルールに変換される。

記述ルールは、条件部と実行部から構成される。対話定義は以下に示す構文で記述する。



本UIMSのベースとなるPSのアルゴリズムは通常のPSとは異なり、ルールをアプリケーションの状態名によってグループ化して検索の範囲を限定する。さらに優先順位の高い順に照合が行われ、照合に成功したルールは即時実行される。

3. 対話設計上の問題とその解決

本UIMSを使用して、UNIX上の電子メールを拡張し、GUIをもった電子メールシステムの開発を行った。その際の問題点と解決策について述べる。

3.1 問題点

開発はC言語による機能モジュールの開発と、記述ルールによる対話の定義を同時に行った。対話定義はラピッドプロトタイピングで行い、次第にルール数を増加していった。

本電子メールの対話は、基本的にユーザ主導型の対話スタイルを採用した。ルールの構文は、ユーザのイベントにシステムが反応する形式の対話記述に適している。従って、個々のルールの記述は容易であった。

しかし、ユーザの入力を非同期に受け付ける場合ばかりでは対話のすべてを構築することはできない。システム側が主

Dialogue Design for the User Interface Management System

Hisae ISHII　Toru SHIMIZU　Yutaka INABA

MITSUBISHI ELECTRIC COMPUTER SYSTEMS(TOKYO) CORPORATION

導権を持ち、連続的にユーザの入力を促す場合がある。また、機能モジュールの実行結果が次の対話および処理の流れを決定する場合もある。それには各ルールを連続的な関係で捉えて記述しなくてはならない。

ルールを記述する上で、第一に問題となるのはこの連続的な対話制御をどのように実現するかである。開発の第一段階では、連続的な制御をまずルールの優先順位とフラグによって定義することに注力した。

その結果、以下の問題が生じた。

(1) ひとつの状態名におけるルール数が多くなり、システム全体の動きが捉えにくくなった。

(2) フラグの数が多くなり、セット/リセットが複数のルールに亘って、複雑になった。

3.2 解決策

先述のように、本UIMSのベースであるPSはアプリケーションの状態によって、ルールの検索範囲を絞り込むアルゴリズムを採用している。

そこで第一のアプローチとして、現在のルールにおけるアプリケーションの状態を細分化し、ルールのグループ化を行った。その際、以下の点を基準とした。

(1) ユーザの一入力がひとつの機能に対応している(連続的な対話制御がない)場合は、現在の状態名を変更しない。

(2) ひとつの機能を実行するのに、複数のルールが必要な(処理の結果が対話の制御に影響する)場合は新しい状態名をつけてまとめる。

(3) 多くの機能に共通的なルールは共通の状態名でまとめる。

その結果、アプリケーションの機能に対応してルールをグループ化できた。また、フラグが減少して複雑さが解消された。この二点によるメリットは、ルールの可読性が良くなったということである。

4. 記述の拡張

だが、ここに新しい問題が生じた。改善前はアプリケーションの状態を、図2に示すように状態A→状態B→状態Aのような単純な遷移で把握できた。しかし、アプリケーションの状態によるルールのグループ化を推し進めた結果、各状態間での複雑な遷移を意識する必要が生じた。例えば、状態A,状態Bから状態Cへは遷移するが、状態Dから状態Cに遷移することはないというような制御が要求される。また、改善前は状態Bのルールを実行した後は、一意に状態Aに戻せばよかった。しかし、状態の遷移が多様化すると、ある状態への入口が複数存在するため、状態を戻す場合に動的な対応が必要となる。従来の記述能力では、以下の点が問題となる。

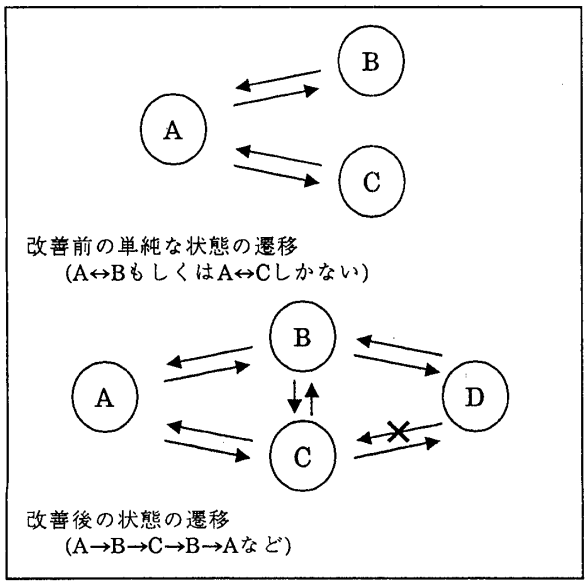


図2 遷移の多様化

(1) 同じルールを実行可能な状態数分用意しなくてはならない。

(2) 各ルールの実行後は明示的に状態を変化させるため、"前の状態に戻す"場合は、前の状態名をルール内で意識して保存しておく必要がある。

これらの記述上の煩雑さを解消するために、ルールの記述に対して以下の拡張を行った。

(1) ひとつの記述ルールに、アプリケーションの状態名を複数記述できるようにした。

(2) ルールを実行しないアプリケーションの状態名を記述できるようにした。

(3) インタプリタが前の状態を保存しておき、ルールには前の状態に「戻る」という記述を可能にした。

(1),(2)の拡張は、アプリケーション内に状態が多くなった場合に有効な記述である。ルールの実行を可能にする状態はAとBだと考える場合と、Cの時だけ実行できないと考える場合の双方のアプローチが素直に記述できる。

(3)の拡張は、状態の遷移に対して割り込みを可能にする。また、それ以上にシステムがアプリケーションの状態履歴を保存しておくことは、undo処理(直前の状態に戻す処理)への対応や、エラー処理の一元化等の発展性を持っている。

5. おわりに

今回の開発においては、ルールの記述にグループ化が有効なことがわかった。しかし、ルールのグループを細分化しすぎると、対話の制御はむしろ複雑になる。比較的抽象度の高い機能によってグループ分割することで、各グループの依存度を最小限にとどめることが重要である。

また、今後の課題であるルール記述を支援するツールには、モジュール化の概念を取り入れることと、各ルール間の関係を視覚的に表現する機能が要求される。

参考文献

[1] 宮崎 ほか 「プロダクション・システムに基づいたユーザインタフェース管理システム」
「文書処理とヒューマンインタフェース研究会」1989

[2] 福岡 ほか 「ユーザインタフェース・システム -プレゼンテーション・コンポーネント-」
情報処理学会 第35回全国大会 1987