ネットワーク管理フレームワークとその開発に関する一考察

荒野高志[†] 青野博[††] 藤崎智宏[†††]

従来，GUI，CASEツール，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分かっておらず，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う．

A Network Management Framework and Its Development

Takashi Arano,[†] Hiroshi Aono[††] and Tomohiro Fujisaki[†††]

Application frameworks for some domains such as GUI, CASE tools 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 as much as we can.

1. はじめに

アプリケーションフレームワーク（あるいは単にフレームワーク）[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)]，製造管理[18)]，ネットワークプロトコルソフトウェア[17)]ほか，いくつかの事例[4),19)]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．

† 日本電信電話株式会社マルチメディア実験推進室
Multimedia Promotion Office, Nippon Telegraph and Telephone, Corp.
†† NTTソフトウェア株式会社通信応用システム事業部
NTT Software Corporation
††† 日本電信電話株式会社ソフトウェア研究所
Software Laboratories, Nippon Telegraph and Telephone, Corp.

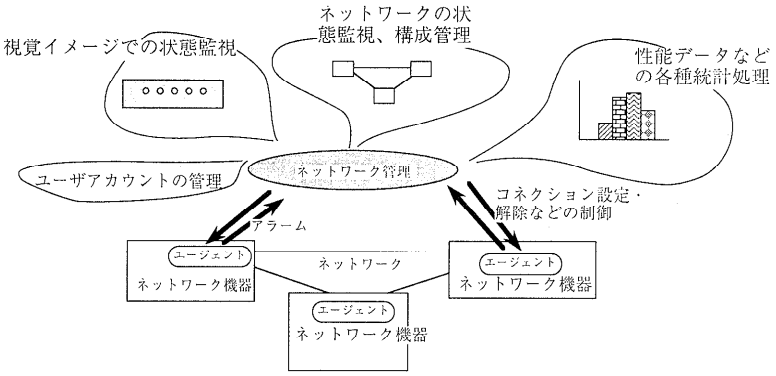


図1 ネットワーク管理とは
Fig. 1 Network management system.

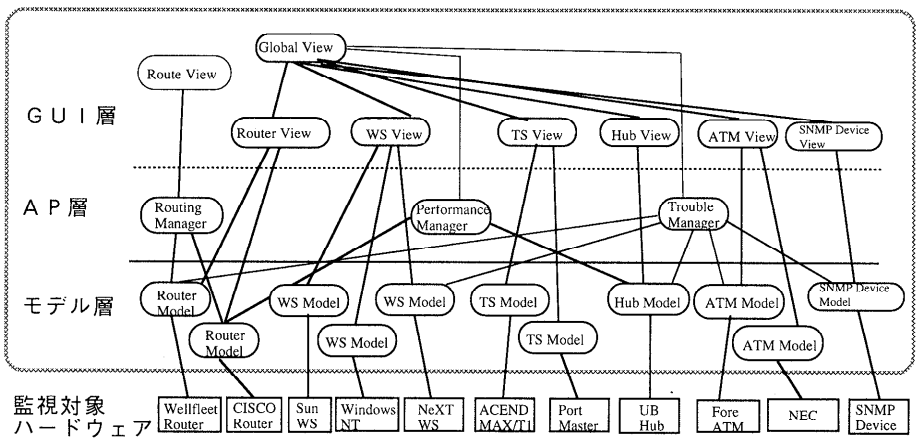


図2 Hat Trick アーキテクチャ
Fig. 2 Hat Trick architecture.

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分かっていない．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．

我々はネットワーク管理の分野において，すでに小規模な実験プログラムに対し，実験を行い，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得ている[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もと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の管理などに用いられている．

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 (Simple Network Management Protocol)[15)]などの特定のプロトコルによって，機器の状態の問合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）．

2.2 フレームワークの構成

図2に我々が開発しているネットワーク管理のフ

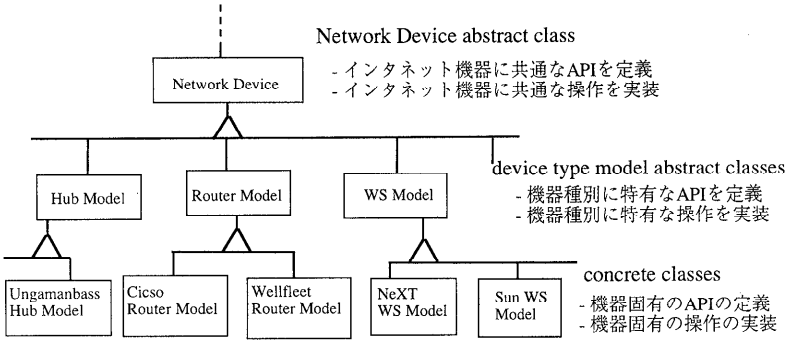


図3 モデルの継承階層
Fig. 3 Inheritance hierarchy of model classes.

レームワーク（Hat Trick）[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（Graphical User Interface）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がある．

このアーキテクチャは，いわゆる3層モデル[9]の1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・アプリケーション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．

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（代理人）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，機器への問合せや設定を行うことができる．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．

現在のフレームワークでは，アプリケーション層のオブジェクトは Trouble Manager などがあるだけで，まだ整備途中である．現在ルーティングの管理を行う Routing Manager や，ネットワーク全体の性能を管理する Performance Manager などを検討中である．

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 Network Device 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われている．たとえば，Router Model 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オブジェクトおよびプロトコル処理を行う Protocol Controller オブジェクトを持つ．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，たとえば Router Model であれば，ポートを管理する役割の Port Manager，トラップなどの履歴を管理する Log Manger などを持つ．また，WS Model 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 Process Manager，ディスクを管理する Storage Manager などがある．

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，特定の機種ごと，特定のプロトコルごとに特定のハンドラ群を用意する．たとえば，Cisco社のルータのモデルでは，Cisco SNMP Hander と Cisco Telnet Handler を持つ．Protocol Controller はこれら複数のプロトコルハンドラを管理し，機能マネージャの要求を受けたら，その要求を適切なプロトコルハンドラに自動振分けする[10]．自動振分け機構としては，Protocol Controller 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**）．Protocol Handler 抽象クラスは各プロトコル

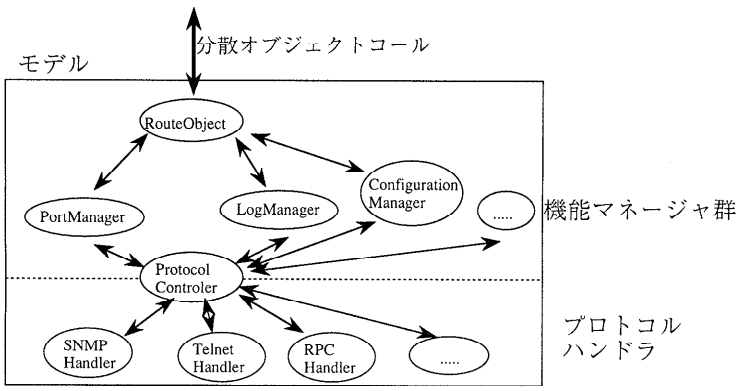


図4 モデル構造の例
Fig. 4 Inside the models.

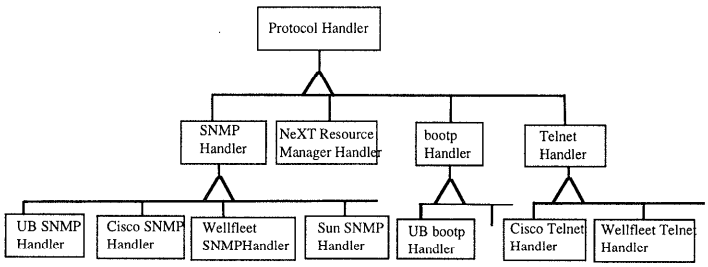


図5 プロトコルハンドラの継承構造
Fig. 5 Inheritance hierarchy of protocol handler classes.

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持つ．たとえば，SNMP の共通 MIB（Managed Information Base）仕様である MIB II 部分は，SNMP Handler クラスで実装され，各ベンダ固有の拡張 MIB は各ベンダ専用の SNMP Handler クラスで実装される．

このような構造をとる最も重要な理由は再利用性である．たとえば，新しいワークステーションを管理するために WS Model 抽象クラスのサブクラスを作成しなければならないとする．この際，必要なのは新しいワークステーションのための Protocol Handler クラスの作成だけで，WS Model 抽象クラスの持つ機能マネージャは，すべてそのまま再利用可能である．また，新しく必要となる Protocol Handler 自身もサブクラシングによって開発できる．

2.3 フレームワークの応用例と再利用効果

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，WS View アプリケーションがそれにあたる．WS View 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 WS Modcl 抽象クラスのものである．すなわち，実際に各機器対応に WS Model クラスの具象サブクラスを作成すれば，WS View 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，Global View 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る（**図7**）．実際の個々の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，Hat Trick 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．これをサブクラッシングしてアプリケーションを構成する．3点目はまったくの新規開発になる．たとえば，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，Performance Manager アプリケーションは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

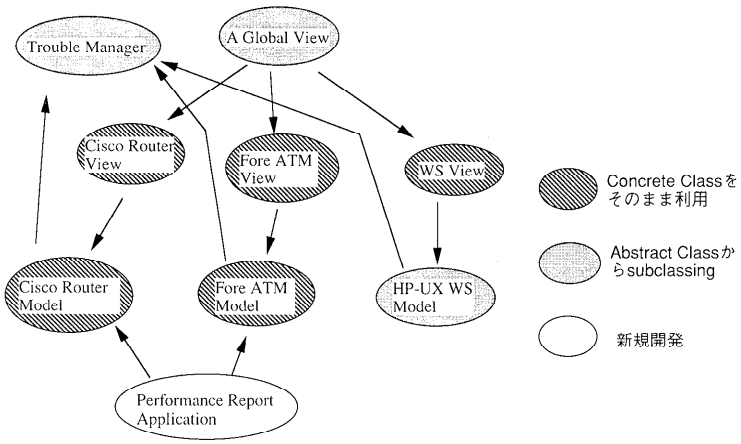


図6 ネットワーク管理システム構築イメージ
Fig. 6 Network management system modules and their development.

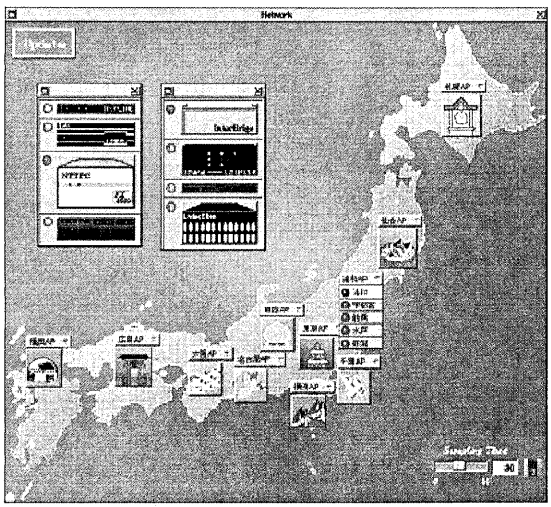


図7 公衆インタネット管理システム
Fig. 7 A network management system for a public internet provider.

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る．

モデルオブジェクトの構築にあたっても，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**に示す．基本的には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 Router Model クラスなどを再利用し，Cisco 社のルータモデル Cisco Router Model 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 である．これは Cisco Router Model クラスをコンパイルするのに必要なすべてのクラスの数であり，システム（NeXTSTEP OS）のライブラリクラスは含まない．共通クラスは，たとえば日付，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，たとえば Router Model 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，新規作成クラスの内部の構成はきわめて単純であり，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．

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いえる．表 1 に上で紹介したアプリケーション Global View のデータを示す．再利用率そのものは同じようなものであるが，新規サブクラスの複雑さはモデルの例より高い．工数見積りは実績がまだそれほど積まれていないので，あまり確固たることはいえないが，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．たとえば，某社インタネット公衆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）を持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 カ月などの実績を持つ．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

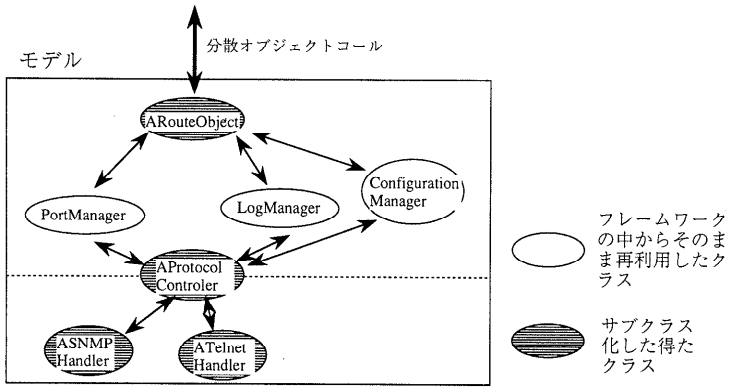


図8 モデル開発の例
Fig. 8 Model development.

表1 クラス再利用率
Table 1 Class reusablity.

モジュール名	Cisco 社ルータモ	Global View 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	12	5
再利用率 $\frac{(B)+(C)}{(A)}$	86%	93%

工数の見積りが 22 人月であり，これに比べて 5.5 倍の開発効率であるといえる．

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，企業内など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．我々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．

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．我々の経験では，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ときに，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．

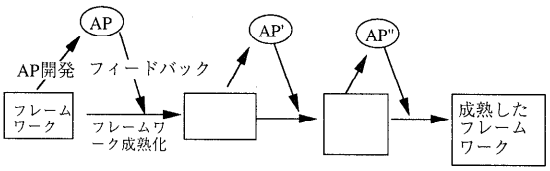


図9 フレームワークの成熟化過程
Fig. 9 Framework evolution process.

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と従来のオブジェクト指向開発の特徴を比較したのが**表 2** である．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という点では，両者にはさほど違いはない．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．

まず，各工程について，**図 10** に通常のオブジェクト指向開発とフレームワークを用いたとき開発とを，それぞれ従来手法とかける人月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，かなり長いといわれている[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．

同じ機能を持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すむ．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，ユーザの要

表2 フレームワークを用いた開発の特徴
Table 2 Traditional OO development and development using frameworks.

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	●再利用率が高く，新規設計・実装量少ない
●多人数を前提	●少人数で開発可能
●ドキュメント指向	●ドキュメント後付け
●大人数のプロジェクト管理	●少人数プロジェクト管理
●ラウンドタイムの長いプロトタイピング	●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

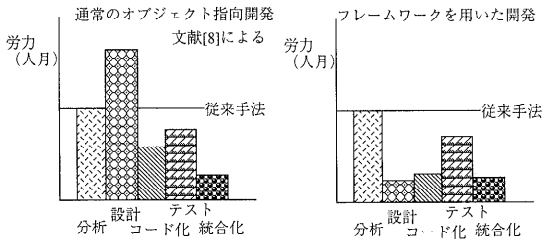


図10 労力配分の違い
Fig. 10 Comparison in terms of effort distribution.

求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．

2.3節で紹介した商用インタネット管理開発には8週間を要したが，その間画面や機能などの仕様変更は都合3回行っている．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う．特に機能設計レベルが大幅に削減できたことが分かる．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は向上した．

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持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起きるすべての要求を見込んで，かつそれに最適な構成を持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．我々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おり，これが現在の再利用率の高さのもととなっている．たとえば，図7で示したようなGlobal Viewアプリケーションはネットワーク全体を監視するアプリケーションであるから，応用システムごとに必要となるが，以前は少しずつ異なる要求に基づき，少しずつ異なるアプリケーションをその都度開発していた．このときは低レベルの部品の再利用は行っていたものの再利用率はさして高いものではなかった．このようなアプリケーションをいくつかばらばらに作成した後に，それらの要求をまとめてフレームワーク化したわけだが，これが現在の93%の再利用率につながっている．

ここでは，アプリケーション開発の場合とは異なり，設計フェーズが最も重要となる．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った議論を行うのが必須である．

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，開発マネージャとは開発チームのリーダであり，チーム内の作業分担，進捗管理などを含むチーム管理を行う役割の人，またシニアマネージャとは通常開発マネージャより上司にあたり，開発への投資の決定権限を持つ役割の人のことを想定している．

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，我々は，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，その設計を評価する手法である．たとえば，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，リスクの回避を行うことができる．

3.2 教　育

フレームワークの構築およびフレームワークを用いた開発において，最も重要なものの1つは，要員・体

制である．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]．我々のプロジェクトでは，フレームワーク（およ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，および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[6]が，我々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ある．

OJT（On the Job Training）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が，OJTの手順としては，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始めるようにしている．インセンティブを与えるという意味からも，実際のプロジェクトの一部を切り出すようにしている．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．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，一渡りの教育は終了する．

このほか，教育的に効果が高いのが，設計レビューおよびコーディングレビューである．レビューの中では，設計・コーディングの欠点の指摘だけではなく，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，議論する必要がある．また，レビューに参加するメンバに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は，レビューの管理者の責務として特に重要である．

上級者の関与の割合も重要であると感じている．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．

3.3 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，CASEツール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，現状のところよく分かっていない．本節では，適用範囲の条件について考察する．フレームワークは「システムあるいはサブシステムのスケルトンインプリメンテーション」[1]と定義されていることからも分かるように，以下の議論はシステム全体だけでなく，その部分的な適用にも同様にあてはまる．

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の条件は対象のとら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，第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．

モデルが安定するとは，時間的な推移によって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，それはまったく徒労である．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，たと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，人為物は何らかの要因で大きく構造が変更されることがある．たとえば，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，たと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．

3.4 体　　制

我々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．1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1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始めた．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，複数のカスタマイズ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．

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%を超えるものであり，設計・コーディングの期間が大幅に短縮できる．また，開発の経験から得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，我々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，一方，フレームワーク成熟化させるプロセスにおいては，設計の重要性や良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．我々は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．

また，本事例では主にフレームワーク構築の立場から議論しているが，一方で利用側の立場からの研究も急務である．フレームワークを利用できるかどうか，複数のフレームワークの選択など，さまざまな問題がある．これらの考慮ポイントとしては，ドキュメンテーションの充実性，パフォーマンス，動作プラットフォーム，ツールとの連動性，バグなどの品質条件など，通常の再利用ライブラリと同様の点があげられる．フレームワーク固有の点としては，フレームワークベンダの技術サポートの可否や，構築予定アプリケーションとの相性などがある．実務的にはフレームワーク，およびそのベンダの評判が重要な判断材料となることが多いが，アプリケーションとの相性問題については基準がなく，研究が待たれるところである．

フレームワークの研究の方法としては，今後1つ1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．

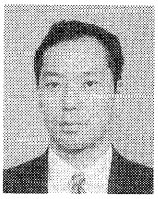
謝辞　本研究のご指導をいただいたNTT研究開発推進部今瀬真氏に感謝します．また，本システム開発にあたりご尽力いただいているNTTソフトウェア研究所蔭山克禎氏，彦坂洋子さん，NTTソフトウェア柴原吉秀氏，NTTネットワークサービス研究所中西弘毅氏，その他のみなさまに感謝いたします．

参 考 文 献

1) Wirfs-Brock, R.J. and Johnson, R.: Surveying Current Research in Object-oriented Design, *Comm. ACM*, Vol.33, No.9, pp.104–124 (1990).
2) Schmucker, K.J.: MacApp: An Application Framework, *Byte*, pp.189–193, Aug. (1986).
3) Lewis, T., et al.: *Object Oriented Application*

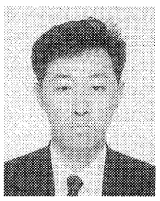
Framework, Manning Publications (1995).
4) Profrock, A.K., Tsichritzis, D., Muller, G. and Ader M.: ITHACA: An Integrated Toolkit for Highly Advanced Computer Applications, *Object-Oriented Development*, Tsichritzis, D. (Ed.), Universite de Geneve, pp.321–344 (1989).
5) Opdyke, W.F. and Johnson, R.E.: Refactoring: An Aid in Designing Application Frameworks and Evolving Object-oriented Systems, *Proc. 1990 Symposium on Object-Oriented Programming Emphasizing Practical Applications (SOOPPA)*, Sept. (1990).
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*, Benjamin/Cummings Publishing (1991).
9) Donovan, J.J.: *Business Re-engineering with Informtion Technology*, Prentice Hall (1994).
10) 蔭山，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研究報告，DSM-95-07040, No.2, pp.349–355 (1995).
11) 荒野ほか：オブジェクト指向フレームワーク再利用性の一実験，第49回情報処理学会全国大会論文集（5），pp.5-191–5-192 (1994).
12) 本位田ほか（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，オブジェクト指向特集号，電子情報通信学会論文誌（D-II），pp.719–728 (1996).
14) Gamma, E.: *Design Patterns*, Addison Wesley (1995).
15) Stallings, W.: *Network Management*, IEEE Computer Society Press (1993).
16) 藤崎，蔭山，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, pp.xx–xx (1995).
17) Huni, H., Johnson, R. and Engel R.: A Framework for Network Protocol Software, *Proc. OOPSLA'95*, pp.358–369 (1995).
18) Schmid, H.A.: Creating the Architecture of a Manufacturing Framework by Design Patterns, *Proc. OOPSLA'95*, pp.370–384 (1995).
19) Bruegge, B., Gottschalk, T. and Luo, B.: A Framework for Dynamic Program Analyzers, *Proc. OOPSLA'93*, pp.65–82 (1993).

(平成8年3月18日受付)
(平成9年4月 3 日採録)



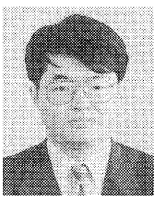
荒野 高志（正会員）

昭和59年東京大学理学部情報科学科卒業．昭和61年同大学院修士課程修了．同年，日本電信電話（株）ソフトウェア生産技術研究所入所．以来，フレームワーク構築・利用法，メトリクス，分散オブジェクトなどのオブジェクト指向ソフト開発法とそれを応用したインタネット管理システムの研究に従事．平成3〜4年イリノイ大客員研究員．現在，マルチメディア実験推進室担当課長．情報処理学会誌編集委員，電子情報通信学会生涯教育講座講師，IEEE COMPSAC'96や情処オブジェクト指向シンポジウム等のプログラム委員など．著書「オブジェクト指向分析・設計—開発現場に見る実践の秘訣」（共著，共立出版）



青野 博（正会員）

昭和39年生．昭和61年九州工業大学工学部情報工学科卒業．平成元年同大学院修士課程修了．同年，日本電信電話（株）ソフトウェア研究所に入所．以来，開発環境，メトリクス，再利用，性能評価などのオブジェクト指向ソフトウェア技術および分散オブジェクト技術に従事．また，分散オブジェクト技術を応用したネットワーク管理システムの研究/開発を行い，平成8年NTTソフトウェア（株）に出向し実用化を行っている．



藤崎 智宏（正会員）

昭和41年生．平成元年東京工業大学理学部情報科学科卒業．平成3年同大学院修士課程修了．同年，日本電信電話（株）ソフトウェア研究所入所．分散オブジェクト環境，コンピュータネットワークの管理に関係する研究に従事．