ネットワーク管理ソフトウェアの動的構成

江頭 徹　　福井眞吾
NEC C&C 研究所

ネットワークを構成する機器群に追加や拡張などの構成変更の頻度が高い場合においても対応できるネットワーク管理システムを実現するために、一部のOSが実装しているコントロールパネルや、コンピュータのbasic input output system (BIOS) に類似するアーキテクチャを提案する。さらに、そのアーキテクチャを基に、Javaアプレットと World Wide Web (WWW) の仕組みを用いてテストベッドシステムを試作した。

Dynamic Arrangement of Network Managemenet Software

Toru EGASHIRA　　Shingo FUKUI
C&C Research Laboratories, NEC Corp.

This paper describes network management architecture for networks to which new equipments are frequently added. The architecture is similar to computer's basic input output system (BIOS) and control-panel: managed equipment contains programs for managers, and a manager loads the programs to manage the equipment. This paper also describes the test-bed system, which employs World Wide Web (WWW) and Java applet techniques.

1　はじめに

近年におけるインターネットやイントラネットの急速な浸透と拡大につれて、ネットワークを構成する機器群にも追加や拡張などの構成変更が頻繁になって来ている。また、続々と開発されるプロトコルやサービスをサポートするためにモデルチェンジし機能を拡張されたネットワーク機器が次々に市場に投入されている。

ネットワーク機器の監視、制御をその中心的機能とするネットワーク管理システムにとって、そのような機器構成や機器の機能の頻繁な変化は少なからず問題となる。ネットワークに新しく追加された機器や機器の新しい機能を管理するために、マネージャソフトウェア (以下マネージャ) およびそのソフトウェアを通じて操作するオペレータ (人間) の知識を拡張する必要があるためである。

インターネットやイントラネットにおいて標準的に用いられているSNMPベースの管理フレームワークにおいては、Management Information Base (MIB) の定義をStructure of Management Information (SMI) [1, 2] に従いMIBモジュールと呼ばれる文書として記述することにより、交換することが可能となっている。MIB-II等の標準的なMIBの他、ベンダ独自に定義されたMIBの多くもMIBモジュールがネットワーク上に公開されており、入手可能となっている。ネットワーク管理システム製品によってはMIBコンパイラ [3] と呼ばれるソフトウェアを備え、これらのMIBモジュールを内部形式として取り込み、新しく定義されたMIBにも管理アプリケーションを対応させる事が可能となっている物もある。しかし、これはMIB中のオブジェクトの型やオブジェクトが変更可能か等が判明しアクセス手段が確立するに過ぎず、個々のオブ

ジェクトと機器の機能との関連やそれに伴いオブジェクトが受ける動的な制約など、オブジェクトのセマンティクスは管理システムには反映されることはない。これは、オブジェクトのセマンティクスはMIBモジュール中に自然言語で記述されているためである。よって、オブジェクトのセマンティクスを理解し必要な操作を判断する仕事はすべてオペレータのものとなり、オペレータの負担となる。

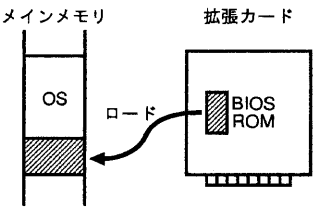
本稿では、パーソナルコンピュータ(PC)のオペレーティングシステム(OS)における機器構成変化への対応法を参考に、マネージャを動的に更新し管理対象機器のセマンティクスを取り込む事によりオペレータの負担の軽減を目指すアーキテクチャを提案する。また、そのアーキテクチャをもとに作成したテストベッドシステムについて報告する。

2 PC OSの動的機能拡張方式

PCはそれを用いるユーザのニーズが多岐に渡るため、それらを満たすための様々な種類の拡張機器が市場に出回っている。そのような拡張機器をインストールするのも多くの場合ユーザ自身であるが、ユーザすべてにコンピュータに関する高度な知識があることは期待できないため、OS側や拡張機器自身にインストールを簡単に行うための工夫が施されている。例えば、一部のGUIベースのOSが備えるコントロールパネルや、拡張カード上のBIOSがそれである:

コントロールパネル PCにグラフィックカードやI/Oカード等の拡張カードをインストールする際には、添付されているフロッピー等を用いてソフトウェアを同時にインストールすることを求められる場合がある。特に、その拡張カードが高機能な場合や特殊な機能を持つ場合に多く、添付ソフトウェアには通常、その拡張カードの機能を最大限に生かすための仕組みが導入されている。例えば、OSが標準でサポートしない機能の制御には専用の設定ソフトウェアが用意され、それを用いて制御するようになっている。MacOSやWindows等のOSにおいては、このような設定ソフトはコントロールパネルと呼ばれるインタフェースによりある程度隠蔽されており、新しい設定ソフトを導入した場合には、あたかもコントロールパネル自体が拡張され、設定項目が増えたかのような見かけを提供している。

BIOS 前述のコントロールパネルは通常、拡張カードに附属するフロッピー等のメディアからインストールするが、BIOSは通常、ROMの形で拡張カード上に実装されているプログラムである。I/Oの基本機能など、OSがブートする以前に必要となる拡張カードの機能を制御するために、OSのサポート無しで使用可能なROMの形で提供されている。



BIOSはCPUのメモリ空間にロードされて利用される

図 1: BIOS

拡張カードに搭載されているプログラムであっても、BIOSではないものもある。カードによってはそれ自身に制御用等の目的でプロセサを持つ物もあり、そのプロセサのためのプログラムがROMとして搭載されていることもある。BIOSがそれらと異なるのは、BIOSはカード自身が用いるのではなく、カードをアクセスするCPUがそれを自身のメモリ空間上にロードして実行するプログラムであると言う点である(図1)。すなわち、利用される側が利用する者のためのプログラムを内蔵している、という点が特徴である。

BIOSはCPU依存の機械語で記述され、容量の制限等から基本的なAPIを提供するだけの場合がほとんどであるが、高級言語で記述しアーキテクチャ依存性を低くしたものや、ブート時に特定のキーを押していると拡張カードの設定メニューを表示する等、高度な機能を持つものもある。

3 ネットワーク管理への応用

コントロールパネルやBIOSが持つ重要な性質は次のとおりである。

- ソフトウェアが機器に附属してくる
- 機器が持つ機能の設定操作方法をプログラムとして持つ(機器のセマンティクスを動作として内包)
- ユーザインタフェースを含む

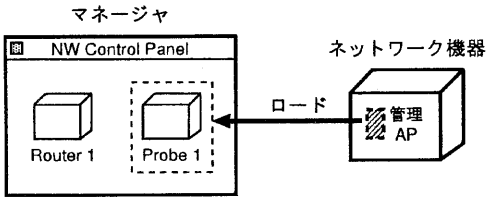


図 2: 提案アーキテクチャ

この2番目と3番目の性質は特に重要である。例えば、グラフィックスカードでは解像度（縦横のドット数）と色数が指定できるようになっているが、完全に独立して設定できるわけではなく、640x400 ドットでは6万色指定可能だが 1600x1200 ドットでは 256 色しか指定できない場合がある。このように相互依存関係があるにも関わらず解像度と色数を独立に設定可能 (これはすなわち機器のセマンティクスを持たないことを意味する) とするユーザインタフェースでは、どのような値の組合せが可能なのかを機器の仕様書等を調べ妥当な範囲の値を入力する作業をオペレータに要求する事になる。しかし、コントロールパネル方式だと解像度を選んだ段階で、選択可能な色数の候補が限定されて提供される。これは、ユーザインタフェースにパラメタの相互依存関係 (機器のセマンティクス) を内包するプログラムが共に提供される事で初めて可能となる。

このような仕組みをネットワーク管理に応用することにより、機器構成変化に容易に対応できる管理システムを構築することができると考え、提案するのが次のようなアーキテクチャである (図 2):

- マネージャにはコントロールパネル相当の動的に拡張可能な “インタフェースの土台” を用意し、一方、
- 管理対象の機器には、自分自身を制御するためのインタフェース機能のプログラムを BIOS のように内蔵させる。そして、
- 監視制御作業を行う際に、マネージャはそのプログラムを機器からロードし、インタフェースを拡張する。

機器構成が変化した場合においてもインタフェースも同時に更新されるため、対応に要する工数は発生せず、マネージャが異なるノード上などに複数存在する環境においても更新は容易である。

近年までは、マネージャの実行環境である管理ステーションのアーキテクチャに依存しないポータブルなインタフェースプログラムを記述する手段が確立されていなかったため、このようなシステムは普及が見込めなかった。しかし、Java 言語 [4] およびその実行環境が WWW ブラウザという形で広く普及し、GUI を持つアーキテクチャ独立のプログラムをネットワーク経由でロードし利用することが日常的に行われている今日においては、このような動的インタフェースは比較的容易に実現できる。

4 PC OS との相違点

ネットワーク管理はいくつかの点において PC の機器管理とは本質的に異なる性質があり、PC における BIOS やコントロールパネル相当のアーキテクチャをネットワーク管理システムに適用するにあたっては考慮する必要がある。

マネージャが複数の可能性 一般にコントロールパネルにおいては、同一のプログラムは同時に1つのプロセスのみが起動できるようになっており、またそれで仕様上十分である事から、複数のプロセスが並行動作することは考慮されていない。

一方、ネットワーク管理ではオペレータは一人とは限らず、異なるノード上などにおいて複数のプロセスが並行して起動される事が想定される。複数のマネージャプロセスからのアクセスが競合を起こさない範囲 (読み出しのみ可能等) で、もしくは適切なアクセス排他制御を行うことにより、機器の状態に矛盾が起こらないようにする必要がある。

持続的使用 PC においてコントロールパネルが利用される局面は、機器の初期設定やその後の設定確認、設定変更等、一時的な使用がほとんどである。そのため、用いられるインタフェースの形式としても、多くの場合プロパティシート型すなわちタブ等を用いた選択切替え式のインタフェースとなっている [6]。この形式では、インタフェースをプロパティの種類ごとに整理したそれぞれを単位インタフェースとして、本のページ状に複数重ねておき、タブを使ってそのうちの1つを選択しアクティブにする方法 (図 3) のように、用意された多くの単位インタフェースのうち選択された1つのみがアクティブ (可視または操作を受け付ける) となる。

一方、ネットワーク管理ではその他にインタフェースが持続的に用いられる局面も多い。例えば、リアルタイムな性能の監視をするためにポーリング採取した値をグラフ等の形で常時表示する場合

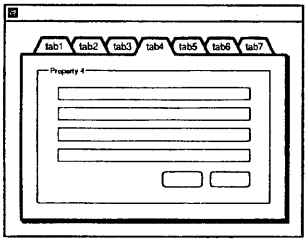


図 3: タブを用いたインタフェース形式

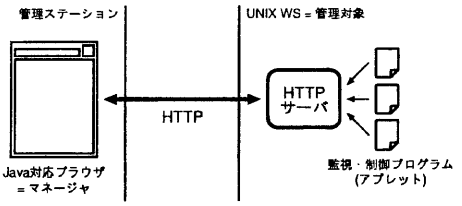


図 4: テストベッドシステム

などである。よって、単位インタフェースを択一的に選択させる他に、各単位インタフェースを独立に持続的に使用することもできるインタフェース形式とすることが必要となる。

5 テストベッドシステム

本提案のネットワーク管理アーキテクチャの実用化を目指し、テストベッドシステムを作成した。

テストベッドシステムは、管理対象機器としてはUNIX ワークステーション (WS)、通信プロトコルとしてはTCP/IP 上の HTTP として、WS に用意したプログラムを HTTP サーバプログラム経由で管理ステーションにロード、実行し、WS を監視制御するシステムとして構築した (図 4)。WS に用意する監視制御プログラムは Java[4] アプレットとして作成しており、Java アプレット対応 WWW ブラウザでそれを実行する。すなわち、WWW ブラウザおよびアプレットがマネージャとなる。Java アプレットとすることにより、GUI を含めたポータビリティが得られ、管理ステーションのプラットフォーム依存性が少なくなるとともに、Java 言語の特長であるマルチスレッド機能により、小機能単位に分割された多数のプログラムをそれぞれ独立に、かつ並行して動作させる事ができる。

マネージャにロードされ実行されたアプレットは、オペレータの操作に応じて、もしくは定期的にポーリングで管理対象機器からデータを取得し、視覚化してオペレータに提供する。オペレータから機器の状態を変更する指示があった場合には、それを機器に伝える。前述のアプレットの転送目的以外に、テストベッドシステムではこのような機器からのデータの取得および機器への状態変更指示にも HTTP プロトコルを用いている。SNMP のトラップに相当する、機器からのイベント通知は現在の所実装していないが、Java のソケット通信用 API を用いてマネージャがイベント受信を行うのも容易と考えられる。

テストベッドシステムでは、4節で述べたネットワーク管理の特徴を考慮し、実装を行っている。以下それらについて説明する。

複数マネージャへの対応 複数マネージャからの操作の競合を防ぐため、管理対象機器内でロック処理によるアクセス排他制御をしている。

図 5でこの処理を説明する。ネットワーク機器を管理するマネージャはまず、その機器 (の HTTP サーバ) にアプレットを要求する (同図 (a))。機器の HTTP サーバはそれを受けて、要求されたアプレットを送り返す (同 (b))。ここまではこのアプレットが機器の状態を操作する可能性の有無に関わらず共通であるが、機器の状態を操作する可能性のあるアプレットは、マネージャにロードされ実行が開始 (または再開) される際に呼び出される、クラスAppletのstart() メソッドで機器にロック確保を依頼する (同 (c))。すでに他のアプレットがロックを保持している場合には、状態の操作は断念する。また、ボタンやチェックボックス等のインタフェース部品をdisable() メソッドにより操作不可能な状態で表示することにより、操作ができないことをオペレータにも知らせる。アプレットの実行が中断する際にはクラスApplet のstop() メソッドが呼ばれるため、ロックを保持しているアプレットはそのメソッドの処理として機器にロック解除を依頼する (同 (d))。このようにしてロックを管理し、複数マネージャからの操作を排他制御する。

このように、テストベッドシステムにおいてはロック機能を non blocking 型の advisory locking として実装しているため、このロック方式を知らないプログラムがロックを無視したアクセスを行ない機器の状態に矛盾を起こす事もあり得る。しかし、管理対象機器自身からプログラムを調達する本提案のアーキテクチャにすべてのノードが従うならば、

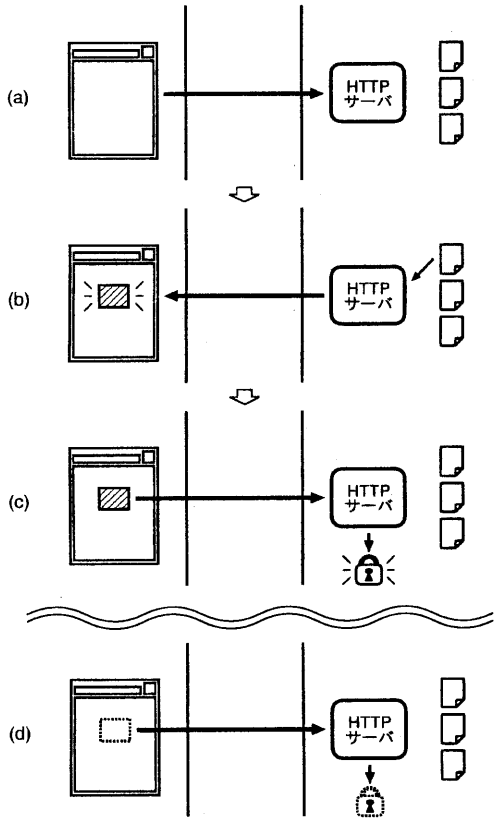


図 5: ロック処理

機器をアクセスするプログラムは各マネージャともその機器からロードした同一のものである。よって、ロックプロトコルは守られることが期待でき、advisory locking でも実用となる。また、同様のプロトコルにより複数マネージャの排他制御を行なっている例としては RMON MIB[5] があり、実用化されている。

インタフェースのカスタマイズ　前述のように、ネットワーク管理においては PC のコントロールパネルのように単位インタフェースを切替えて利用する形式の他に、持続性のあるインタフェースを実現する必要がある。テストベッドシステムにおいては、単位インタフェースをアプレットとして実装し、単位インタフェースの切替えはブラウザのウィンドウのスクロール操作として対応づけている。ある単位インタフェース (アプレット) を持続的に利用する必要がある場合には、簡単な編集機能を通

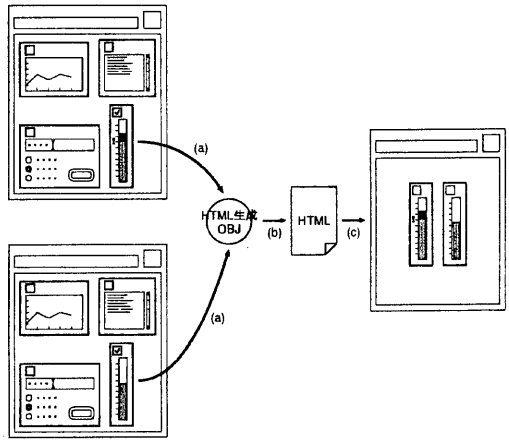


図 6: アプレットの選択

じて別のウィンドウにそのアプレットをコピーすることにより、元のウィンドウの操作と独立して持続的に利用する事を可能としている。この編集機能としては、すでに多数製品化されているビジュアルな HTML 編集ソフトウェアを用いる事も効果的であるが、テストベッドシステム作成時にはアプレットを含む HTML ファイルをドラッグ・アンド・ドロップのように簡単な操作で編集できるものは存在しなかったため、独自の編集機能を実装している。テストベッドシステムの各アプレットには編集機能専用のボタンが配置されており、別のウィンドウに用意された「選択終了」ボタンと組み合わせて操作する事によりオペレータが持続的に利用するアプレットを選択する事を可能としている。オペレータが必要なアプレットのボタンを押して選択し、「選択終了」ボタンを押すと、選択したアプレットだけからなる新しいウィンドウが作成されるようになっている。この編集機能の処理は図 6のように行っている。ブラウザ側には HTML 文書を生成するオブジェクトをあらかじめ用意している。オペレータからボタンを操作され選択されたアプレットは、そのオブジェクトに自身の情報を伝えて (図 6(a))、オブジェクトが生成する HTML 文書にそのアプレットの再生に必要な HTML タグを記述させる (b)。その後オペレータが「選択終了」ボタンを操作すると、オブジェクトが生成した HTML 文書がブラウザに読み込まれ (c)、選択されたアプレットのみからなるウィンドウが作成される。

実際の画面表示例を図 7に示す。

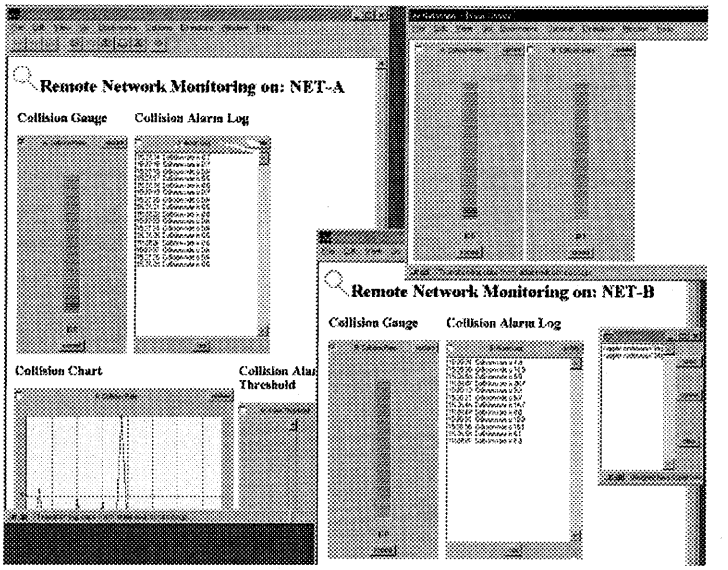


図 7: テストベッドシステムの画面

6 おわりに

今回試作したシステムは管理対象機器としてUNIX WS、プロトコルとしてHTTPを用いた暫定的なものである。本システムをベースとして、監視・制御アプレットをルータ等、実際のネットワーク機器に内蔵し、JavaのSNMP API拡張[7, 8]等を用いてSNMPプロトコル経由でアプレットをロードする拡張を行ってゆく。

その他、改良すべき点として、今回のシステムでは簡易な編集機能によりウィンドウ内のアプレットの組み合わせの編集を可能としたが、HTMLのテーブル機能や、提案されているレイアウト拡張[9]等によりアプレットの配置を含めた編集を許し、オペレーションの局面に応じた使いやすいレイアウトを実現できる事が望ましい。今後の課題としてテストベッドの改良を進めてゆく。

参考文献

[1] Rose, M. and McCloghrie, K., *Structure and Identification of Management Information for TCP/IP-based Internets*, STD 16, RFC 1155, RFC 1212, May 1990.

[2] Case, J. et al., *Structure of Management Information for Version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1902, SNMPv2 Working Group, January 1996.

[3] Roberts, S., An Introduction to SNMP MIB compilers, In *The Simple Times*, Vol. 2, No. 1, 1993.

[4] Arnold, K. and Gosling, J., *The Java Programming Language*, Addison Wesley, 1996.

[5] Waldbusser, S., *Remote Network Monitoring Management Information Base*, RFC 1757, Carnegie Mellon University, February 1995.

[6] *Windows Interface Guidelines for Software Design*, Microsoft Corporation, July 1995.

[7] *Java Management API*, `http://www.javasoft.com/products/JavaManagement/`

[8] *ADVENT JAVA SNMP Package*, `http://www.adventnet.com/snmp_api.html`

[9] Bos, B. et al., *Frame-based layout via Style Sheets*, W3C NOTE, `http://www.w3.org/pub/WWW/TR/NOTE-layout.html`, 1996.