情 報 学 基 礎 31－4
（1993．9．14）

クラスライブラリの分類枠組について

羽生田 栄一　　藤野 晃延

富士ゼロックス情報システム

再利用可能なクラスライブラリを用いることにより、ソフトウェア開発における生産性と品質の向上が期待されている。しかし期待された成果が実際に得られたという成功事例はそれほど多くなく、クラスライブラリを再利用するにあたっての問題点が幾つか浮かび上がって来ている。

このようなクラスライブラリを再利用するに当たっての問題点を解明し、再利用しようとする利用者に役立つクラスライブラリについての概要情報、あるいはプロファイルがいかにあるべきかを、既存の流通クラスライブラリについての調査を基に考察し、現実的レベルでクラスライブラリを流通促進するための試案として、クラスライブラリの分類枠組を策定した。

An ideal guideline for classification of class library

Eiichi Hanyuuda　　Terunobu Fujino

Fuji Xerox Information Systems

Since efficiency and usefulness of reusable class library is commonly and widely accepted as a way to improve software productibity and quality, there are still relatively a few succes stories are available which fully utilize the potential usefulness of a certain class library into the development of target application. Indeed, there seems some difficulties exist to make utilization of reusable class library happen in an application system development.

To overcome these difficulties, pursuits for an ideal model of profile or scheme of descriptive information concerning to class library, which helps user or consumer of the class library in a variety of way, has been done based on the survey of several class libraries currently available. As a result, a tentative guideline has been proposed for the purpose of realization of the distribution of class library in more real world sense.

1 はじめに

Meyer[10] が指摘しているように、クラスライブラリの主要な構成機構である「継承」は、クラスライブラリを「提供する者のための仕掛け」であって、それを利用しようとする者にとって必ずしも役立つな仕掛けとなっているわけではない。クラスライブラリの一層効果的な利用を目指し、利用する者の立場から、どのような情報が提供されればクラスライブラリの利用促進が可能となるのか、について以下に述べる「分類想定枠組第0版」を1つの規範として策定した。

このような考察を実施した背景には、以下の諸問題点が強く認識されていたからである。

(a) オブジェクト指向に基づいたクラスライブラリの開発／維持／管理、およびチーム開発の経験から、単純に**「個々のクラスを再利用する」**のと、**「クラスライブラリを使いこなす」**のとでは、全く違った知識／経験が必要である。

(b) 「クラスを再利用する」のは簡単、しかし「クラスライブラリを使いこなす」のは大変。特に経験の乏しいユーザにとって、後者を自力で克服することは非常な困難が伴う。(あるいは非常に time consuming)

(c) オブジェクト指向の恩恵を享受するには、「自分で作成する」という NIH (Not Invented Here) 症候群から脱却し、自分以外の人の作ったクラスライブラリを再利用することに長けなければならない。

(d) しかし、他人の作ったクラスライブラリを理解し、使いこなすには越えなければならない障害が多い。その障害 (difficulty of re-using) を一般的に少しでも軽減できないだろうか。

(e) そのためにはクラスライブラリに共通して存在し、しかし実際には明示的に示されていない「ある種の」認知的構造— **Super Structure(s)** — を明らかにする必要がある。

静的で単純な class 相互間の関係 (例えば super/sub class relationship など) を把握しただけではクラスライブラリは使いこなせない。明示的にはどこにも記述されていない class 群相互の動的な依存関係を、共通の基盤に従って明らかにすることにより、クラスライブラリを利用する側にとっては利用する際のガイドラインとなり、さらに クラスライブラリを提供する側にとっては他の クラスライブラリとの差異を明確にしたクラスライブラリの開発が可能となり、双方にとって意義あるものとなる。

この今までは「明示的に」されていなかった クラスライブラリに隠された固有な情報を明らかにし、クラスライブラリのより一層の流通と再利用性の向上を図ることを目的に、「クラスライブラリの分類枠組」を検討、策定した。

2 構成的分類枠組

クラスライブラリの利用者と作成者の両者の理解を助け、ライブラリの利用と構築を促進するキーとなるのは、クラスライブラリの構成を自然かつ多面的に表現できる枠組みの提供にあると考えられる。そのライブラリの構造とそのセマンティクスを自然に記述でき、ライブラリ構成や設計上の意図まで伝えられるだけの強力な表現力すなわち、ドキュメンテーション能力をもった枠組みである。

2.1 構成的アプローチの簡潔な解説

クラスライブラリの構成に着目してクラスライブラリを検討しようとした背景には、次のことを明かにしたいという意図があった。

- クラスライブラリを議論する視点には、クラスライブラリを利用する立場でクラスライブラリの選定方法について議論する視点と、クラスライブラリを構築する立場でクラスライブラリの構成を議論する視点が存在する。

- 前者の立場でクラスライブラリを議論する場合、あるアプリケーション領域（例えば証券・銀行システムというものがひとつのアプリケーション領域に対応する）に対して複数のクラスライブラリが提供されている状況が望ましい。しかし、現状では、ここまで多くのクラスライブラリが開発されている状況には無い。むしろ、今後、様々なアプリケーション領域に対してクラスライブラリを開発していく必要の方が高い。

- こうした中、今後開発されるクラスライブラリに対する設計のガイドラインとしてどのような機能、どのようなクラスが必要なのか、また既

存のクラスライブラリでは何が機能として欠けていてそれを提供するのにどのようなクラス群が必要なのか、を明かにしたいと考えた。

こうした意識から、我々はクラスライブラリの構成に着目して検討することとし、クラスライブラリの構成モデルとして以下に示すクラスライブラリの枠組を得た。

A. カテゴリ／レイヤ

B. RCSI アーキテクチャ

C. デザインパターン [7][8]

D. 実装パターン（抽象実装）[5]

2.1.1 構成的分類枠組みに関する定義

プログラムの部分を使いたいと思うだれかに対して、そのプログラム部分の構造と意図をどのように記述し伝達するかという問題を解決しなければならない。

我々が必要としているのは、

○ 様々なレベルの情報がその意図にしたがって構造化できる

○ その情報を読んで具体的な行動が起こせるほどの経験知識を実質的にコード化、構造化できるような枠組み

である。そこでは、what と how と why が人間の理解という認知形式に見合った形で適切に分配され構造化された記述を提供していなければならない。そのプログラムの基本的なサービス＝機能内容が示され、ごくおおざっぱにどのような動きをするのかが説明され、さらにそのプログラム断片の大局的な意図を比喩のレベルで理解できるとよい。

枠組みに関して別の言い方をすると、

> OOA-OOD-OOP の過程において、クラスライブラリとして用意される既存のクラス、および新規クラスが、どのように切り出され／利用されるかを整理して、クラスライブラリがどのように構築されるべきかの基準をさだめるもの

という見方もできる。各レベルは以下のように考えられる。

○ カテゴリ／レイヤ

クラスライブラリ構成の一般形を示し、ライブラリの位置付けを明確にする。

○ RCSI アーキテクチャ

問題解析の過程で表面化してくるクラス間の関係を明確にする。

○ デザインパターン

クラス設計作業におけるクラス間の関係のパターンを明確にし、クラス利用／構成の側面からの指針を示す。

○ 実装パターン（抽象実装）

クラス実装作業におけるクラス間の関係のパターンを明確にし、クラス利用／構成の側面からの指針を示す。

各レベルの関係としては、RCSI アーキテクチャによって分析されたクラス構成に対して、デザインパターン、実装パターンを参考にクラス構成を順次、確定していく。そして、カテゴリ／レイヤとして整理する形態をとる。

2.2 枠組みの各レベルの関係

前節で示したように、カテゴリ/レイヤ、RCSI アーキテクチャ、デザインパターン、実装パターンの四種の枠組は、それぞれある視点でクラスライブラリの構造を抽出した抽象である。ここではこれらの枠組をそれぞれレベルと呼ぶ。各レベルは異なる視点からなるが、それぞれは相互に関連している。といってそれぞれは緩やかな関連であり、おのおのある程度独立して作業を進められるように考慮している筈である。すなわち、クラスライブラリのある部分をこれら 4 つのレベルのそれぞれの観点でそれなりに表現することが可能である。ここでは、各レベルの個別の枠組みの説明に移る前に、これら 4 レベルの枠組みの関連について整理する。

○ カテゴリ／レイヤはそのクラスライブラリのアーキテクチャを見取り図として提示するためのおおざっぱなクラスの容器の設定である。各クラスをその利用目的（ドメイン）とそのクラスが表現するオブジェクトの抽象度から分類した枠組みと言える。分野やドメインをキーにするのはこのレベルの枠組みだけなので、その意味で他のレベルの枠組みと直交する関係にある。例えば、あるカテゴリのあるレイヤとその下のレイヤのクラスの関係が、ある実装パターン従っ

ているといったことが生じるし、あるレイヤ上の、あるカテゴリと他のカテゴリのクラスの関係が、RCSI アーキテクチャに従っているということも生じる。あるいは、あるカテゴリのあるレイヤに属している各クラスが全体としてあるデザイン・パターンに従っているということも生じる。

○ RCSI アーキテクチャは、ある概念や機能を提供するクラスをどのように認識し、どうクラス階層内で位置づけ、そのクラス階層内の各クラスをカテゴリ／レイヤにどう割り付けるかというクラス階層の認識／同定に関する枠組みである。特定の機能を提供するクラス群をその実現プロセスに着目して分類したものとも言える。複数のクラス群である概念を表現しようとするときの、その概念に対して各クラスが果たす役割を表現する枠組レベルなので、RCSI アーキテクチャに属している特定のクラス間の関係が、あるデザイン・パターンや実装パターンに従っているということも生じる。

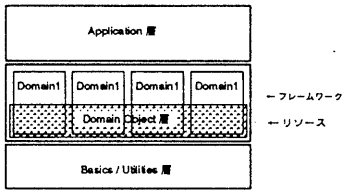
○ 一方のデザインパターンでは、クラス階層というクラス概念間の縦の構造ではなく、オブジェクト間の横の繋がりに着目し、そこに生じるパターンを抽出し、カタログ化し類似のパターンが見て取れることが多い。パターンとは、そうしたクラス群のある機能の提供にお け意味的役割、役割分担を抽象化したものと見ることもできる。従って、RCSI アーキテクチャに属している特定のクラス間の関係が、あるデザイン・パターンや実装パターンに従っているということも生じる。RCSI アーキテクチャレベルよりも、動的な側面、相互作用的な関係がより強調される。

○ さらに抽象実装／実装パターンという枠組みは、実装言語によらずに設計の下流過程でとりうる一般的な実装パターンとして「実現」という概念を抽象化しようというものである。上記の各枠組みレベルで認識、設定された各構成物を実際のソフトウェアとしてどう実装するかという問題を理想的な仮想のオブジェクト指向言語を設定することにより吸収することができるだろう。そのための手段として、各言語毎に提供しているオブジェクト指向メカニズムの洗練の度合いに応じて、その現実のメカニズムと理想的なオブジェクト指向機能とのギャップを埋めるための技法をリストアップしている。ここではC++をより理想的なオブジェクト指向言語として機能させるための技法を中心にまとめている。

2.3 枠組みの詳細

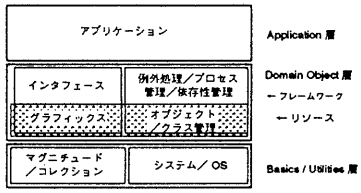
2.3.1 カテゴリ / レイヤ

図 カテゴリ／レイヤの概要図を示す。



カテゴリ / レイヤの概要

例 Smalltalk のクラスカテゴリの分類をマッピングした図を例として示す。



カテゴリ / レイヤの Smalltalk へのマッピング例

解説 クラスは図に示したように 3 つの層に分割される。以下に各カテゴリ／レイヤの説明を行う。

(a) Base/Utility 層
ライブラリを形成するために必須のクラス群のレイヤで、上位層において利用される基本的なクラス群およびシステム非依存性を保証するシステムインタフェース用のクラス群である。

(b) DomainObject 層
Base/Utility 層のクラス群を利用して構築されるドメイン向けクラス群を提供する。この層は対象とするドメインごとに分割され構成される。この層のクラス群はさらにドメイン毎にリソース／フレームワークというレイヤに分類される。リソースは、そのドメインにおいて利用される基本的なクラス群で、

フレームワークは、そのドメインのアプリケーションを作成する上で基本となる概念を提供するクラス群である。

(c) Application 層
ドメイン層のクラス群を利用して構築するアプリケーション固有のクラス群である。

2.3.2 RCSI アーキテクチャ

RCSI アーキテクチャとは、あるドメイン内のクラスを

(0) 認識 (*R*ecognition)

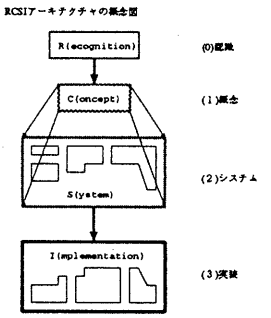
(1) 概念 (*C*oncept)

(2) システム (*S*ystem)

(3) 実装 (*I*mplementation)

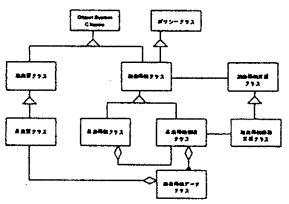
という 4 つの階層的な枠組みで位置づけることにより、その クラスのそのクラスライブラリ全体における抽象的な役割を明確化しようとするものである。

図 R-C-S-I 階層図を示す。



R-C-S-I 階層図

例 Borland C++ の Basic classes に適用した例を示す。



カテゴリ相関図

例えば、コレクション系のクラス群を考えた場合、容れ物としての Collection というものがまず設定され、その基本的な機能／プロトコルを定義する。これは、「容れ物」を認識したレベルと考えられる。

次に、容れ物としての標準的な最低限のプロトコルを保持したうえで、さらに各個別の状況に応じて機能や振る舞いのパターンを適切に提供できる「概念」クラスを定義している。これはいわゆる抽象データ型の概念に対応すると考えればよい。たとえば、BTree とか HashTable、Stack といった適切なメッセージプロトコルの集合として規定できる対象がこのレベルである。

次に、抽象データ型としての各概念を抽象的なシステムとして論理的に構成するアーキテクチャを示す論理実現レベルが存在する。これを「システム」と呼んでいる。例えば、BTree であれば、Node, InnerNode, LeafNode といった BTree を構成する上で論理的に必要となる補助概念クラス群の構成のされ方＝システムのことである。

最後に、各システムを実際に物理的に実装するのに必要な物理実現レベルのクラス群が登場する。この Borland の例では、対応物が存在しないが、例えば、各ノードをメモリ上に展開するのかディスク上に展開するのかを表現するメモリ管理クラス等がそれに当たるだろう。

解説 RCSI アーキテクチャの各レイヤはそれぞれひとつ下のレイヤによって支えられている。認識は概念によってサポートされ、概念はシステムつまり抽象的な機構や構造によってサポートされ、さらにシステムや構造をマシンの上で実現するのを実装のレイヤがサポートしている。

これはつまり、概念抽象の梯を一段づつ昇るごとにひとつ下のレイヤが「リソース」となってくれることを表している。逆に言うと、ある概念枠組み（フレームワークやコンセプトやポリシー）を一つ下のレイヤでより詳細に規定してやると言うことである。

この RCSI アーキテクチャは、抽象クラス、具象クラスの概念を発展させたものであり、すぐれたオブジェクト指向プログラマがクラスを構築する際に暗黙にしたがっていたガイドラインをより明確にしたものと考えられ、経験的な裏付けがあると言ってよい。

2.4 デザインパターン

デザインパターンとは、設計構造の抽象的な記述であり、しかもできるだけ

○ モジュール性

○ 柔軟性

○ 拡張性

○ 再利用性

を向上させるように選び取られたソフトウェア構成上のパターンを形成している必要がある。

デザインパターンの構成 各パターンは一貫した構造をもつ必要がある。各パターンは以下のような構成を取る。

(a) **パターン名**
その問題や解法を想起させる名前

(b) **事前要求パターン**
そのパターンが必要となる前に考慮しなければならない他のパターンの要約

(c) **問題状況**
そのパターンによって解かれるべき問題の核心を記述した簡潔なパラグラフ

(d) **図解**
問題を説明する図解

(e) **制約**
その解法に関する制約を考察した短い文章

(f) **解決手順**
その問題をどのように解決するかを記述した数パラグラフ

(g) **解説**
解法の説明

(h) **事後要求パターン**
そのパターンが満たされた後に考慮することになる他のパターンの要約

なかでも一番重要なのは図解であり、そのパターンの本質は簡潔な図解によって提示される。デザインパターンは実装言語に依存しない形で表現記述されている必要がある。

デザインパターンの例 デザインパターンの要件の典型例を示す。

(a) パターン名
「サブシステム」

(b) 事前要求パターン
「クラス」、「ユーザー要求」、その「ユーザー要求」状況に関与する「クラス」群の抽出

(c) 問題状況
あるまとまった機能／サービスをいかに複数のオブジェクトから構成するか

(d) 図解

(e) 制約
サブシステムに参加する各オブジェクトはそのサブシステムの提供するサービスをインプリメントしている必要がある。すなわちサブシステムのプロトコルに対して参加オブジェクトが実装メソッドを提供できることが前提となる。

(f) 解決手順
対象システムをおおざっぱな機能的側面でいくつかのサブシステムに分解する。その際、各サブシステムも1個のオブジェクトとして扱いカプセル化して独立性を確保し、他のオブジェクトやサブシステムとは狭いインターフェイス（プロトコル）を通してメッセージでやりとりするようにせよ。

(g) 解説
各サブシステムは利用者側（クライアント）から見ると一群のサービスの集合を提供するサーバとして機能する。

(h) 事後要求パターン
大きなサブシステムの場合には、そのサブシステム内での各オブジェクト間の協調関係を示す「チーム」パターンを参照せよ。また、サブシステムが「ユーザ要求」パターンを満たすことの確認も必要である。もしそのサブシステムの実装を考慮する場合には、「抽象実装」パターンを参照せよ。

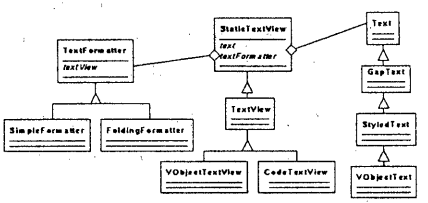
解説 上述したパターンの構成要素は、すべてが必ず現われるわけではなく、不要なもの、そのパターンの適用状況に不適当なものは省略されうる。

パターンの構成要素のなかで最も重要なものは、図解である。これによりそのパターンをまさにその構成オブジェクト間の「模様」＝「パター

ン」として2次元的に提示されたものである。したがって、パターンの外形やシルエットがそのパターンの環境とのインターフェイスを示唆しており、そのパターンの内部の模様がその問題状況を解決するための構造や機構を示唆していると考えることができる。

クラス間の汎化関係もひとつのパターンと考えることができるが、クラス間の横の繋がりや、インスタンス間の集団的な関係をクラス間の関係としてテンプレート化したものもパターンになりうるし、こちらの例のほうがクラスライブラリの構成をパターンとして表現する際にはより重要といえる。クラス間の汎化関係はRCSIアーキテクチャでよりマクロに表現できる手段があるからである。複数のオブジェクトがお互いにどう協調してある構造や機能や役割を提供しているか、ということがある程度静的に記述できるための枠組みがパターンなのである。

ここでは「サブシステム」以外の他のパターンの例は省略するが、他に「組立て」パターン、「役割協調」パターンなどがある。



「役割協調」パターンあるいは「MVC」パターン

2.5 抽象実装 / 実装パターン

前節ではデザインパターンについて述べたが、それはあくまでも特定の言語に依存しない、設計上の概念を自然にかつ適切に表現するための枠組みとして提示したものであった。そしてそこで選択されたデザインパターンは、実装時にある特定の言語をもちいてプログラムコードとして実現されることになる。

言語をオブジェクト指向プログラミング言語に限れば、設計とのインターフェイスを仮想化し、設計レベルにより近い形で抽象的な実装を行なうことが可能となる。つまり、オブジェクト指向言語のもつべき標準的な機能というものを特定し、個別の言語でそれらの標準機能を実現してやればよいのである。ここでは、オブジェクト指向の一般的な設計が可能となるレベルのオブジェクト指向標準言語機能でのシステムの実装を「抽象実装」と呼び、そのような抽象実装を可能にするための標準言語機能を個別のプログラミング言語を使って提供するための技法を「実装パターン」と呼ぶことにする。

これらの技術を用いた実装は、個々のアプリケーション固有の実装(たとえばXlibを用いてGUIを実装する場合)とは区別され、またコーディング規則の取決めというレベルのものでもない。Coplienはこのような技術を、高度な利用目的に対応した概念としてidiomsという表現を用いている[5]。抽象度の高い実装の基盤を提供する標準的なオブジェクト指向機能の仮想的な実現ノウハウということで、ここでは実装パターンと呼ぶことにする。

典型的な実装パターンは以下の4つである。

(a) 「正規直交クラス形式」実装パターン

組み込みの型とユーザー定義のクラスとを区別することなく完全に整合的に扱うという抽象実装のための技法

(b) 「ミニマルクラス定義」実装パターン

「正規直交クラス形式」実装パターンの応用パターン。これにより、クラスライブラリを構成する各クラスのプロトコルを統一でき、それらを組み合わせて使う際の基礎を提供する。

(c) 「ハンドル / ボディ」実装パターン

クラス定義を利用のインターフェイスを提供するハンドルクラスとそれを実装するボディクラスに切り分けることにより、ユーザーは内部の実装を意識することなく、抽象データ型としてクラスを利用できるようになる。

(d) 「メモリ管理」実装パターン

「ハンドル/ボディ」実装パターンの応用として、レファレンスカウント方式によるメモリ管理機能を提供するボディクラスを定義することにより、ポインター操作の誤りからユーザーを解放することができる。

(e) 「エンベロープ / レター」実装パターン

「ハンドル/ボディ」実装パターンの特殊化された形で、ポインターをオブジェクトそのものとして操作することを可能にする実装の枠組み。これにより、ポインターの取り扱いが容易になり、C++のCとして

の悪い面を避けることができる。また、継承階層にもとづくポリモアフィズムの強化にも役立つ。

3 まとめ

現在この業界でもっとも強く望まれているであろう、様々なプラットフォーム上で共通に使うことのできる、GUIを含む一般的なクラスライブラリ／アプリケーションフレームワークをできるだけ速やかにしかも低コストで普及させる努力をすることである。そのためには、できるだけ普遍的な概念で構成されたライブラリであることが必要不可欠であり、実用に使い込んでいけるだけの信頼性と頑健性、そしてさらに普及のことを考えた適用情報の提供がなされていることが望ましい。ここに述べた枠組に準拠した情報が提供されるなら、それら一連の情報を網羅的に把握する際の大きな助けとなる。

そうしたクラスライブラリの普及啓蒙活動を通して、ソフトウェア開発のインフラストラクチャとしてのクラスライブラリの流通の必要性、そしてクラスライブラリとして世の中に流通するために最低限クリアしている必要のある、ライブラリ構成およびドキュメンテーション上の条件を理解してもらう努力を続ける必要がある。

尚、本論文に述べた内容は、(株) シグマシステム「92年度シグマ会開発技術高度化委員会オブジェクト指向開発基盤検討部会」における活動 [1] を基にしている。また現在、提案枠組を支援する CASE 環境の試作と、その試作において構築されるクラスライブラリを提案枠組に則って評価する作業を継続して行なっている。

参考文献

[1] オブジェクト指向開発基盤検討部会報告書, 1993、(株) シグマシステム

[2] Grady Booch, Michael Vilotr, *The Design of the C++ Booch Components*, ECOOP/bs OOPSLA'90 Proceedings, 1990

[3] Peter Coad, *Object-Oriented Patterns*, CACM, September 1992, Vol. 35, No. 9

[4] William R. Cook, *Interfaces and Specifications for the Smalltalk-80 Collection Classes*, OOPSLA ' 92 Proceedings, 1992

[5] James Coplien, *Advanced C++, Programming Styles and Idioms*, Addison-Wesley, 1992

[6] Cox, B.J., *Object Oriented Programming - An Evolutionary Approach*, Addison-Wesley, Reading, MA, 1986

[7] Thomas Eggenschwiler, Erich Gamma, *ET++ SwapsManager: Using Object technology in the Financial Engineering Domain*, In Proceedings of OOPSLA'92, pp. 166-177, 1992, ACM

[8] Erich Gamma, Andre Weinand, *ECOOP'91 Tutorial: Object-Oriented Design in ET++*, 1991

[9] B. Meyer, *Object-Oriented Software Construction*, 1990, Prentice Hall

[10] B. Meyer, *Lessons from the design of Eiffel libraries*, CACM, September 1991, Vol. 33, No. 9