ソフトウェア工学 107－18
（1996．1．19）

ネットワーク管理ドメインにおける アプリケーションフレームワークの構築事例

荒野高志　青野博　藤崎智宏　中西弘毅
arano@slab.ntt.jp
日本電信電話株式会社
ソフトウェア研究所
〒180 東京都武蔵野市緑町 3-9-11

従来、GUI、OSなどの分野で、アプリケーションフレームワークが構築され、その有効性が評価されてきた。しかし、フレームワークの適用可能な範囲についてはよくわかっておらず、さまざまなアプリケーション分野でフレームワークの構築を試みることが重要となっている。本論文では、ネットワーク管理の分野において、フレームワークの構築に成功したので、報告する。まず、構築したフレームワークの構造やそのアプリケーション開発例を説明し、その再利用効果を定量的に提示する。また、プロジェクト管理面などを中心に、構築にあたって得られたノウハウを交えて、ドメインに独立した考察を行なう。

A Case Study on Application Framework Development in the Network Management Domain

Takashi Arano, Hiroshi Aono, Tomohiro Fujisaki and Kouki Nakanisi
Software Laboratories, Nippon Telegraph and Telephone , Corp.
3-9-11 Midori-cho, Musashino-shi, Tokyo, 180, Japan

Application frameworks for some domains such as GUI and OS have been developed and demonstrated their reusability and effective development. However, it is not clear yet what domains the framework concept can be well applicable for. It is important to try to develop and evaluate various domains of frameworks. This paper reports a network management system framework the authors developed successfully. The paper illustrates the architecture and how to reuse it with some metrics, and discusses project managment issues and framework applicablitity with several know-hows the authors have got in the development. Discussions are fairly domain-independent.

1．はじめに

アプリケーションフレームワーク（以降単にフレームワークと呼ぶ）[1] [3] がオブジェクト指向ソフトウェアの再利用性の鍵と言われてから久しい。文献 [1] ではフレームワークを「特定ドメインのアプリケーション、あるいはアプリケーションサブシステムのスケルトンインプリメンテーションであり、抽象クラスと具象クラスからなり、オブジェクトの協調モデルを提供するもの」と定義している。フレームワークはドメイン固有なクラスライブラリであり、ライブラリの中でいくつかのクラスが互いに関係しあっている。フレームワークを再利用する場合には個々の単一クラスではなく、ライブラリの全体を再利用し、アプリケーションの要求に従って、必要なクラスだけをカスタマイズする。そのままサブクラス化せずに使えるクラスについてはそのまま再利用する。

フレームワークは単にインプリメンテーションの再利用だけでなく、設計の再利用も行なうことを目的としている。すなわち、オブジェクト指向設計の重要な部分は、クラス間の関係を設計する部分であるが、フレームワークはクラス間の関係を内包しており、フレームワークの再利用は、即クラス関係の再利用になる。最近、設計を再利用する手法としてデザインパターンが提案され、広く研究されている [14]。デザインパターンはクラスの関係をパターンとしてとらえ、設計に用いようとするものである。フレームワークは、ドメインに固有なものである点と、実際にコードを含む点がデザインパターンと異なる。フレームワークは適用範囲は狭いが、効果は大きい。

フレームワークの成功したドメインとして、グラフィックユーザインタフェース [2] [3]、OS [3]、財務処理 [4] などの事例が知られており、設計・実装の再利用のおかげで開発工数が大幅に短縮されることが報告されている。今後徐々に実開発にも広くフレームワークが適用されていくと考える。

しかし、フレームワークの適用可能な範囲についてはよくわかっていない。さまざまなアプリケーション分野でフレームワークの構築を試み、そこで得られたノウハウを一般的に議論することが重要となっている。

われわれはネットワーク管理の分野において、既にトイプログラムに対し実験を行ない、アプリケーションのクラスの９０％以上のクラスが再利用であったという結果をえている [11] [12]。今回は実用的なフレームワークの構築に成功したので、報告する。本フレームワークを元に作られたネットワーク管理システムは実際の商用インタネットなどの運用に用いられている。

本報告では、まず構築したフレームワークの構造やそのアプリケーション開発例を説明し、その再利用効果を定量的に提示する。また、開発プロセス、教育などプロジェクト管理面などを中心に、構築にあたって得られたノウハウを交えて、ドメインに独立した考察を行なう。

2．ネットワーク管理フレームワークについて

2．1　ネットワーク管理とは

ネットワーク管理システム [15] とは、ネットワークを運用するために、ネットワークやその上の機器を管理するシステムである。機能としては、ネットワークの状態監視や故障時の対応などを含む故障管理、ネットワークトラフィックなどの性能管理、ネットワークの構成に関する管理、ユーザアカウントや課金などの管理、ネットワークのセキュリティに関する管理などがある。管理対象機器には通常、遠隔から管理可能なエージェントが動作している。エージェントはSNMPなどの特定のプロトコルによって、機器の状態の問い合わせや設定に応じることができる。ネットワーク管理システムはこれらのエージェントと通信する形で構成される（図１）。

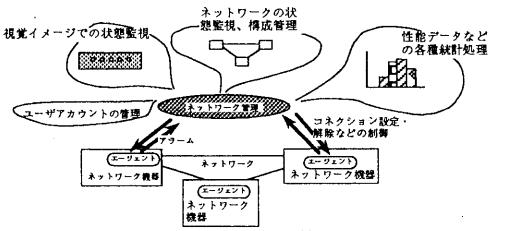

図１　ネットワーク管理とは

2．2　フレームワークの構成

図２にわれわれが開発しているネットワーク管理のフレームワーク (HatTrick) [7] のアーキテクチャを示す。管理対象となっている各種機器を管理するために、管理対象と１対１対応するような（分散）オブジェクトをおく。このオブジェクトを総称して

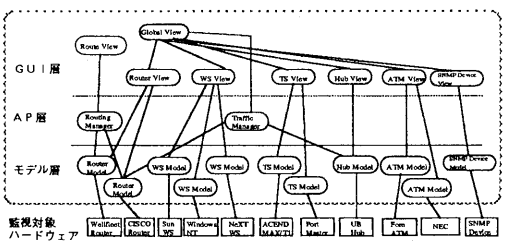

図２　HatTrickアーキテクチャ

モデルと呼び、アーキテクチャ上ではモデル層と呼ぶ。モデルを操作して、ネットワーク管理業務を行なうような層がアプリケーション層であり、ネットワーク管理のＧＵＩを形成するのがビュー層である。また、この３層以外に、分散オブジェクト群の動作をサポートするオブジェクト群 [5] がある。

このアーキテクチャは、いわゆる３層モデル [9] の一つの応用例としてもとらえられる。ただし、ビュー層のオブジェクトが直接モデル層のオブジェクトを通じて、機器の状態を直接見たり設定したりするショートカットの場合も多い。また、GUI 層・ＡＰ層のオブジェクトをあまり区別せず実現することもある。例えば、GlobalView アプリケーションはネットワーク全体を監視・制御するアプリケーションであるが、AP 層としてのネットワークの構成管理とGUI層としての視覚的な画面インタフェースを実現する。

モデルはアプリケーション側から見れば、ある種の対象機器のプロキシであり、アプリケーションに対し、管理対象固有のプロトコルを隠蔽しつつ、機器への問い合わせや設定を行なうことができる。また、機器からあがってきたアラームを登録されたアプリケーションに転送する役割も果たす。TCP/IP系のネットワーク管理では、SNMP というプロトコル以外にも、Telnet や RPC、BOOTP などさまざまなプロトコルを利用できるが、これらのプロトコルの差異はアプリケーションには見えない。

これらのオブジェクトは、クラス記述上ではそれぞれ継承階層をなしている。例として、図３にモデルの継承階層を示す。TCP/IP系の機器のモデルである NetworkDevice 抽象クラス、管理機器の種別対応の抽象クラス群、ベンダ別の具象クラスからなる。それぞれの抽象度のところで、適切なインタフェースと適切な実装が行なわれている。例えば、RouterModel 抽象クラスではルート表設定のためのインタフェースが規定されており、それらに必要な機器管理プロトコルは通常ベンダによって異なるため、それらはベンダ固有の具象クラスで実装される。

また、モデル内部の構造を図４に示す [16]。ルートとなるオブジェクトが、各機能をつかさどる機能マネージャ群及びプロトコル処理を行なうプロトコルコントローラをもつ。機能マネージャは機器種別によって何を持つかが異なり、例えば RouterModel であれば、ポートを管理する役割の PortManager、トラップ等の履歴を管理する LogManger などをもつ。また、WSModel においては、ワークステーション上のプロセスを管理する ProcessManager、ディスクを管理する StorageManager などがある。

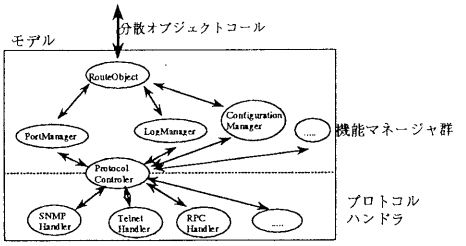

図４　モデル構造の例

プロトコルハンドラは個々のプロトコルを処理するものであり、特定の機種毎、特定のプロトコル毎に特定のハンドラ群を用意する。例えば、Cicso社のルータのモデルでは、ＣｉｃｓｏＳＮＭＰＨａｎｄｅｒ と CicsoTelnetHandler をもつ。プロトコルコントローラはこれら複数のプロトコルハンドラを管理し、機能マネージャ群の要求を受けたら、その要求を適切なプロトコルコントローラに自動振分けする [10]。自動振分け機構としては、プロトコルコントローラが各プロトコルハンドラに対し、その要求を受け付けるかを問い合わせて、OK を出したハンドラに処理を任せるという仕組みとなっている。

プロトコルハンドラ自身も継承構造をなしている（図５）。ProtocolHandler 抽象クラスは各プロトコル対応のサブクラス群を持ち、それらプロトコル対応のサブクラスは特定の機器／特定のプロトコル対応の具象サブクラスをもつ。例えば、SNMP の共通 MIB 仕様である MIBII 部分は、SNMPHandler クラスで実装され、各ベンダ固有の private MIB は各ベンダ専用の SNMPHandler クラスで実装される。

２．３　フレームワークの応用例と再利用効果

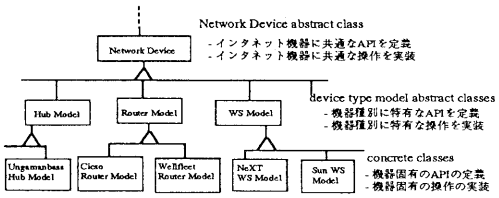

図３　モデルの継承階層

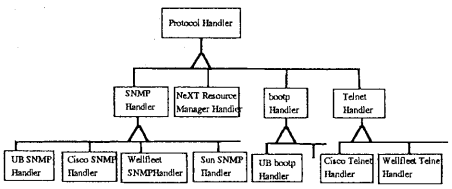

図５　プロトコルハンドラの継承構造

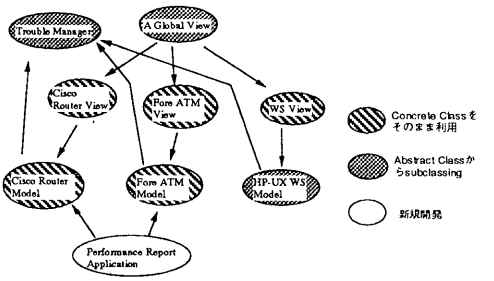

図6　ネットワーク管理システム構築イメージ

ネットワーク管理システムの構築イメージを、図6に示す。モジュールの開発法として、基本的に3種類がある。まずはモジュールをそのまま再利用できる場合である。図6では、WSViewアプリケーションがそれにあたる。WSViewアプリケーションは、ワークステーション上のプロセス管理、ユーザ管理などを行うアプリケーションであるが、使用しているモデルインタフェースはWSModel抽象クラスのものである。すなわち、実際に各機器対応にWSModelクラスの具象サブクラスを作成すれば、WSViewアプリケーションは修正なしでそのまま再利用できる。次は、GlobalViewアプリケーションを例にとる。このアプリケーションはネットワーク全体を監視し、制御するアプリケーションであり、図7は公衆インタネットの管理システムのGlobalViewアプリケーションを示している。実際の個々のGlobalViewアプリケーションでは、ネットワークの構成やそれを表示する画面構成は個々に異なるので、HatTrickフレームワークでは、その骨組みだけを抽象クラスとして用意している。これをサブクラス化してアプリケーションを構成する。3点目は全

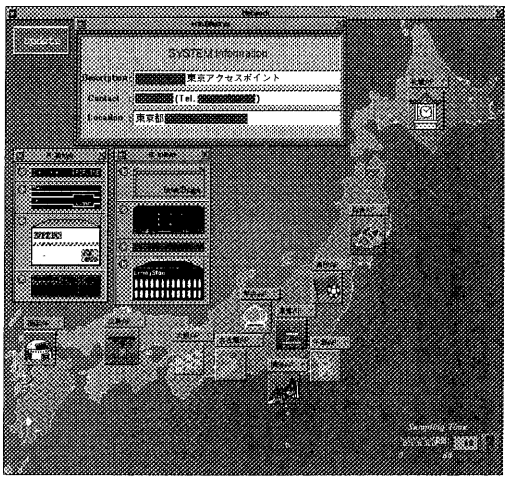

図7　公衆インタネット管理システム

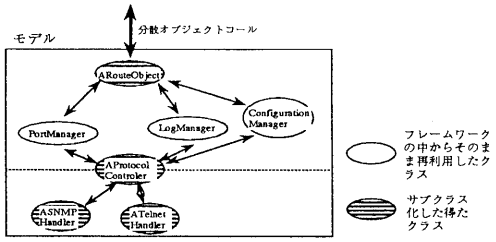

図8　モデル開発の例

くの新規開発になる。例えば、現フレームワークには性能管理アプリケーションがなかったとすれば、PerformanceManagerアプリケーションとして、必要なモデルオブジェクトをアクセスして性能管理を行う処理を新たに記述しなければならない。しかし、こうして新規に記述したアプリケーションも（その要求自体が一般的であれば）、後のフレームワーク成熟化フェーズで汎用的になるように修正してのち、フレームワークに組み入れられ、再利用可能な部品となっていく。

モデルオブジェクトの構築に当たっても、既存のものをそのまま再利用できる場合と、新しい機器に修正しなければならない場合とがある。後者の場合の構築例を図8にに示す。いくつかの既存サブクラス化を行うだけでよい。

表1にスーパクラスであるRouterModelクラスなどを再利用し、Ｃｉｃｓｏ社のルータモデルCicsoRouterModelクラスを構築したときの再利用率を示す。総クラス数は８８（30kline）であり、これにはシステム（HatTrickがベースにしているのはNeXTSTEP）のライブラリクラスは含まない。共通クラスは、例えば日付、時間やIPアドレスなどをクラス化したものやネットワーク管理用のグラフィックの部品クラスなどから構成される。上位クラスとは、例えばRouterModel抽象クラスなど、直接／間接にスーパクラスとして参照するクラスの数である。再利用率（再利用クラス数／総クラス数）は８６％になる。また、開発者の実感としては８６％以上の恩恵を受けた感じがしている。これは、新規作

表1　クラス再利用率

モジュール名	Cicso社ルータモデル	GlobalViewアプリケーション
総クラス数 (A)	88	73
共通クラス数 (B)	57	54
上位クラス数 (C)	19	14
新規作成クラス数	12	5
再利用率 $\frac{(B)+(C)}{(A)}$	86%	93%

成クラスの内部の構成は極めて単純であり、設計上の考慮点などはほとんどなく、コーディングも簡単であるためである。そのため、工数は、機器仕様の調査の時間を除いて、多くの例において実績的には１人週程度ですむ。

表２　フレームワークを用いた開発の特徴

通常のオブジェクト指向開発	フレームワークを用いた開発
・新規設計・実装量多い ・多人数を前提 ・ドキュメント指向 ・大人数のプロジェクト管理 ・ラウンドタイムの長いプロトタイピング	・再利用率が高く、新規設計・実装量少ない ・少人数で開発可能 ・ドキュメント後付け ・少人数プロジェクト管理 ・短いラウンドタイムのプロトタイピング ・フレームワークの理解が必要 ・フレームワークへのフィードバックプロセスが必要

アプリケーション層／ビュー層のオブジェクトにもほぼ同様のことが言える。表１にGlobalViewアプリケーションのデータを示す。再利用率そのものは同程度以上の高い値を示しているが、新規サブクラスの複雑さはモデルの例より高い。工数見積りはまだ実績があまり積まれていないので、あまり確固たることは言えないが、今までの少数の例では２～４人週程度である。

このようにして、システム全体はこれらのモジュールの積み上げで完成され、特に新規の機能が多くなければ、２～４人月程度で開発できる。例えば、某社公衆インタネットサービスのネットワーク管理システム（図７）の開発に２名×２カ月、１０種類程度の異なる仕様（プロトコル）をもったワークステーションを含むCALS用ネットワーク管理システムに同様に２名×２カ月などの実績をもつ。このほかの事例もほぼ同様の工数となっている。前者の場合、従来型の開発を行った場合（ある程度のツールなどは使う前提で）の工数の見積りが２２人月であったので、これに比べて５．５倍の開発効率であると言える。

３．考察

３．１　開発プロセス

フレームワークを再利用するという場合、フレームワークを単に再利用するだけの立場と、そのフレームワークを成熟化させていくような開発と２つの立場が考えられる。一般に、GUIのフレームワークを入手して再利用する場合は前者になるであろうし、企業内等でソフト資産を再利用可能な形にしていこうという場合は後者になるであろう。われわれのネットワーク管理システムは後者に属するので、本節では後者の立場でフレームワークの開発プロセスについて考察を行いたい。

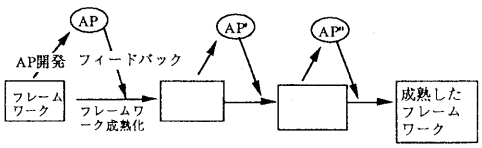

図９　フレームワークの成熟化過程

後者の場合のトップレベルの開発プロセスは図９のようになる。アプリケーション開発とそこで開発されたもののフレームワークへの組み込みなどによるフィードバックの２つの主要なプロセスからなる。われわれの経験では、この２つの区別は厳密である。特定のアプリケーションやシステムを開発する時に、そのまま再利用可能となる部品を作っていくのは非常に難しいから、その再利用可能な形に整形するフェーズが必要となるためである。以下で、その各々についてさらに議論する。

３．１．１　フレームワークを用いた開発プロセス

フレームワークを用いた場合の開発プロセスと通常のオブジェクト指向開発のプロセスを比較したのが表２である。開発工程そのものはさほど違いはない。どちらもインクリメンタルでイテラティブな開発プロセスをとる。大きな違いは、分析・設計・コーディングなどの各工程の割合、ドキュメンテーションの重要性、プロトタイピングサイクルの長さである。これらは主に再利用性の違いからくる。

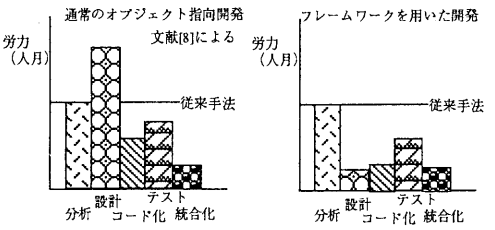

図１０　労力配分の違い

図１０に、各工程の労力配分を比較したものを示す。通常のオブジェクト指向開発においては、設計にかける時間は従来手法とくらべて、かなり長いと言われている [8]。一方、フレームワークを用いた場合は、設計時間が短い。これはフレームワークの再利用は、設計の再利用となっているからである。

同じ機能をもったシステムの開発という意味では、フレームワークを用いた開発は工期も短く、また一般に開発要員も少なくて済む。設計としての決定事項も少ないため、ドキュメンテーションの重要

性はさしてない。むしろ、少しでも早期に開発して、ユーザ要求の獲得のためのフィードバックサイクルを回す方が重要である。

図7で紹介した公衆インタネット管理での開発では、開発8週のうち、画面や機能などの仕様変更は都合3回行なっている。当初の要求仕様分析に1週間、機能設計は1日程度、詳細設計はコーディングと同等のレベルで行ない、2週程度（簡単なデバッグ含む）、さらにプロトタイプの評価と要求仕様調整を行なう。以降、同様のサイクルを何度か実行し、最後に総合試験を行なう。この値から、特に機能設計レベルが再利用により大幅に削減できたことがわかる。また、仕様のフィードバックをかけることにより、システムの有用性や使い勝手が向上した。

3．1．2　フレームワークの成熟化プロセス

成熟化プロセスでは、アプリケーションで開発したクラス（群）をフレームワークに組み込むために整形したり、それらに伴いフレームワークの中のクラスの構成やインタフェースを変えたりする。非常に多くのサブクラスをもつクラスがある場合に、その中間にクラスを設けてインタフェースや実装の共有をはかるなどの例がある。工数はかかるが、機能拡張は基本的には行なわない。

フレームワーク構築のためには、この成熟化プロセスは本質的であり、成熟化プロセスなくしてはフレームワークは構築しえない。それは当初から将来にわたっておきるすべての要求を見込んで、かつそれに最適な構成をもったフレームワークを構築するのは、ほぼ不可能であるからである。われわれのプロジェクトにおいても、この成熟化プロセスを何度か実行しているが、これが現在の高い再利用率に貢献している。開発マネージャとしては、いつどのようにこの成熟化プロセスを管理していくかというのがキーとなるわけであり、また、シニアマネージャも成熟化プロセスの重要性を十分に認識しておく必要がある。

ここでは、アプリケーション開発の場合とは異なり、設計フェーズがもっとも重要となる。どのようなクラス間の関係やクラスインタフェースが再利用性を向上させるかという観点で議論を行う必要がある。

成熟化プロセスでは大きな修正工数がかかることがあるが、われわれは、このような場合には、メトリクスを用いたリスクドリブンなプロトタイピング手法を用いることとしている [13]。すなわち設計のディシジョンを検証するために、一部をプロトタイピングし、メトリクス計測によって、その設計のディシジョンを評価する手法である。例えば、クラスの抽象化を行う際に、一部のクラスについてのみ実装などを行い、その再利用性、保守労力、クラスの質などを計測し、変更前後の差分を評価することを行い、リスクの回避をおこなうことができる。

3．2　教育

フレームワークの構築及びフレームワークを用いた開発において、もっとも重要のことの1つは、要員である。フレームワークを理解しなければ、再利用することもできなければ、また理解せずに使おうとしても手戻りを生じることも多い [12]。われわれのプロジェクトでは、フレームワーク（及びシステムのクラス群）を勉強して、要員が立ち上がるまでの期間をおよそ3カ月程度みている。これにオブジェクト指向、及びGUI Builderなどの開発ツールの勉強も必要な場合はさらに3カ月程度必要である。ただし、習熟期間は人によって異なる。一般には手続き型言語になれた人はオブジェクト指向への転換は難しいといわれているが [6]、われわれのプロジェクトでは優秀なCプログラマが優秀なオブジェクト指向設計プログラマに転換した例もいくつかある。

われわれのプロジェクトでは、要員教育のためのプログラムを用意している。これは、演習、参考図書、OJTからなる。演習では、典型的なトイプログラムの課題をいくつかこなすこととなる。各々の課題は修得すべき項目を明確に意識したものとなっている。参考図書は分析設計法及びプログラム言語などの市販本である。これは主に自習用である。OJTでは実際のネットワーク管理フレームワークの一部を開発して、徐々にフレームワークの中身を勉強していく。OJTでは、まずアプリケーションよりの部分、特にビジュアルな部分からはじめるようにしている。これらの部分はオブジェクトが明確であり、またフレームワークの中のクラスの相関をさほど知る必要はないからである。インセンティブを与えるという意味からは、実際のプロジェクトの一部をきりだすようにしている。アプリケーションの部分の修得が終わると、OJTとしては、モデルオブジェクトを開発する課題を行う。モデルは2章で説明したように、このネットワーク管理フレームワークの中核をなす部分であり、新しいモデル開発のためには、既成のクラス構成を十分理解する必要がある。この段階が終われば、一通りの教育は終了する。

また、このような新しい要員のための教育プログラム以外に、教育的に効果が高いのが、設計レビュー、及びコーディングレビューである。ひとつ

の意味は上級者の設計・コーディングをみて、あるいは上級者間の設計についての議論を聞いて、中級者以下のメンバは再利用の重要性や再利用に向いた設計のパターンについて自然と学ぶことができる。また、それ以上に重要なのは、初中級者にとって、自分たちの設計・コーディングについて、議論が行われることである。自分が何らか考えてえた結論について、議論が行われるのは、他人の問題の場合と違って、考え方がよく身に付くようである。もちろん、単に上級者が設計・コーディングの欠点を指摘するのではなく、レビューの中で欠点の理由やどういう場合にそうなるか、あるいは改善法について、示唆する必要がある。また、レビューに参加するメンバの意識として、再利用を最大限にしようという考え方を共有させることも、レビューの管理者の重要な責務である。

上級者の関与の割合も重要であると感じている。すなわち、新たな要員を1人増やしたときと、数人増やしたときでは、どうも前者の方が立ち上がり方が早いようである。この点はサンプルが少ないので断定的には結論づけられないが、上級者が頻繁に初級者をみていて、その都度、不適切な考え方をしないように随時指導するのは効果が高いと考える。

3．3　体制

われわれのプロジェクトにおいては、モジュールごとの縦割りの分担を行う体制はとっていない。ひとつには各専門家による分業開発である。フレームワークに精通し、オブジェクト指向的にも技術が高い開発者は、基幹の部分のモデルなどを専門に開発する。また、GUIのグラフィックデザインはその得意な技術者がすべてのシステムにわたって、横通しでその部分を開発する。もうひとつは、OJTとの関係である。上級者が難しい部分、初級者は平易な部分という縦割り分担を行う場合もあるが、場合によっては、上級者がクラスインタフェースとメソッドの概略だけ書き、すなわち詳細設計を行い、初級者が残りのコーディングとデバッグを行うという分担を行うこともある。

また、体制の中にフレームワークライブラリアンもおく試みをはじめた。フレームワークが徐々に大きくなってきて、全体の整合性やチェックアウトした部品のフィードバックが十分に行われず、クラスに複数のバージョンなどが遍在するような事態になったためである。ライブラリアンはライブラリの管理やアプリケーションからの部品のフィードバックのための修正などを行う予定であるが、アプリケーション開発者との責務分担の詳細は今後の課題である。

3．4　フレームワークの適用範囲

フレームワークについてはGUIなどの分野で成功をおさめているが、どの分野であれば、フレームワークが作りうるかについては、現状のところよくわかっていない。本節では、適用範囲の条件について考察する。

まず第1の条件は対象となるものが明確であること、第2の条件は対象の捉え方がほぼ一意であり、共通認識があることである。ネットワーク管理の場合は対象は管理機器という具象物であり、またManaged Objectなどの世界標準 [15] も存在しており、この条件を満たす。また、GUIについてもウィンドウ、メニューなどは目に見えるものであり、その機能も明確に定められている。

第3の条件は対象モデルが安定していることであり、第4は似たようなシステムが多いこと、すなわちフレームワーク使用頻度が多いことである。3．1節で述べたように、フレームワーク構築のためにはその成熟化プロセスが必須となる。第3／4の条件は成熟化プロセスにかかるコストが後の再利用によって適正にペイするかという条件である。これが満たされない限りはフレームワークとしては存在しえない。

対象モデルが安定するとは、時間的な推移によって対象を抽象化したモデルがあまり変更されないということである。ネットワーク管理の場合、管理機器は非常に頻繁にアップグレードされたり、新機種が登場したりすることも多いが、管理機器としての抽象モデルはさほど変更はない。単に機器の処理速度が上がる、扱えるポートの数が増えるなどのマイナチェンジにすぎない。また、GUIの抽象モデルもここ何年もさして変化がない。

頻度の条件は重要である。成熟化にかかるコストは、利用回数が多ければ多いほど、1回分のコストが低くなるからである。端的には、後に再利用する見込みのないクラスの整理を行っても、それは全く徒労である。ネットワーク管理の場合には、適用箇所は非常に多い。世の中に新しい（少しずつ構成の異なる）ネットワークは次々と作られ、ネットワーク管理システムの開発要求は後をたたない。また、GUIについても、おおよそすべてのシステムはGUIがつくのが通例であり、適用頻度は非常に多い。

一方、ある種の分野、例えば給与計算システムは、対象となるものが人為的な規則であったりする。一般に人為物のモデル化は難しい。まず、汎用性がなく、組織や文化によってモデルが異なることが多

い。また、人為物はなんらかの要因で大きく構造が変更されることがある。例えば、給与計算システムのモデルは社則やあるいは税制の変更などに大きく影響される。また、ある種のソフト、例えばミサイル制御や交換機のソフトなどは、何年かに1度作成されるだけであり、頻繁に再利用されることは少ない。このようなシステムでは、一般に成熟化に投資したコストが回収できる見込みが薄く、フレームワークが成立しにくい。

4．まとめ

本論文ではネットワーク管理分野におけるフレームワークの例を示した。フレームワーク内のクラス構成について説明し、再利用性を計測した。再利用率は８０％を越えるものであり、設計・コーディングの期間が大幅に短縮できる。また、開発の経験からえたいくつかの知見として、開発プロセス、教育、フレームワークの有効な分野について考察を行った。

開発プロセスは、フレームワークを再利用する開発、フレームワークを成熟化させるプロセスに分けられることを述べた。特に後者の成熟化プロセスはフレームワーク構築に本質的である。実際、われわれのプロジェクトでも何度かの成熟化プロセスをへて、現在の再利用率にいたっている。また、フレームワークを再利用する開発においては設計時間が非常に短縮できる点がBoochの考えるオブジェクト指向の開発時間と大きく異なることを示した。一方、フレームワーク成熟化させるプロセスにおいては、設計の重要性やよい設計をメトリクス評価しプロトタイピングする手法についてふれた。

要員のスキルと教育は非常に重要なポイントである。われわれは３－６カ月程度の教育プログラムを開発し、適用している。また、レビューや上級者の関与が初中級者のスキル向上に非常に重要であることを明らかにした。

フレームワークの適用範囲については、満たすべき４つの条件を考察した。ドメイン対象の明確性、共通認識性、安定性、利用頻度である。これらのどれが欠けてもフレームワークは構築できない。

フレームワークという概念が提案されてから、かなりの年月がたつが、最近ようやくGUI以外のさまざまな分野のフレームワークの成功例が報告されるようになってきた。しかし、どの分野なら、どの範囲ならうまくいくという考察はまだ十分になされていない。これに対し、ひとつひとつ成功事例を重ねていくとともに、一般的な知見として分野を越えた議論を行っていく必要があるであろう。

参考文献

[1] Wirfs-Brock, R.J., and Johnson, R. : Surveying Current Research in Object-Oriented Design, Communication of ACM, Vol.33, No.9, pp.104-124, 1990

[2] Schmucker, K.J. : "MacApp : an application framework," Byte, pp. 189-193, August, 1986.

[3] Lewis, T., et.al., Object Oriented Application Framework, Manning Publications Co., 1995

[4] Profrock, A.K., Tsichritzis, D., Muller, G. and Ader M. : "ITHACA : An Integrated Toolkit for Highly Advanced Computer Applications, Object-Oriented Development," Tsichritzis, D. Ed. Universite de Geneve, 1989, pp. 321-344.

[5] 藤崎、薮山、荒野、ネットワーク管理システムにおける分散オブジェクト管理、第71回マルチメディア通信と分散処理研究会、情報処理学会, 1995

[6] Henderson-Sellers, B., and Edwards, J.M. : The Working Object, Prentice Hall, 1994

[7] Arano, T. et al. : A Computer Network Management System Platform based on Distributed Objects, 6th IFIP/IEEE international workshop on Distributed Systems : Operations & Management (DSOM'95) , 1995

[8] Booch, G., Object Oriented Design with Applications, The Benjamin/Cummings Publishing Company, Inc., 1991

[9] Donovan, J.J., Business Re-engineering with Informtion Technology, Prentice Hall, 1994

[0] 薮山、藤崎、荒野、ネットワーク管理におけるマルチプロトコルハンドリングについて、情報処理学会分散システム運用技術研究グループ資料、DSM-95-07040, 1995

[11] 荒野他：オブジェクト指向フレームワーク再利用性の一実験、情報処理学会第49回全国大会、1994

[12] 本位田他編：オブジェクト指向分析・設計 - 開発現場に見る実践の秘訣 -、共立出版、1995

[13] 荒野、中西、藤崎、フレームワーク成熟化段階におけるメトリクス計測事例とその考察、オブジェクト指向'95 シンポジウム、情報処理学会、pp. 261-pp. 268, 1995

[14] Gamma, E., Design Patterns, Addison Wesley, 1995

[15] Stallings, W., Network Management, IEEE Computer Society Press, 1993

[16] 藤崎、荒野、分散オブジェクト指向プラットフォームにおけるルータの運用・管理、情報処理学会分散システム運用技術研究グループ資料、DSM-95-01028, 1995