システムソフトウェアと
オペレーティング・システム 71－9
マルチメディア通信と分散処理 73－9
（1995．11．30）

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

中村章人 (電子技術総合研究所) 中川 祐 * (富士ゼロックス情報システム)
石川雅基 * (システムトゥエンティ・ワン) 音川英之 * (シャープ株式会社)
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

オブジェクト指向分散環境 OZ++ は，分散環境を利用した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．本論文では，OZ++ のエンドユーザ向けプログラミング環境 OZ++/Frame の設計と実装について述べる．ユーザは，ボタンやフィールド等のユーザインタフェース (UI) 部品の組み合わせによる UI の作成と，各 UI 部品上のイベントに対する処理プログラムを記述することで，アプリケーションを作成する．OZ++/Frame を利用することで，ユーザ間では，データだけでなくアプリケーションの UI も含めて共有・交換できる．また，ソフトウェアモジュールを公開するための仕組みであるカタログと OZ++/Frame とを組み合わせて利用することで，アプリケーションを構成する部品プログラムの公開と共有を，ネットワークワイドに行える．

OZ++/Frame: An End-User Programming Environment of Object-Oriented Distributed Environment OZ++

Akihito Nakamura (Electrotechnical Laboratory)
Yu Nakagawa* (Fuji Xerox Information Systems) Masaki Ishikawa* (System Twenty One)
Hideyuki Otokawa* (Sharp) Michiharu Tsukamoto (Electrotechnical Laboratory)

* Researcher, Tsukuba Laboratory, Open Fundamental Software Technology Project

Object-oriented distributed environment *OZ++* is a programming environment that makes easy to develop and to execute distributed software, based on transferring and sharing of classes and objects in a distributed environment. This paper presents the design and implementation of an end-user programming environment *OZ++/Frame* on OZ++. In the OZ++/Frame, users can develop application software with graphical user interface easily by placing and arranging components like buttons and fields on the window. Components can be taken from software module database called *catalogue*, and the behavior of components can be customized by users. By using OZ++/Frame, users can exchange not only data but also the user interface of them. In addition, by cooperating with the software module database *catalogue* of OZ++, it becomes easy to open developed application software and/or parts of them to the public, and to share such software in the network.

1 はじめに

OZ++ は，オブジェクトの交換と共有に基づく分散処理環境である．OZ++ が提供する分散環境とは，ネットワーク上に広く分散した計算機間で，クラスとオブジェクトを相互に利用し合うことを可能にする分散プラットフォームである [2]．このプラットフォームには，分散アプリケーションを開発するために，OZ++ 言語が用意されている [3]．

OZ++ では，OZ++ 言語を直接利用したアプリケーション開発のための環境としてワークベンチ [5] がある．ワークベンチは，プログラム中のクラス名と OZ++ がクラスを識別するためのクラス ID との対応表であるスクールの管理を主として行い，コンパイラの起動，クラスのブラウズ，クラスのバージョン管理等の機能を提供する．また，カタログというモジュールデータベースとのインタフェースを持っており，開発したソフトウェアをカタログに登録して公開する機能を持っている．

OZ++ 上のソフトウェア開発という観点では，分散したクラスの共有に主眼が置かれている．これに対し，アプリケーションプログラムの利用という観点では，データであるオブジェクトをいかに容易に作成し，かつ容易にそのオブジェクトにアクセスできるかが重要である．しかし，アプリケーションプログラムを利用して業務を行うユーザ (エンドユーザと呼ぶ) は，ソフトウェア開発の専門家でない場合が多い．エンドユーザにとっては，ワークベンチを利用した OZ++ 言語でのプログラム開発は困難である．**OZ++/Frame**(以下フレームと書く) は，このようなエンドユーザ向けのアプリケーションプログラミング環境である．

フレームの目的は以下の三つである．

- エンドユーザのアプリケーションプログラミングを支援する．
- アプリケーション間の連係を容易にする．
- アプリケーション間で統一されたユーザインタフェースを提供する．

フレームは，オブジェクトの構造を制限したフレームワークを提供することによって，エンドユーザでも理解しやすい直観的で統一された構造のアプリケーションの作成を支援する．ユーザは，アプリケーションの部品となるクラスを自分で作成することもできるし，カタログに公開されている部品を利用してそれらの組み合わせでアプリケーションを開発することもできる．また，開発したアプリケーションまたはその部品は，カタログを介して公開することができ，それらの再利用はカタログとのインタフェースを利用して容易に行える．

フレームで作成したアプリケーションは，それを構成しているオブジェクトの構造が統一されているので，アプリケーション間の連係を比較的簡単に行える．フレーム間でデータ (オブジェクト) のやり取りを容易に行えることに加えて，それらの操作方法 (UI) も一緒に共有・交換できる．

フレームは，アプリケーションの部品となるクラスをグラフィカルなユーザインタフェース (UI) 部品と一体化して提供し，ユーザは，ウィンドウ上でこれらの部品を組み合わせて UI を作成するとともに，ユーザからのイベントに対する処理を記述することで，アプリケーションを作成していく．

エンドユーザ向けのプログラミング支援を目指したシステムに，IntelligentPad[1] や Apple 社の HyperCard 等がある．これらのツールは，単一計算機上のアプリケーション作成を支援するものであり，フレームのように分散環境でのアプリケーション作成を目指したシステムはほとんどない．また，遠隔のツール間でのデータまたはオブジェクトのやり取りや，部品を公開する仕組みを提供しているものはない．フレームは，OZ++ が提供するクラスの配送とオブジェクトの転送機能を利用し，分散したフレーム間でオブジェクトをやり取りできることと，カタログを利用して部品の公開と共有を支援する仕組みを提供していることが特徴である．

本論文の構成は以下のとおりである．2 章では，フレームの構造と機能を示す．3 章では，フレームの実装について述べる．4 章では，現状の問題点と今後の課題について述べる．

2 フレームの構造と機能

本章では，フレームの構造と機能について述べる．

2.1 アプリケーションの構造化

フレームは，OZ++ のアプリケーションプログラムであると同時に，アプリケーションを開発する環境でもある．フレームで開発できるアプリケーションの構造は，エンドユーザでも理解しやすい単純なものにした．しかし，この構造は，多くのアプリケーションに利用できる汎用的なものになっている．フレームで作成したアプリケーションはシリーズというオブジェクトとして実体化される．

フレームで作成したアプリケーションは，4 階層の部品オブジェクトから構成される (図 1)．シリーズが最上位，ボタン，フィールド，図形が最下位のオブジェクトとする．各枝は，上位のオブジェクトが下位のオブジェクトを部品として利用できることを意味する．それぞれは，アプリケーションの機能を構成する部品であると同時に，アプリケーションの UI を構成する部品としての役割も持つ．

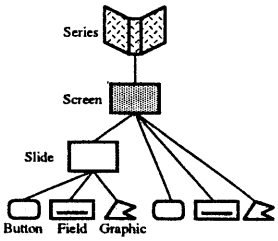

図 1: シリーズの構造

- ボタンとフィールドは，それぞれユーザからのマウスイベントの入力と，文字列の入力 / 表示を行うためのオブジェクトである．通常，これらのオブジェクト上に発生するイベントは，アプリケーションの特定の機能を起動するトリガとして利用される．図形オブジェクトには，直線，多角形，円，ビットマップ等の種類がある．
- スライドは，ボタン，フィールド，図形を複数保持するオブジェクトである．ディスプレイ上では，一定の大きさの矩形である．
- スクリーンは，一つ以上のスライドに共通して利用できるフォームのようなもので，スライドと同様にボタン，フィールド，図形を複数保持できる矩形である．
- シリーズは，スライドとスクリーンを複数保持し，スライドについては順序付けて管理している．

シリーズの UI の表示およびイベントの伝達は，フレームによって実行される．フレームは，各シリーズについてディスプレイ上に一つのウィンドウを用意し，選択された一つのスライドをこのウィンドウに表示する．このとき，まずこのスライドに対応するスクリーンが表示され，スライドはこのスクリーン上にスーパインポーズされる．各スライドに対して，対応するスクリーンは必ず一つだけある．(名前が示すように，「スライド」は透明で透けて見え，スクリーンはスライドを投影するまさに「スクリーン」である．)

各部品オブジェクトは，それぞれフレームが提供するある決められた抽象クラス (基本部品クラス) の子孫クラス (ユーザ定義部品クラス) のインスタンスとして生成される．各基本部品クラスで定義されていて，ユーザ定義部品クラスでユーザが実装を再定義するメソッドをユーザ定義メソッドと呼ぶ．例えば，「次のスライドに切替える」というボタンは，ボタンの基本部品クラス `Button` を継承し，ユーザ定義メソッド `MouseUp` を再定義したユーザ定義部品クラス `ButtonNext` のインスタンスである．

図 2は，メールクライアントをシリーズとして作成した例である．ここでは，一つのスライドが一通のメールに対応し，送信者や内容等のデータを保持するためのフィールドを持っている．スクリーンは，いくつかの制御ボタン (reply, print 等) を持っており，各スライドで共通に利用する．このシリーズをウィンドウに表示するときには，スクリーン上に一つのスライドがスーパインポーズされ，ユーザには (1) のように見える．

2.2 フレームの機能

本節では，フレームの機能について述べる．

2.2.1 カタログを利用した部品の公開と共有

エンドユーザにとっては，アプリケーションの部品となるクラスでさえも自分で開発するのは困難である場合が多い．フレームの重要な機能の一つは，ユーザが利用したい部品を簡単に入手できる仕組みを提供することである．また，ユーザが新規に作成または改良した部品を，公開して再利用できるようにすることである．OZ++ では，カタログを介してソフトウェアの公開を行う．カタログは，ソフトウェアモジュール (クラスの集合) に階層的な名前を付与して管理するデータベースで，ネットワークワイドにアクセス可能である．

ユーザは，カタログブラウザを利用してネットワーク上のカタログから望みの部品を探し，部品オブジェクトを生成する．ユーザが自分で作成した部品クラスを公開したい場合は，その部品に名前を付けてカタログに登録する．フレームから公開できる部品は，シリーズ，スクリーン，スライド，ボタン，フィールド，図形である．例えば，図 2のメールクライアント全体をシリーズとして公開できるし，ボタンだけを公開することもできる．

各部品の機能としての役割は，クラスとしてカタログを利用して公開できるが，フレームのようなシステムでは部品のビューも一緒に公開できる必要がある．例えば，「次のスライドに切替えるボタン」のクラスをカタログに登録して公開するときに，矢印 (→) や「Next」という文字列をラベルとして付けたボタンとして公開したい．こうしておけば，部品を再利用するときのインタフェースをグラフィカルにして，ユーザの検索を効率良くし，部品のビューも共有・再利用できるようになる．

OZ++ のカタログは，クラスを公開する仕組みを提供している．部品のビューを公開するために，カタログを拡張する方法と，インスタンスのデフォルト値としてクラスに埋め込む方法が考えられる．現在は，実装が簡単な後者の方法を採用している．

2.2.2 分散環境での利用

OZ++ の特徴の一つは，ネットワーク上でオブジェクトを転送できることであり，送り先にクラスの実行可

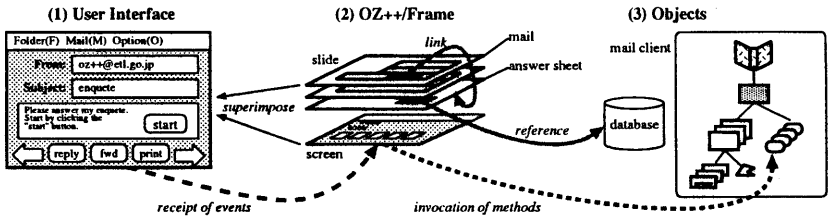

図 2: OZ++/Frame の構成と動作

能コードがなくても OZ++ のクラス管理システムが自動的に配送する．フレームは，シリーズ間でのスライド / スクリーンの転送およびコピー機能と，フレーム間でのシリーズとスライド / スクリーンの転送およびコピー機能を提供している (図 3)．すなわち，カタログを利用したクラス (プログラム) としてのアプリケーションの公開に加えて，オブジェクト (データと UI) を共有できる．

例えば，フレーム間でシリーズを指定したスライドのコピーや，スクリーンとそれを利用するすべてのスライドの一括コピー等を行える．ここで，スライド等の部品オブジェクトを転送するときに，転送先にそのクラスが存在するかどうかをユーザは意識する必要がない．また，オブジェクトは，その構造を保ったまま転送されるので，ある部品オブジェクトを転送するとその下位の部品もいっしょに転送される．そのオブジェクトのメソッドが起動されたときに，クラスがそこにないならば，OZ++ のクラス管理システムにより自動的に配送される．

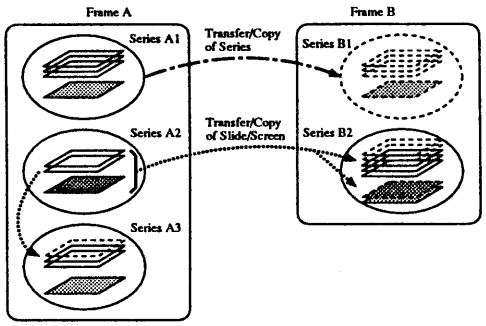

図 3: シリーズとスライド / スクリーンの転送 / コピー

分散環境でフレームを利用する例として，図 2のメールクライアントを用いて「メールの中にアンケートを同封して送り，その回答結果をデータベースに格納する」というメールを作る場合の手順は次のようになる．まず，メールとアンケートの解答用紙をそれぞれ一つのスライドとして作成する．メールのスライドに解答開始ボタンを付けて，このボタンから解答用紙へのリンク (2.2.3参照) を張る．また，解答用紙のスライドには解答終了ボタンを付けて，この中でデータベースのデータ追加メソッドを起動するようにユーザ定義メソッド `MouseUp` を再定義する．そして，メールのスライドを相手先のフレームのメールクライアントに転送する．

このとき，スライド上のボタンやフィールドは，すべて構造を保ったまま転送され，コピー先でもまったく同じビューが得られる．また，データベースの参照と解答用紙へのリンクも保存され，受信側で解答開始ボタンを押せば解答用紙が現れ，解答終了ボタンを押せばデータベースへ解答結果が格納される．ここで重要なのは，フレームやデータベースがネットワーク上に分散していても，同様に動作することである．OZ++ では，ネットワーク上で一意な識別子を持つオブジェクトを生成し，これをネットワークワイドに参照できる機構を提供しているからである [2].

2.2.3 ハイパーリンク

フレームは，シリーズの部品オブジェクト間にリンクを定義できる仕組みを提供している．各部品オブジェクトからスライドとシリーズへのリンクが可能で，スライドへのリンクはそのスライドへの切替え，シリーズへのリンクはそのシリーズの起動の意味を持つ．例えば，ボタンからスライドへのリンクを張ることで，そのボタンを押したときに現在表示されているスライドがリンク先のスライドに切替わる．

ボタンからスライドへのリンク以外にも，フィールド内の文字列の一部からスライドへ，ボタンからシリーズへ，図形からスライドへといった様々なリンクを定義できる．このリンク機能を利用することで，内部のデータに複雑な参照関係を持ったアプリケーション (例えばハイパーテキスト) を容易に作成できる．

2.2.4 UI とオブジェクトの結合

フレームの重要な機能の一つは，フレームで作成したアプリケーションを実行することである．すなわち，UI を表示し，UI 上のイベントとシリーズ内の部品オブジェ

クトのメソッド起動を結びつけることである．例えば，図2の例でユーザがあるボタンを押した場合，まずこのイベントがUIからフレームへ伝達される．フレームは，押されたボタンのオブジェクトに対して，メソッド`MouseUp`を起動する．

ある基本部品クラスで定義されているユーザ定義メソッドは上位の基本部品クラスにも含まれており，ここでもユーザが再定義できるようになっている．すなわち，下位の部品オブジェクトのユーザ定義メソッドから上位オブジェクトの同じユーザ定義メソッドを起動することで，イベントを上位オブジェクトに伝達できる．

フレームのようなアプリケーションは，ユーザが対話的に利用する場合が主であるが，別なアプリケーションからUIを使わずに利用する場合も考えられる．これは，シリーズ内のデータを別なアプリケーションから利用する場合や，あるアプリケーションで作成したデータをシリーズに格納する場合などが考えられる．フレームは，UIを通して提供している機能をパブリックなメソッドとしても公開している．これらのメソッドは，シリーズ内の部品オブジェクトからも，フレームの外から，すなわちまったく別のアプリケーションからも起動できる．言い替えると，これらのメソッドはフレームのAPIであり，フレームは様々な利用形態を支援している．

3 フレームの実装

本章では，フレームの実装とその動作について述べる．

3.1 オブジェクト

フレームの主要機能は，フレームコアというオブジェクトとして実装した．フレームコアは，その構成オブジェクトとして，プロジェクタおよび0個以上のシリーズを持つ(図4)．

プロジェクタは，UIとフレームコアとの仲介をする．UIからイベントを受け取ってフレームコアのメソッドを起動し，フレームコアからのメソッド起動でUIの操作を行う．2.2.4で述べたように，UIを通して利用できるフレームの機能は，フレームのパブリックなメソッドとして公開し，これらのメソッドはフレームのAPIとしてフレームコアに実装した．UI部分を分離することで，フレーム以外のアプリケーションからフレームを利用できるようにした．

例えば，前述のメールにアンケートを同封し解答結果をデータベースに格納する例では，このデータベース自身をシリーズとして作成してもよい．別の集計アプリケーションからフレームのAPIを利用して，このデータベースにアクセスできる．

3.2 ユーザインタフェース

OZ++ でのUIの実装には，OSプロセスを起動してそのプロセスでUIを提供するという方法をとる．すなわち，これまではUI(またはその部品)をオブジェクトとして扱うことができなかったが，フレームによってUI部品をオブジェクトとして扱う枠組が提供された．

フレームのUI部分は，Tcl/Tkを利用して実装した．フレームの起動時にプロジェクタオブジェクトがOSプロセスとしてTclのインタプリタwishを起動し，Tclスクリプトを実行する．これ以降は，wishからプロジェクタのイベントディスパッチャへは文字列のコマンドが送信され，プロジェクタはwishの標準入力へプロシージャ名と引数を文字列として書き込むことでTclのプロシージャを直接起動する．各部品オブジェクトに対応するUIは，Tkのウィジェットとして実装した．ボタンは`button`ウィジェット，フィールドは`text`ウィジェット，スクリーンは`canvas`ウィジェット，図形は`canvas`上のアイテム，スライドは`button`，`field`，アイテムのリストである．

OZ++ のクラス管理システムは，プログラムの実行に必要なクラス以外のファイル(例えばTclスクリプト)の配送も行える．フレームのTclスクリプトおよびビットマップファイルは，クラス管理システムによって自動的に配送される．

フレームの現在の実装では，OZ++ 言語で記述した部分が約2,000行，Tclで記述したUI部分が約2,000行となっている．フレームの実装に新たに開発したクラスの数は，約20個である．

4 課題

現版のフレームで検討中の問題，実装されていない機能は以下のとおりである．

- ビューの公開と共有
  2.2.1で述べたように，各部品の機能を記述するクラスはカタログによって公開しているが，ビューの公開はインスタンスのデフォルト値としてクラスに埋め込む方法を取っている．すなわち，あるクラスについて一つのビューしか公開できない．しかし，同じクラスで複数のビューを公開でき，ユーザは機能とビューの両方を再利用できるほうが望ましい．このためには，ビューを提供するクラスを分離して，カタログで公開することが考えられる．現在は，ユーザの選択範囲が拡大してしまい検索のコストも高くなるのでこのような設計はしていないが，今後の課題である．
- OZ++/Chess
  エンドユーザ向けのアプリケーション開発支援という観点では，フレームの汎用性に加えて，表計算や

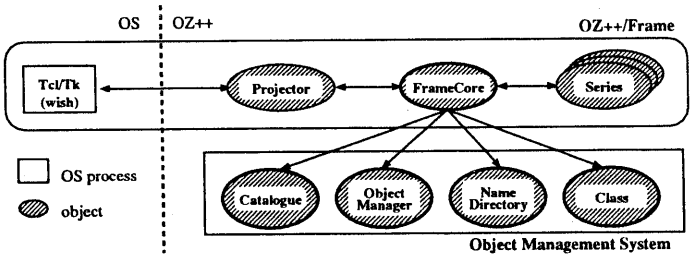

図 4: OZ++/Frame の実装

ワークフロー等の業務処理向けアプリケーションを容易に作成できることが重要である．このようなアプリケーションをシリーズとして作成し，カタログを利用して共有することもできる．しかし，このようなアプリケーション開発を，フレームが提供している部品だけで行うのは困難である．そこで，表計算やワークフロー等の業務処理向けアプリケーションを容易に作成できるようにフレームを拡張する OZ++/Chess を開発中である．

- アクセス制御
  現版のフレームは，アクセス制御機構を実装していない．OZ++ は，ユーザ認証のためにアカウントディレクトリを提供しており，これを利用したマルチユーザ環境でのアクセス制御機構を実装し，許可のないユーザによるシリーズの操作を排除する必要がある．フレームの API には，文字列による各オブジェクトの検索のためのインタフェース，スライドやスクリーンの編集，追加・削除等の更新のためのインタフェースがある．これらのインタフェースを複数ユーザについてアクセス制御するために，アクセスリスト等の実装を検討している．

- スキーマ変換
  OZ++ はクラスのバージョン管理機構 [4] を提供しており，パブリックなインタフェース (メソッドのシグネチャ) が同一のクラスは，外部参照して利用する限り，交換可能である．OZ++ では，クラスの共有と再利用を促進するために複数バージョンのクラスの共存を許し，スキーマ変換の仕組みを提供しない．このため，クラスのバージョンアップに伴い，旧バージョンで生成したインスタンスを利用できなくなる場合がある．フレームでは，オブジェクトの種類を制限しているため，オブジェクトのインスタンス変数の値をオブジェクト以外の形式で取り出し，これをまたオブジェクトにもどす仕組みを容易に実現できる．現在，ファイルを利用してこれを実現する方法を検討している．

5 まとめ

本論文では，OZ++ のエンドユーザ向けプログラミング環境である OZ++/Frame について述べた．アプリケーションを構成する部品クラスを分散環境で容易に公開・共有できる枠組を提供し，これによってエンドユーザでも容易にアプリケーションを作成できる枠組が提供できた．

また，アプリケーションを構成する部品オブジェクトの機能と UI の二つの側面を統合的に扱うことができる．これによって，アプリケーション間では，データだけでなくその操作方法 (UI) も含めたやり取りをネットワークワイドに行えるようになった．

本研究は，情報処理振興事業協会「開放型基盤ソフトウェア研究開発評価事業」の一環として行われたものである．

参考文献

[1] Tanaka, Y. and Imataki, T., "IntelligentPad: A Hypermedia System allowing Functional Composition of Active Media Objects through Direct Manipulations," *Proc. of the IFIP 11th World Computer Congress*, 1989.

[2] 塚本他，「オブジェクト指向分散環境 OZ++ の基本設計」，情報処理学会研究報告 93-OS-61(SWoPP 鞆の浦), 1993.

[3] 西岡他，「オブジェクト指向分散環境 OZ++ の言語の基本設計」，情報処理学会第 46 回全国大会, 1993.

[4] 新部他，「OZ++ コンパイラによるクラスの版管理」，情報処理学会研究報告 94-PRG-18(SWoPP'94), 1994.

[5] 音川他，「オブジェクト指向分散環境 OZ++ のプログラミングパラダイム」，情報処理学会研究報告 95-PRO-2(SWoPP'95), 1995.