3M－4　オブジェクト指向分散環境OZ++のOZ++/Frameコアシステムの設計

中川 祐*(富士ゼロックス情報システム)　中村 章人 (電子技術総合研究所)
塚本 享治 (電子技術総合研究所)
*: 開放型基盤ソフトウェアつくば研究室研究員

1　はじめに

オブジェクト指向分散環境 OZ++ は分散環境におけるソフトウェアの開発と利用を容易にすることを目的としたオブジェクトの共有と交換に基づくシステムである。[1]

OZ++/Frame は初級ユーザに対して容易にアプリケーションを開発できることを目的としたツールである [2]。これは GUI を利用することが前提である。当然 OZ++ プログラムに対して GUI の機能を提供するためのライブラリが必要であるが、プログラマが GUI を利用したアプリケーションを開発するにはこの部分だけを提供することも有用であることに気づいた。そこで今回コアシステムというものを設計した。これは OZ++ プログラマに対して GUI を提供するためのライブラリである。これは OZ++/Frame の一部であるが、この部分だけを独立して用いることも可能である。本稿ではコアシステムの設計について説明する。

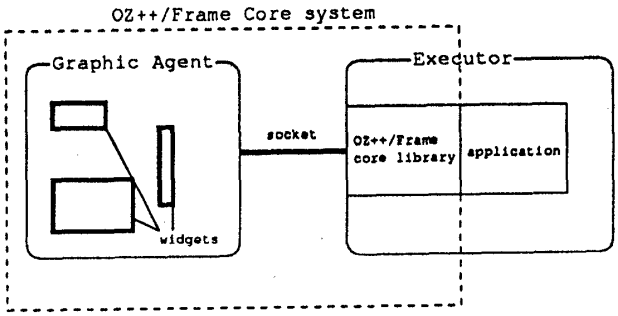
2　コアシステムの設計

2.1　基本設計

GUI ライブラリにはさまざまな形態が考えられる。今回は Widget キットを提供することにした。これは使い方を理解するのが比較的容易であること、多くの場合に十分な機能を提供し得ること、既存システムを利用することにより開発時間が短縮できることが理由である。

またハイパーカードを参考に Widget を保持するためのスクリーンとデータを保持するためのスライドという概念を導入することとした。これはカード型データベース等を構築する際に有効である。

2.2　コアシステムの構成



OZ++ は既存 OS 上で専用カーネルを実行しその上でオブジェクトを動作させるミドルウェアの手法を採用している。このカーネルをエグゼキュータと呼ぶ。エグゼキュータの実装の都合上 X-lib を直接リンクできない。このため実際に描画を行ないイベントをハンドリングする部分はエグゼキュータ上ではなく別プロセスで実装される。このプロセスをグラフィックエージェントと名付けた。

グラフィックエージェントはエグゼキュータから fork されその標準入出力ストリームは socket-pair にバインドされる。グラフィックエージェントとエグゼキュータは socket-pair を通じて通信を行なう。

グラフィックエージェントおよび socket-pair はクラスライブラリを提供することにより利用者に対して隠蔽される。このライブラリをコアシステムライブラリと呼ぶ。

3　グラフィックエージェント

コアシステムをグラフィックエージェントとコアシステムライブラリに分割した時点で双方にどのような機能を分担させるかが、設計で重要となる。グラフィックエージェントを単純な描画命令の集合にするとコアシステムは拡張性に優れる反面、コード量が増え動作が遅くなってしまう。今回はグラフィックエージェントに Widget を定義させ、Widget に対するサービスをグラフィックエージェントが提供するという方針を採用した。

そこでグラフィックエージェントの実装には tcl/tk を採用することとした。tcl/tk のシェルである Wish がそのままグラフィックエージェントに対応する。また

Design of core system of OZ++/Frame in OZ++ :
An Object-Oriented Distributed Systems Environment
Yu Nakagawa*(Fuji Xerox Information Systems, Co., Ltd.),
Akihito Nakamura(Electrotechnical Laboratory),
and Michiharu Tsukamoto(Electrotechnical Laboratory)
*:Researcher, TsukubaLaboratory, Open Fundamental Software Technology Project

Widget は tk をそのまま利用できる。さらにコアシステムライブラリからグラフィックエージェントへの通信はインタプリタである tcl そのものを利用している。

Widget はまずラベル、ボタン、チェックボタン、フィールドを作成した。

4 コアシステムライブラリ

OZ++ ではクラスのインタフェースと実装を個別に扱っている [3]。あるクラスの顧客クラスはそのインタフェースのみを押え、実装の決定は環境により決定される。

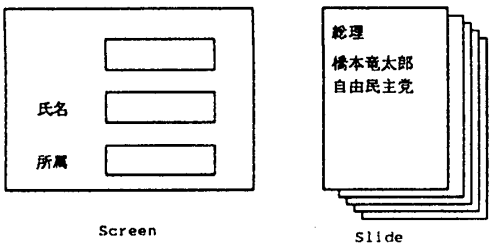
コアシステムライブラリの場合そのインタフェースと実装は異なる意味を持つ。インタフェースは OZ++ プログラムに対して GUI を用いるための枠組を提供する。対して実装はグラフィックエージェントに依存した作りとなる。すなわちグラフィックエージェントの修正による OZ++ プログラムの変更はすべてコアシステムライブラリの実装部分が請け負う。

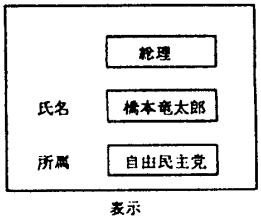
4.1 スクリーンとスライド

OZ++/Frame の特徴としてスクリーンとスライドの概念を採り入れたことが挙げられる。これはハイパーカードを参考にしたものである。

スクリーンは Widget を保持し、その座標を把握している。スライドは Widget に対応するデータを保持している。例えば住所録を作る場合氏名、住所、電話番号といったデータはフィールドに記述する。住所録のスクリーンは 1 つだけ存在し、フィールドを持ち、それをどの位置に表示すべきか知っている。スライドは住所録のデータ件数だけ存在し、各人の氏名、住所、電話番号を保持している。しかしそれらのデータをどの位置にどのように表示させるかについては関知しない。

コアシステムはスライドを順序付で保持している。そしてその中から 1 枚だけを表示することができる。スライドはスクリーンを 1 つ持つ。表示はスクリーンにスライドを与えることにより可能となる。





4.2 Model

Model はアプリケーションとコアシステムライブラリを結び付けるためのクラスである。Model は抽象クラスでありその具象クラスはアプリケーションで用意する。Model は Widget、スライドあるいはスクリーンにバインドすることができる。Model は想定されるイベントに対応したメソッドが用意されている。

例えばボタンが押下された場合、まず Button クラスに通知される。Button は自身の Model を探す。まず自身にバインドされている Model があればそれを、なければ自身を保持しているスクリーンにバインドされている Model を求める。求めた Model に対してボタンが押されたというイベントを通知する。

1 つの Model は複数の Widget に対してバインドされ得る。イベントメソッドにはメソッドをインボークしたオブジェクト自身が渡されるため Model はどの Widget のイベントであるか判別することができる。

5 まとめ

現在コアシステムの第一版が完成し、評価を行なっている。今後コアシステムの上に、オブジェクトのコピーと属性の変更機能を提供して、初級ユーザのアプリケーション作成を可能とする OZ++/Frame ミューテータサブシステムを構築する。また表計算を行なう OZ++/Chess を作成する。

本研究は、情報処理振興事業協会 (IPA) の「開放型基盤ソフトウェア研究開発評価事業」の一環として行なわれたものである。

参考文献

[1] 西岡他：「オブジェクト指向分散環境 OZ++ システム第一版の実現」,Swopp '95, Aug.1995

[2] 中村他：「オブジェクト指向分散環境 OZ++ のエンドユーザプログラミング環境 OZ++/Frame」, 情報処理学会マルチメディア通信と分散処理研究会, Nov. 1995

[3] 新部他：「OZ++ コンパイラによるクラスの版管理」,Swopp '94, Aug.1994