グラフィカルエディタ作成用ツールキット GIST (2)
1D－2 — 対話処理の実現方法 —

Paul C. Brown　Terry M. Topka
General Electric Company, Corporate Research and Development
井上 健　土田 崇　上原 みな　○村田 なつめ　渡邊 多恵子
横河電機株式会社 オープンシステム研究所

1 序

GISTは対話型アプリケーション，特にグラフィカルユーザインタフェース(GUI)を用いたアプリケーションの開発を支援するためのものである．GUIはマウスなどのポインティングデバイスを用いてグラフィックスと対話する視覚的な操作環境であり，キャラクタディスプレイの時代に比べ，ユーザにとっては画期的にLook&Feelで扱いやすくなった．しかし，2次元のグラフィック平面上にユーザがインタラクションする場合の数は，飛躍的に増大し，そのインタフェース（対話処理）を設計することの複雑さは増大している．

複雑なアプリケーションではその設計をおこなうために状態遷移図を用いて動作を表現する必要がある．GISTでは，オブジェクトの構造化された状態に応じたアクションを記述できる枠組を用意し，対話処理の構築を容易にした．本稿ではGISTにおける対話処理の実現方法について説明する．

2 イベント処理とコントローラ

GISTの全てのユーザインタフェースは，コントローラと呼ばれるクラスのオブジェクトがつかさどる．キーボードやマウスなどの入力デバイスによるユーザからのアクションはGISTのイベントオブジェクト発生という形態をとり，コントローラに渡される．イベントを受けとると，コントローラは自分の状態に応じたアクションを実行し，ディスプレイなどの出力デバイスを通して，文字，イメージ，グラフィックスとしてユーザに提示したりする．

GIST — A Toolkit for Graphical Editor Development
Paul C. Brown, Terry M. Topka
Corporate Research & Development, General Electric Company
P.O.Box 8, Schenectady, NY 12301, USA
Takeshi Inoue, Takamu Tsuchida, Mina Uehara
Natsume Murata, Taeko Watanabe
Open Systems Laboratory, Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi, Tokyo, 180 Japan

コントローラはイベントを解釈するオブジェクトである．コントローラの動作を，コントローラの状態と，その状態におけるイベントの解釈の記述により構造的に記述するために，GISTでは簡単なイベント解釈言語EIL(Event Interpretation Language)とその処理系を用意した．EILによる対話定義の記述は，コントローラに対して状態名を構造化して記述しながら，各状態が受けとるイベントとそれに対する操作を定義することによりすすめられる．

2.1 対話定義の基本書式

EILによる対話定義の基本書式を示す．

```
STATE stateName IS
   ENTER {code to be executed when the state is entered}
   EXIT {code to be executed when the state is left}
   TARGET {code to be executed
           when the state is the target state}
TRANSITIONS
   eventName1 [condition] => nextStateName
     {code to be executed}
   eventName2 ...
```

この定義はstateName（状態名）という状態のとき，eventName1（イベント名）というイベントが発生すると，[condition]（条件）が真の場合は，まず{code to be executed}（実行すべきコード）を実行し，次にnextStateName（次の状態名）という状態に遷移することを表す．実行コードはC++で記述する．

ある状態に遷移するどのようなイベントであろうと，また前の状態や次の状態がどのような状態であろうと，状態に入ったり出たりするたびに特定のコード（アクション）を実行させたい場合はENTER節とEXIT節で定義する．また，この状態が状態遷移の最終到達先である場合に特定のコードを実行させたい場合はTARGET節で定義する．

一つの状態をいくつかの子状態（SUBSTATE）に分割する場合は，共通のイベント処理を親の状態遷移定義の中に記述し，子状態に固有のイベント処理をそれぞれの子の状態遷移定義の中に記述する．

階層化された状態の場合 (図 1)，イベントが発生すると，現在の状態の状態遷移定義を調べ，このイベントに対し定義されている応答があるかどうかを判断する．もしなければ，次に親の状態遷移定義を調べ，それでもなければ... というように調べていき，状態の階層のルートまで上って行く．調べたどの状態遷移定義でも応答が定義されていない場合には，そのイベントは無視される．

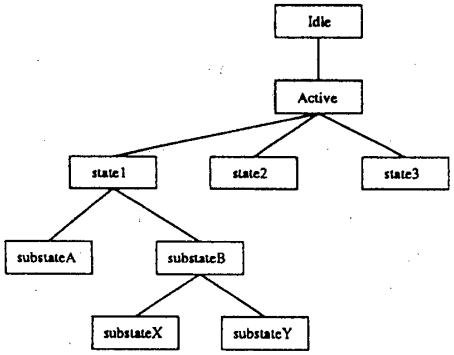

図 1　階層を持つ状態の例

2.2　コントローラ

EIL で記述された処理はコントローラオブジェクトの動作として実現され、アプリケーションは複数のコントローラの協調動作により，ユーザとの対話制御を行なう．アプリケーション中のコントローラは GIST で提供しているコントローラクラス（GeController）のサブクラスであり，そのすべてのコントローラはこの基本コントローラクラスの基本動作を共有する．

一つのコントローラを単純化すると図 2 のように見える．

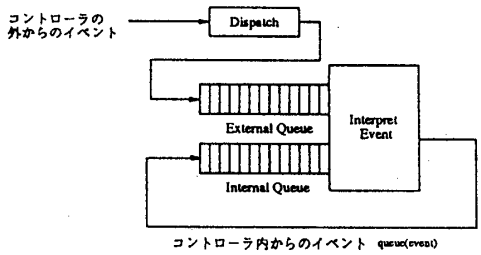

図 2　コントローラの内部構造

アプリケーション設計者は GIST で提供している関数 `GwcreatePaneEvent()` を使用して，ユーザのアクションなどに対応したイベントオブジェクトを生成する．この関数は，最初に全てのイベントを受けとるアプリケーションのメインコントローラであるルートコントローラの `dispatch()` メソッドを呼び出す．これによりイベントはルートコントローラの待ち行列に置かれる．その後，イベント処理メインループ（`GwmainLoop()` 関数）で，ルートコントローラの `processQueues()` メソッドを呼び出し，待ち行列からイベントを取り出して解釈し，EIL で記述されたアクションを実行する．

複数のコントローラを使ってアプリケーションの動作をモジュール化した方が都合が良い場合は、コントローラを階層化したり，スクリーニングコントローラを用いる．

互いに独立して動作する複数のウィンドウを持つアプリケーションで，各々のウィンドウを別々のコントローラで管理したいような場合には，各々のコントローラをルート (親) コントローラの子供とする．親コントローラは，イベントを子にあたる適切なコントローラにディスパッチして，処理を各々に任せる．GIST には，図形の移動やリサイズあるいはテキストの編集といった機能をつかさどるコントローラが用意されており，これらを子コントローラの一つとして利用できる．

スクリーニングコントローラは，元のコントローラを遮蔽し，外部イベントの解釈を元のコントローラより先に行なう．スクリーニングコントローラを用いると，イベントをディスパッチする方法を変更せずに，コントローラの動作を簡単に変更することができる．

3　まとめ

対話的なアプリケーションを設計する際のひとつの大きな課題である，ユーザからの入力に対して，アプリケーションがどのように内部状態を変化させ，どのようにユーザに対して出力を呈示するかという問題にたいし，GIST で採用した対話処理記述方法について述べた．EIL による対話処理の記述は，動作定義をモジュール化して，アプリケーションの対話制御を構造化した枠組の中で実現している．

今後は，プログラマではない対話設計者でも簡単に利用できる，グラフィカルな対話処理定義ツールの開発を検討したい．

[参考文献]

[1] James Rumbaugh, “State Trees as Structured Finite State Machines for User Interfaces”, Published in ACM SIGGRAPH Symposium on User Interface Software, October 1988.

[2] David Harel, “On visual formalism”, Communications of ACM 31,5,pp.514〜530, May 1988.