システム管理に必要な情報を自動的に収集・分析するための機構 (magP) に関する研究

東京工業大学 大学院
情報理工学研究科 数理・計算科学専攻
中嶋 良彰
(学籍番号 95M37187)

平成8年度修士論文

指導教官 大野 浩之 講師

平成9年1月24日

目次

第1章

はじめに

システム管理において、作業のために必要な情報を収集し分析することは、管理者個人の知識や経験に頼る部分が多く複雑で時間のかかる作業である。この原因の１つとして、情報を収集するために、さまざまなコマンドや管理プロトコルの利用、各種サービスの動作ログの参照などが必要とされることが考えられる。現在、一般に普及しているシステム管理支援アプリケーションの多くは、CMIPやSNMPなどの管理プロトコルを主な情報の収集手段として開発されている。しかし、実際にシステム管理において必要とされる情報はこれらの管理プロトコルでは十分に収集できない。また、インターネットの爆発的な普及によって、システムを24時間正常に運用することや、セキュリティの保護が重要になっている。そのため、システム管理者が直接作業を行うことができない状況にあっても、障害発生や不正な侵入などを検出できる必要がある。

そこで、特にシステム管理における情報の収集作業および分析作業を中心に、自動化あるいは半自動化を行うことを考え、システム管理支援機構magPの設計と実装を行った。magPは知識ライブラリと呼ばれる管理者の知識に相当するライブラリを利用して、システム管理に必要な情報の収集および整理を支援する機構である。また、情報の収集手段には、実際にシステム管理において利用されるさまざまな道具を利用できるため、必要な情報の大部分を収集できる。知識ライブラリは、情報の収集方法、分析方法、および管理者への報告方法の記述の集合であり、管理者の知識に相当する。magPは、この知識ライブラリによって、既存の情報収集の手段を組み合わせて統合するとともに、あらかじめ用意された標準的なインタフェースを採用して、新たなツールの開発を容易にしている。知識ライブラリの主な特徴は、構造の階層化によって、再利用性が向上することやシステム依存部が分離されることである。検出された事実は、報告機構内に用意されている手段の中から最適なものを用いて管理者に通知される。また、あらかじめ用意されている知識ライブラリでは作業に不十分である場合、動作中のmagPの知識ライブラリに対してライブラリの追加・削除を行うことができる。さらに、知識ライブラリの拡張に用いる知識ライブラリ記述言語ついては、オブジェクト指向の利用やさまざまな工夫によって記述力の向上を図っている。

本論文では、考案した知識ライブラリ記述言語について、その記述力の向上、および記述量の減少を達成するための工夫を中心に述べ、magPがシステム管理作業の改善に貢献することを示す。まず、第2章で管理アプリケーションとその開発環境の現状と、それらが本来満たすべき要素について述べる。第3章では、本論文で設計・実装を行ったシステ

ム magP の主な働きと構成について述べ、第 4章で magP の実行例を示す。続く、第 5章では、magP によって達成される管理作業の改善点について述べ、第 6章で現在の magP の課題を整理し今後の開発計画について述べる。最後に第 7章はまとめである。

第 2 章

管理アプリケーションの現状と問題点

この章では、管理アプリケーションおよびその開発環境の現状と、それらが本来満たすべき要素について議論する。

2.1 管理アプリケーションの現状

ネットワークの拡大によって、分散システムが一般的になり、システムの管理作業はますます複雑化している。OSI 管理モデル [20] によれば、システム管理は、構成管理、性能管理、障害管理、課金管理、機密管理の 5 つに大別される。どの作業においても、管理対象を調査し、適切な操作を行うという工程は共通しており、このような作業を支援するために多くのアプリケーションが研究開発されている。本論文では、このようなシステム管理を支援するためのアプリケーションを管理アプリケーションと呼ぶことにする。

従来の管理アプリケーションは、システム管理者によるその場かぎりの設計によって開発され利用されることが多かった。そのため、一般的な問題に対して広い適用範囲をもつような管理アプリケーションは少ない。これは、実際にシステム管理の際に発生する個々の重要な問題を解決するためには、その問題に特化して設計および実装された管理アプリケーションが効果を発揮しやすいためである。そして、システム管理者によって開発されたさまざまな管理アプリケーションが存在し、システム管理は標準的な枠組を持たず繁雑のものになっている。

現在、システム管理に必要な情報の収集方法として一般に広く利用されているものに、CMIP[4] や SNMP[1] といった管理プロトコルがある。これらを利用すれば、遠隔地にある管理対象に対する情報収集や設定変更を容易に行うことができる (図 2.1)。この際、SNMP では MIB[11][12] によってさまざまな機器に統一的なインタフェースを与えることができる。また、現在、システムの監視ソフトフェアは CMIP や SNMP を利用しているものがほとんどである。

しかし、CMIP や SNMP によって得られる情報は、管理対象の特定の部分の状態を表すような比較的細かな情報ばかりで、管理対象の状態をおおまかに把握したい場合には扱い

にくい。また、実際、熟練したシステム管理者は`ping`[1]、`tcpdump`[2]、`netstat`[3] のようなコマンドによって得られる情報や、経路制御、電子メール、WWW などの各種サービスの動作ログなどを利用することが多く [22]、CMIP や SNMP だけでは十分なシステム管理を行うことができない。

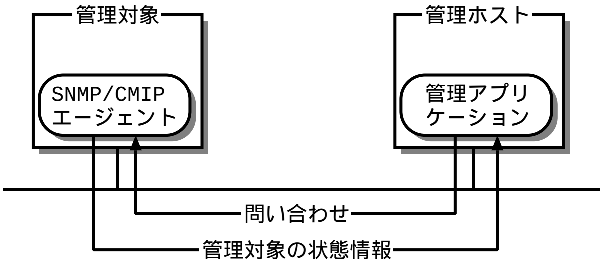


図 2.1 CMIP や SNMP のエージェントを通じて管理対象の状態情報を得る

CMIP や SNMP に対する API を規定することによって、CMIP や SNMP を利用した管理アプリケーションの開発を容易にする試みも行われている。XMP [4] や XOM [5] はこのような試みの 1 つで、C 言語に基づいた API を提供している (図 2.2) [19][18]。さらに、最近では分散オブジェクト指向技術の標準である CORBA[14][21] に CMIP や SNMP を組み合わせることによって、管理アプリケーションを記述するための新たな枠組を提供するなどの試みも行われはじめている [10][8]。しかし、そのような枠組を利用して、アプリケーションを開発することは、C 言語や CORBA や IDL [6] などの修得、ORB(Object Request

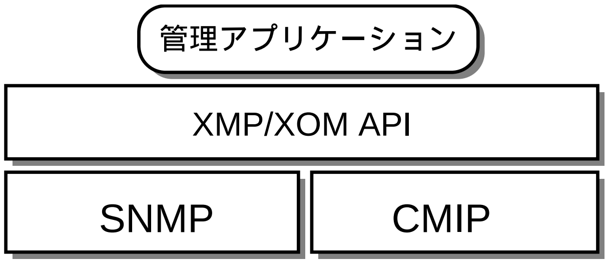


図 2.2 XMP/XOM を用いた管理アプリケーションの開発

---

[1] ICMP の ECHO_REQUEST パケットを指定したホストへ送り接続性の確認、通信速度の計測を行うことができるコマンド。

[2] ネットワーク上のトラフィックをダンプし観測するためのコマンド。

[3] ネットワークインタフェースの状態、経路テーブルなど、ネットワークに関する各種データを取得するためのコマンド。

[4] X/Open Management Protocols

[5] X/Open OSI-Abstract-Data Manipulation

[6] CORBA におけるオブジェクトのインタフェース記述言語

Broker) の整備などが必要である。

以上のように、現在の広く利用されている管理アプリケーションのほとんどはCMIP や SNMP を中心として設計されているため、実際にシステム管理に必要な情報を十分に扱うことができず、実用上非常に問題がある。また、必要な情報を十分に収集するためには、さまざまなコマンドや各種サービスのログを利用する必要があるが、これらはいずれも異なるインタフェースや構造を持っている。そこで、このような問題を解決するために、豊富に存在する情報の収集手段を効率良く利用できる管理アプリケーションおよびその開発環境が望まれている [8]。

2.2 管理アプリケーションが満たすべき要素

管理アプリケーションを開発するにあたって、特に考慮すべき要素は次の通りである。

- さまざまな情報の収集が可能
- 必要な計算機リソースが少ない
- インテグレーションが容易
- 高いポータビリティ
- セキュリティに優れている
- リビジョン管理が容易

以下では、それぞれについて説明する。また、3章以降では、これらの要素を考慮した上で設計・実装を行ったシステム管理支援機構 (magP) について説明する。

さまざまな情報の収集が可能

2.1節で述べた通り、システム管理に必要な情報の中にはSNMP やCMIP などの標準的な管理プロトコルによって収集できないものも多い。例えば、DNS [7]や NIS [8]の設定情報、システムの動作記録や電子メールサービスの動作記録などである。そのような情報を利用するために、管理アプリケーションは、SNMP や CMIP 以外にもさまざまコマンドやサービスを効率良く統合しながら情報を収集できる必要がある。

また、各管理対象から収集した管理情報は、多くの場合、文字列、整数値、浮動小数などさまざまタイプの情報である。そこで、管理アプリケーションはこのようなさまざまなタイプの情報を扱える必要がある。また、そのためのアプローチもいくつか考案されている。例えば、「String Notation[2]」はその一例である。

---

[7] Domain Name System

[8] Network Information Service

必要な計算機リソースが少ない

管理アプリケーションがシステムに与える負荷や要求するリソースは、少くともシステムの他の処理に影響を与えない程度に小さくなければならない。管理アプリケーションが管理すべきシステムを圧迫してはならない。

インテグレーションが容易

2.1節で述べた通り、あらゆる問題に適用可能な管理アプリケーションを開発することは難しい。また、発生しうるすべての問題に対して管理アプリケーションを開発することは不可能である。そこで、過去に開発された管理アプリケーションを再利用し、組み合わせることを考える。これによって新たな管理アプリケーションを開発することができれば、過去に開発された有用な管理アプリケーションの恩恵を得るだけでなく、開発にかかるコストは大幅に軽減することになる。また、そのためには将来の統合を考慮した上で管理アプリケーションを開発する必要がある。

高いポータビリティ

管理アプリケーションは高いポータビリティを持っていることも重要である。ある組織内のネットワークが同一のオペレーティングシステムで構成されていることは稀で、多くの場合、多種多様なシステムで構成されている。そのようなネットワークを管理するためには、管理アプリケーションが高いポータビリティを持ち、異なるシステム上で動作する必要がある。また、管理アプリケーションがシステムに依存しないコードで記述されていればさらに良いと言える。

リビジョン管理が容易

分散システムでは、管理アプリケーション自体もネットワーク上で分散して運用される。そのため、分散して動作している複数の管理アプリケーションのリビジョン管理が重要になる。リビジョンの管理がおろそかになると、版の違いによって予想外の問題が生じる可能性がある。しかし、管理アプリケーションの数が増加するにつれて、新しい版の管理アプリケーションの導入は多大な労力を要するため、なんらかの方法でリビジョンの管理を支援する必要がある。

セキュリティに優れている

管理アプリケーションは、システム管理に関する重要な設定情報を扱うため、セキュリティも深く考慮されている必要がある。優れた認証機構や暗号化技術を導入などによって、管理アプリケーションを重大なセキュリティホールにしないことが必要である。

第3章

magPの設計と実装

本章では、システム管理に必要な情報を自動的に収集・分析するための機構 (magP[1]) の主な働きと構成について述べる。magPは2.2節で述べた管理アプリケーションの満たすべき要素を踏まえた上で設計および実装されたシステムである。

3.1 magPの働き

magPはユーザ[2]に次のような機能を提供する。

- システム管理に必要な情報の自動収集
- 情報を整理しシステム管理者の分析作業を支援
- 情報から推測される事実の報告
- 多様な情報収集手段を統合し共通インタフェースを提供
- 抽象度の高い分析規則の記述

システム管理において、システム管理者は作業に必要な情報を収集するために、`ping`、`traceroute`、`netstat`などのコマンドの実行、電子メール・経路制御などの各種サービスのログファイルの参照、SNMPによる管理対象への問い合わせなどを行う。しかし、このような情報の収集作業は、異なるインタフェースを持った多くの情報収集手段を効率良く組み合わせて使用する必要があり、豊富な経験と時間を必要とする作業である。そこで、magPはこの情報の収集作業および収集した情報の分析作業を可能な限り自動化することによって、作業を支援する。また、収集した情報の分析作業が完全に終了しないとしても、情報を整理しより把握しやすい形で提示できれば、システム管理者の分析作業は大幅に改善される。magPは知識ライブラリと呼ばれるシステム管理者の知識に相当するライブラリを用いて動作する (図3.1)。この知識ライブラリはクラス定義を単位として構成され、各クラス定義にはシステム管理に必要な情報の収集方法、分析方法、およびシステム管理者

---

[1] magPの発音は「まぐぴい」である。この名称は「magpie(かささぎ)」という黒白の羽毛で鳴き声がやかましく、さまざまな小さなもの (システム管理に必要な情報) を巣に集めてしまう鳥の名前に由来している。

[2] 本論文ではmagPを利用するシステム管理者およびユーザを「ユーザ」と呼ぶことにする。

への報告方法が知識ライブラリ記述言語によって記述されている。システム管理者は、知識ライブラリに新たなクラス定義を追加することによって magP の動作を自由に拡張することができる。また、知識ライブラリの主な特徴は、構造の階層化によって、再利用性が向上し、システム依存部が分離されていることである。magP の実行系は、知識ライブラリ内の定義に従って、システム管理に必要な情報の収集および分析を行う (図 3.2)。実際には、各クラス定義から生成されたインスタンスが互いに通信を行いながら情報の収集・分析作業を行っている。

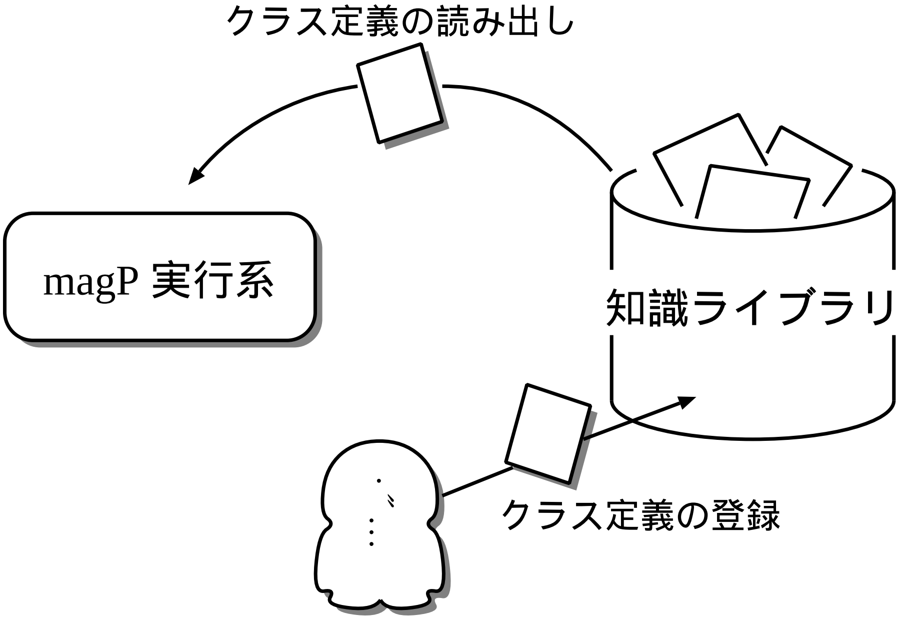


図 3.1 実行系と知識ライブラリ

magP は既存の多くの管理ツールを利用し、さまざまな情報を収集することができる。利用できる情報収集の手段や情報源は、外部コマンド、外部ファイル、SNMP、SYSLOG[3]などである。外部コマンドは、UNIX のシステム管理者が頻繁に利用する `ping`、`tcpdump`、`netstat` などのようなコマンドである。また、各種サービスのログ (動作記録) のようなファイルも情報源として活用できる。その他、SNMP を利用することによって、SNMP をサポートしているベンダーの製品ならばすべて調査可能になる。

UNIX では、SYSLOG を利用して各種アプリケーションやサービスの稼働状況を記録することが一般的であるため、SYSLOG の記録する情報を情報源とすることは、システムの監視に必要不可欠である。このように、本来、まったく異なるインタフェースを持った情報収集手段を統合することができるため、これまでに蓄積されてきた多くの強力な管理ツールを効果的に利用することができる。

---

[3] UNIX で標準的に用いられる動作ログ記録システム

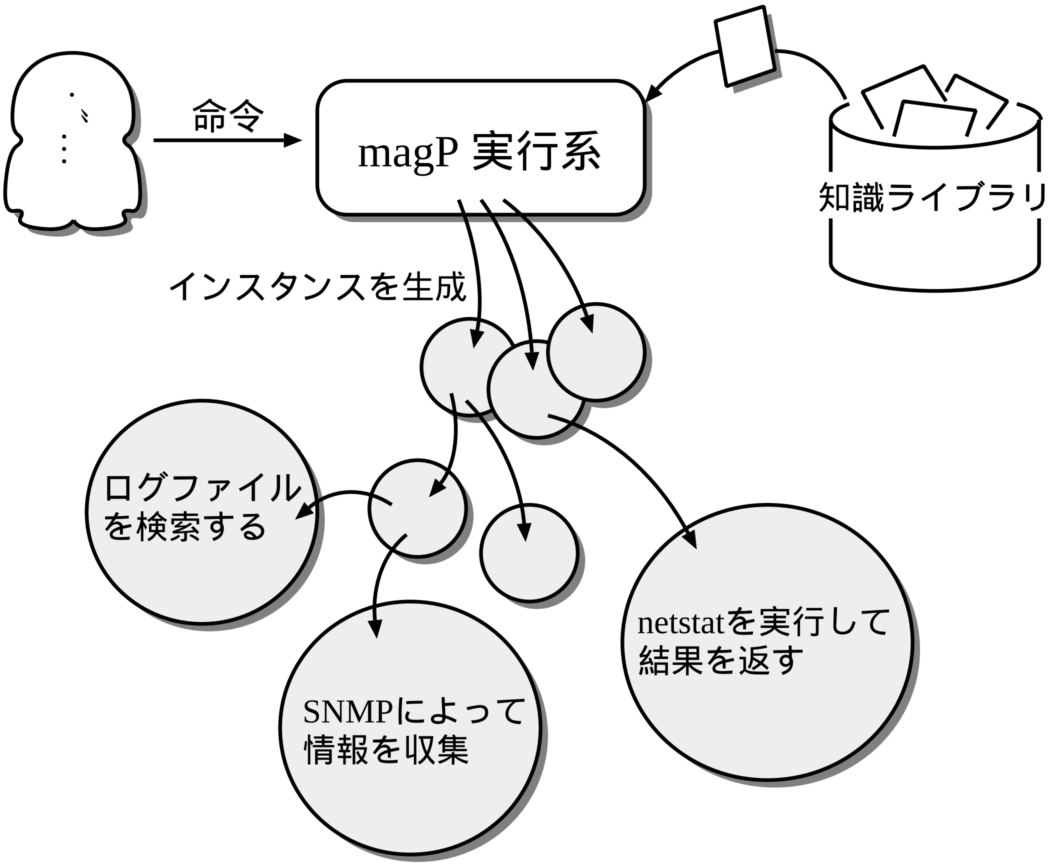


図 3.2 知識ライブラリを用いて情報の収集と分析を行う

なお、magP は開発プラットフォームとなるオペレーティングシステムとして主に UNIX を利用している。しかし、システムの大部分は高いポータビリティを持つ Java 言語 [3] によって開発され、知識ライブラリは 3.8節で述べる知識ライブラリ記述言語によって記述されるため、さまざまなオペレーティングシステムで利用可能である。

3.2　magP のシステム構成

magP は構成要素は次の通りである (図 3.3)。

- 知識ライブラリ
- ライブラリコンパイラ
- 実行系
- データベース
- ユーザインタフェース

- 報告機構

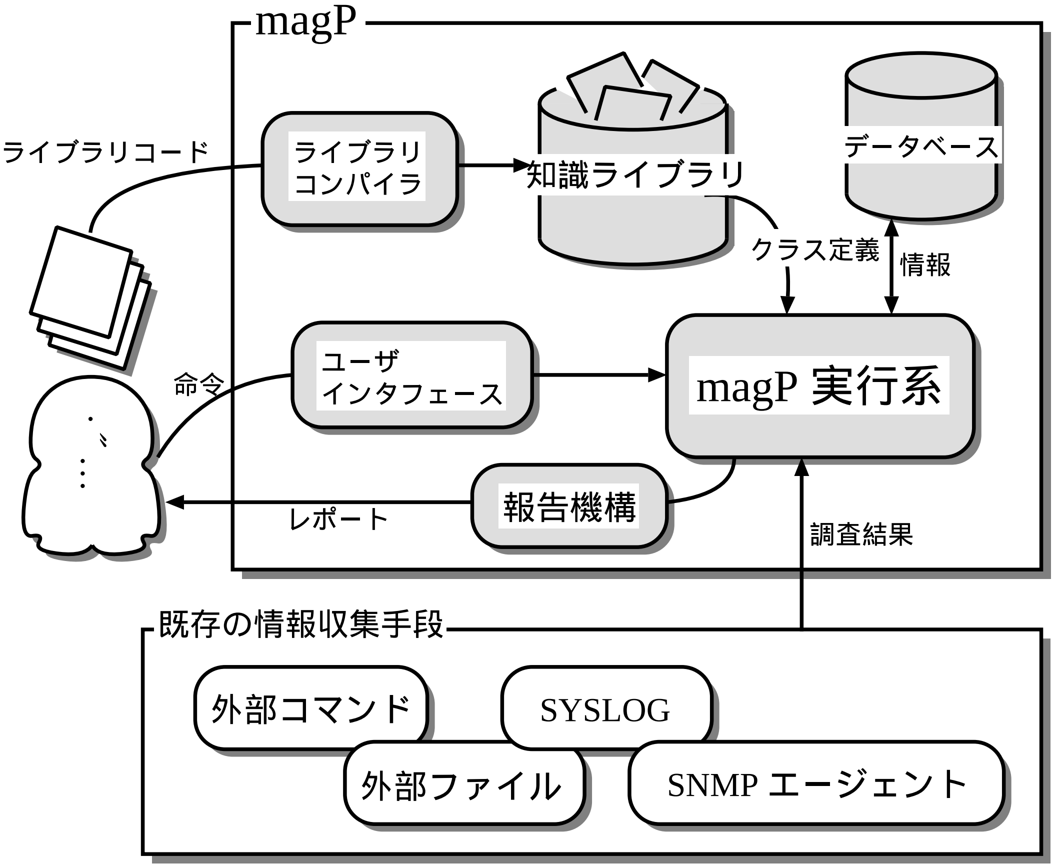


図 3.3 magP のシステム構成

magP のユーザは知識ライブラリ記述言語によって、外部コマンドの実行方法や実行結果の分析方法をクラス定義として記述する。新しいクラス定義を記述する際には、すでに知識ライブラリ内に存在するクラスを再利用することもできる。記述されたクラス定義は、ライブラリコンパイラによってコンパイルされ知識ライブラリに登録される。実行系はインタプリタであり、知識ライブラリ内のクラス定義に基づいてインスタンスを生成し動作する。その処理の過程で、情報を記録し管理するためにデータベースが利用される。ユーザインタフェースは、ユーザが実行系に対して、新たな処理の開始、停止、状況の報告などを依頼するために使用される。分析結果のユーザへの報告には報告機構が利用される。報告の手段は、電子メール、ページャ、FAX などを利用できる。実際には、データベースや報告機構も知識ライブラリの一部である。また、報告機構の報告先にユーザインタフェースを指定することによって、端末の画面上でシステムの監視作業を行うこともできる (3.8節)。

3.3 知識ライブラリ

知識ライブラリは、システム管理に必要な情報の収集方法、分析方法、およびシステム管理者への報告方法の記述の集合であり、クラス定義を単位として構成され、このライブラリはシステム管理者の知識に相当する。

クラスは、情報の収集を行うクラスと、情報の分析を行うクラスに大別される。

例えば、情報の収集を行うクラスには、`ping`、`traceroute`、`tcpdump` といった外部コマンドの実行方法、実行結果の分析方法が記述される。また、各種サービスの稼働状況はログファイルに記録されることが一般的であるが、そのようなファイルも情報源として利用することができる。さらに、現在、広く利用されている SNMP を利用して情報を収集することもできるため、ミクロな情報からマクロな情報までより多くの情報収集手段を利用することができる。

分析を行うクラスは、情報収集を行うクラスを利用して情報を収集し、分析条件が満たされ次第、ユーザに対して報告を行う。そのために、情報収集を行うクラスのインスタンスを生成し、それらと通信を行いながら処理を行う。

一般に、情報収集の手段はシステム依存度が高く、オペレーティングシステムやネットワークによって異なるインタフェースを持っている。そこで、あらかじめ情報収集の手段に関する定義を知識ライブラリ内に用意し、以後は、これを通して情報にアクセスすることにする。これによって、クラス定義を利用すれば、システムに依存しない方法で情報にアクセスすることができ、情報収集の方法に関する知識を要求されることなく、ユーザは分析規則の記述のみに専念することができる。

3.4 実行系 (`magp`)

magP の実行系 (`magp`) は次のもので構成される。

- インタプリタ
- スケジュールテーブル
- インスタンス管理テーブル
- アクセス制御管理テーブル

スケジュールテーブルには、ユーザが magP に行わせたい作業とその実行条件が登録されている。例えば、作業は「外部との通信状態を監視する」「異常なプロセスを検出する」「不正な侵入を検知する」などである。実行条件は、特定の時刻に実行、一定の間隔で実行など、時間による指定の他に、必ず 1 つはその作業に相当するクラスのインスタンスを動作させておくなどの指定も可能である。実行系は、このスケジュールテーブルに従って、知識ライブラリから必要なクラス定義を読みだし、インスタンスを生成する (図 3.4)。

スケジュールテーブルに従って生成されたインスタンスはそれぞれ専用のスレッドが割り当てられ動作する。図 3.5にこの過程を簡単に示す。スケジュールに従って生成されたイ

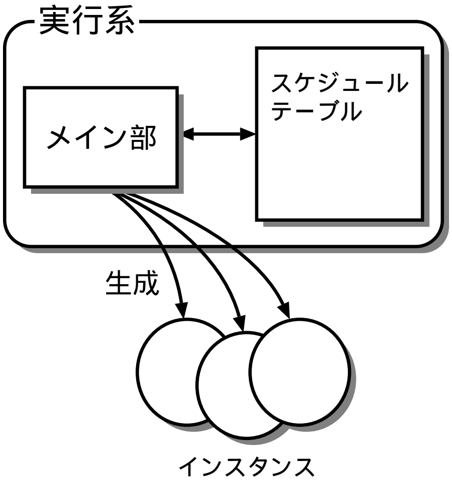


図 3.4 実行系はスケジュールテーブルに従ってインスタンスを生成

ンスタンスは、さらに細かな情報収集作業を行うクラスのインスタンス B,C,D を生成する。インスタンス A は、生成したインスタンス B,C,D から情報を受け取って分析を行う。インスタンス A は、分析結果に従って、次の作業に相当するインスタンスの生成、ユーザへの報告、あるいは他のホスト上で動作する magP へメッセージの送信を行う。また、magP は複数のスレッド上で処理を行うため、同時に複数の調査分析作業を行うことができる。また、インスタンス管理テーブルは、動作中のインスタンスを管理するために用いられる。

実行系はユーザからの要求に従って動作する。次節で詳しく述べるが、この要求は、例えば、スケジュールの登録・変更・削除、インスタンスの生成・停止・削除などである。magP は重要な管理情報を処理するシステムであるため、実行系への要求は慎重に取り扱われなければならない。そこで、magP へのアクセス権を制御するためのアクセス制御テーブルを持っている。このアクセス制御テーブルに従って、不正なユーザからの要求は排除している。

また、実行系はリモートシステム上にインスタンスを生成し、情報の収集・分析を行うことができる (図 3.6) ため、ローカルとリモートの区別をほとんど意識せずに、知識を記述することができる。

3.5 データベース

実行系では、処理の過程で収集された情報は必要であればデータベースに登録され、将来の分析作業に利用することができる。この部分は現在の magP では開発中である。データベースについては 6.4節で改めて述べる。

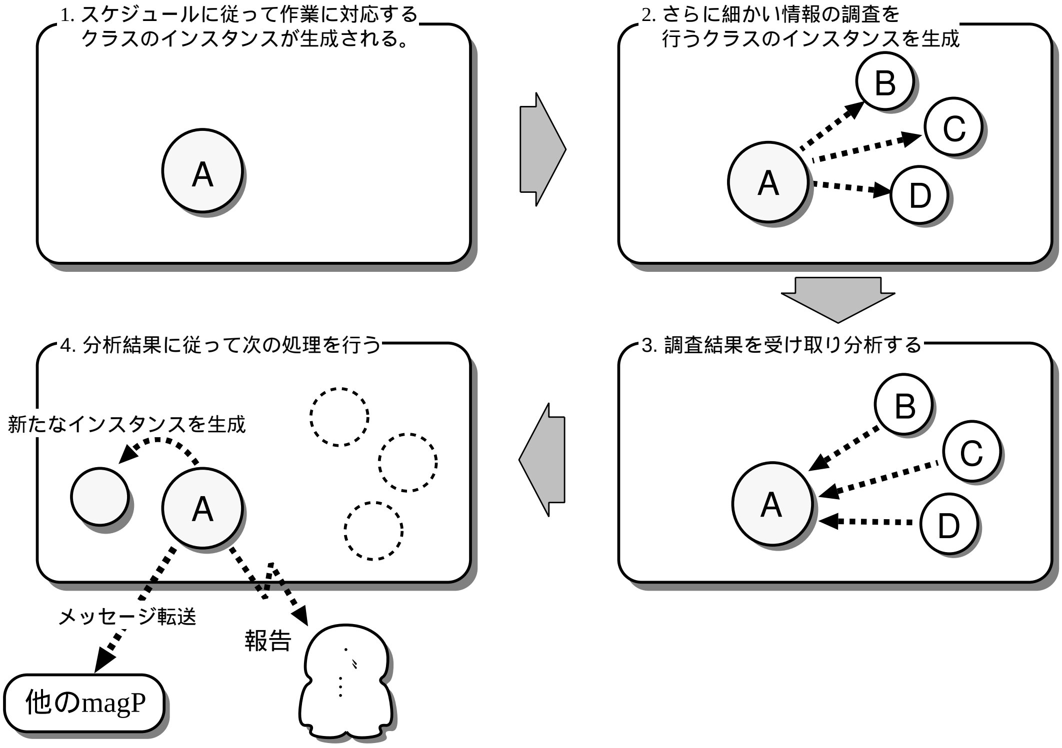


図 3.5 インスタンスの相互通信によって情報の収集・分析作業が進む

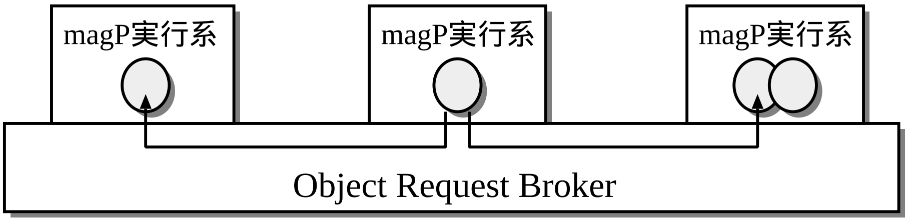


図 3.6 リモートシステム上にインスタンスを生成する

3.6 ユーザインタフェース (`magpcon`)

magP の動作は専用のグラフィカルユーザインタフェース (`magpcon`) によって制御する。ユーザは、この `magpcon` を用いて実行系に次のような要求を送ることができる。

- magP の動作状況の調査
- インスタンスの生成・停止
- スケジュールの登録・変更・削除

- magP の停止

図 3.7が、このユーザインタフェースのトップウィンドウであり通常はこの状態である。なんらかの操作を行いたい場合には、「Control Panel」ボタンを押すことによってメニューウィンドウを開く。図 3.8は作業を行っている様子である。

図 3.7 ユーザインタフェース (`magpcon`) のトップウィンドウ

図 3.8では、実行系によって生成され現在動作中のインスタンスの一覧を `magpcon` によって表示させている。また、この一覧内の名前をクリックすることによって、各インスタンスの動作を停止させることができる。

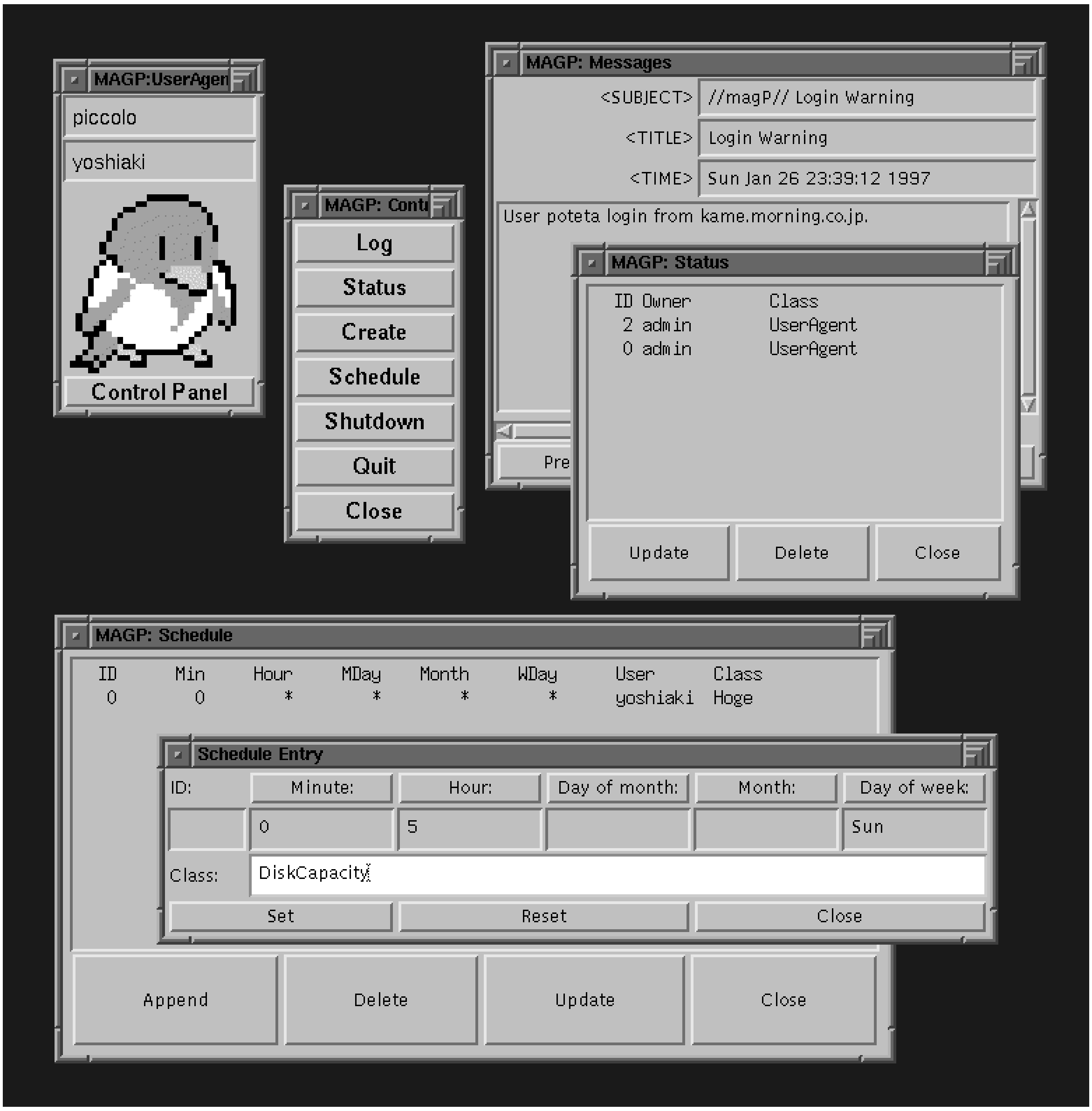
magP の実行系が持つスケジュールテーブルに対してエントリの登録・変更・削除を行うことができる。例えば、図 3.9 では、`magpcon` によって、1 時間ごとに `DiskCapacity` クラスのインスタンスを生成するスケジュールを登録している。`DiskCapacity` クラスはディスクの使用率を監視するクラスである。詳しくは 4.1節で説明する。

また、`magpcon` を作業結果の報告先として指定し、画面上で監視作業を行うこともできる。報告機構については 3.7節および 4章で説明する。

3.7 報告機構

magP では分析結果をユーザへ通知するために、電子メール、ユーザインタフェース、ページャ、FAX を用いる。

電子メールやグラフィカルユーザインタフェースによって障害発生などの事実を通知する仕組みはごく一般的であり、多くの管理アプリケーションで採用されている。また、ページャや FAX を報告手段として利用する方法は、システム管理者が計算機を利用できない状況にあっても、障害の発生やシステムへの侵入を検知することができるため、分析結果を短時間でシステム管理者へ通知する手段として非常に有用である。



piccolo
yoshiaki

<SUBJECT> //magP// Login Warning
<TITLE> Login Warning
<TIME> Sun Jan 26 23:39:12 1997

User poteta login from kame.morning.co.jp.

```
ID Owner        Class
 2 admin        UserAgent
 0 admin        UserAgent
```

```
ID    Min    Hour    MDay    Month    WDay    User       Class
 0     0      *       *       *        *      yoshiaki   Hoge
```

ID:
Minute: 0
Hour: 5
Day of month:
Month:
Day of week: Sun
Class: DiskCapacity

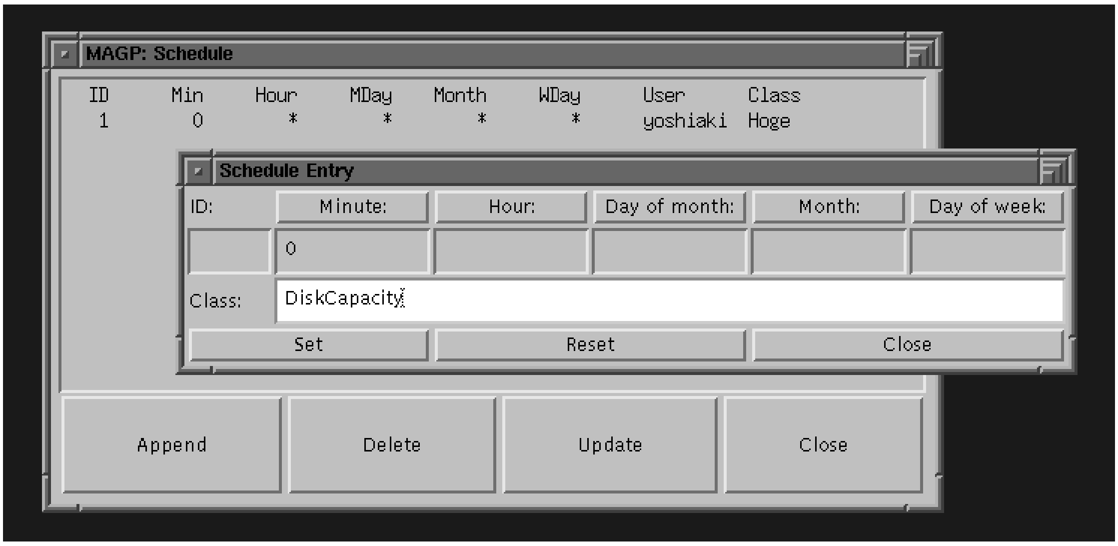
図 3.8 ユーザインタフェース (`magpcon`)

しかし、報告手段によって一度に伝えることができる情報量に大きな差があること、システム管理者の所在や状態によって最適な手段は変わることを考慮する必要がある。

magP の報告機構では、ユーザは報告内容の登録および報告手段の選択を行うだけで良い。報告機構は登録された情報から報告手段に最適な報告書を自動生成しユーザに通知する。そのため、ユーザは報告手段の違いを意識せずに利用することができる。

報告機構は次のような流れで使用される (図 3.10)。

1. Report クラスのインスタンスを生成する。



ID	Min	Hour	MDay	Month	WDay	User	Class
1	0	*	*	*	*	yoshiaki	Hoge

ID:	Minute:	Hour:	Day of month:	Month:	Day of week:
	0				

Class: DiskCapacity

図 3.9 magpcon によるスケジュールの登録

2. 報告事項を登録する。
3. 報告手段を指定する。
4. 報告を実行する。

このようにして報告された電子メールによる報告の例が、図 3.12である。また、ユーザインタフェースを報告先として指定することによって、端末の画面上で報告を確認することもできる。ユーザインタフェースは、報告書が到着すると、トップウィンドウのキャラクタをアニメーションさせることによってシステム管理者に知らせる。届いた報告書は、トップウィンドウのキャラクタをクリックすることによって眺めることができる (図 3.11)。

現在、以上のような magP の報告機構は開発中である。これについて、詳しくは 6.5節で述べる。

3.8 知識ライブラリ記述言語

あらかじめ用意されている知識ライブラリでは作業に不十分である場合、「知識ライブラリ記述言語」によって新たなクラス定義を記述し、知識ライブラリを自由に拡張することができる。また、動作中の magP の知識ライブラリに対してクラス定義の追加・削除を行うこともできる。

この節ではこの知識ライブラリ記述言語によって達成すべき目標と設計方針、記述例について説明する。

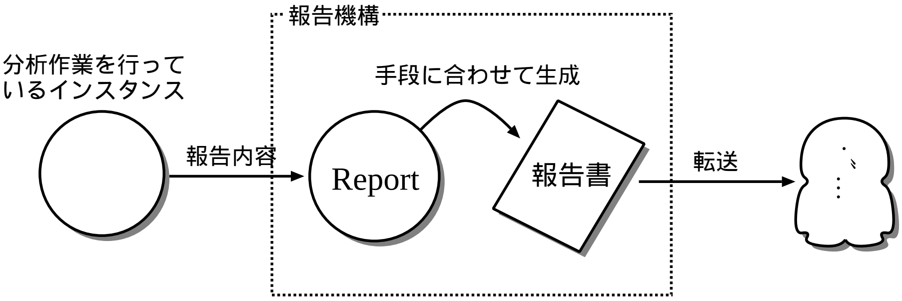
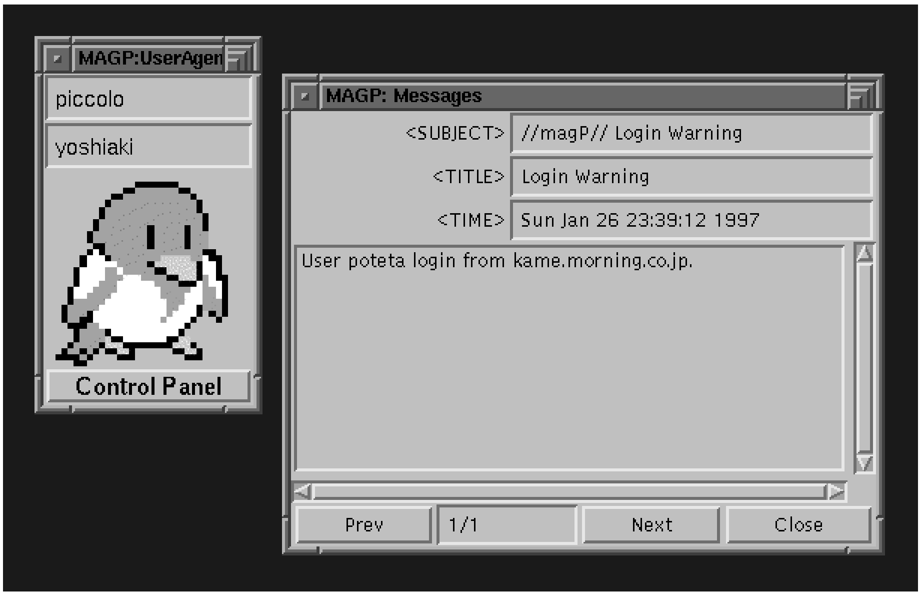


図 3.10 報告手段に合わせた報告書を生成し転送



piccolo

yoshiaki

<SUBJECT> //magP// Login Warning

<TITLE> Login Warning

<TIME> Sun Jan 26 23:39:12 1997

User poteta login from kame.morning.co.jp.

1/1

図 3.11 GUI による報告書の表示

3.8.1 目標

知識ライブラリ記述言語を設計するにあたって、特に次の点を考慮した。

- 情報収集手段の充実
  クラス定義中で、外部コマンド、ログ、SNMP などさまざまな情報収集の手段を利用可能にし、システム管理に必要なほとんどの情報を収集可能にする。
- 高い再利用性
  新たなクラス定義を記述する際には、既存のクラス定義の再利用できるようにし、開発にかかるコストを大幅に減少させる。

```
  10  97/01/23 17:39 yoshiaki@magp.co. 次回のセミナーについて
█ 11  97/01/23 17:46 magp@magp.co.jp   //magP// Login Warning

-かんなJ.:--%%-Mule: +inbox         5:54pm 0.76   (Summary)--L11--[0 more]------
Subject: //magP// Login Warning
From: magP <magp@magp.co.jp>
To: yoshiaki@magp.co.jp
Date: Thu, 23 Jan 1997 17:46:52 +0900
X-Dispatcher: impost version 0.96++ (Jan. 13, 1997)
Lines: 5

<TITLE> Login Warning
<TIME>  Thu Jan 23 17:46:50 1997
----
User poteta login from kame.morning.co.jp.







-かんなJ_:--%%-Mule: *Mew message*   5:54pm 0.76   (Message)--L13--Bot---------
```

図 3.12 電子メールによる報告の例

- 優れたポータビリティ
  あらゆるオペレーティングシステムで利用可能なクラス定義を記述可能にし、複数のシステム間で知識ライブラリを共有することができるようにする。
- 分散管理
  複数のシステムを同時に管理するようなクラス定義の記述を可能にし、分散システムの管理を効率良く行えるようにする。

3.8.2　設計

3.8.1節で述べた目標を達成するために、次のような方針で知識ライブラリ記述言語の設計を行った。

- オブジェクト指向の技法を利用する。
- 情報の収集手段および分析方法はオブジェクトである。
- オブジェクト間の通信によって処理を行う。
- システムに依存する部分をオブジェクトで隠蔽する。
- オブジェクトの継承機能を利用して再利用性を高めている。
- 分散環境下においてもオブジェクトを扱えるようにする。
- システム管理に特化したデータ型を提供する。

知識ライブラリ記述言語では、オブジェクト指向の持つ概念のうち、特に情報隠蔽および継承機構を利用している。

ユーザは、新たなクラス定義を記述する際に、既存のクラス定義を継承し再利用することができる。

システムによって収集方法が異なるような情報が存在する場合、まず、その情報を収集するためのクラスを個々のシステムに合わせて用意する。この時、各クラスは外部に対して共通のインタフェースを提供すれば、目的の情報はどのようなシステムおいても共通のインタフェースで収集可能になる。このように、システムに依存する部分を情報収集を行う最下層のクラスに隠蔽できるため、さまざまなシステムで利用可能なクラス定義の記述が可能になりポータビリティが向上する。

より具体的で細かな情報を収集するクラスが存在する場合、それらと通信を行いながら分析作業を行い情報の抽象度を高めるクラスを用意する。これによって、図3.13のように、低レベルの情報収集を行うクラスを利用して、高位のクラスはより抽象度の高いインタフェースをユーザに提供できるようになる。

また、オブジェクト指向の技法を用いることによって、モジュール化が進み既存のクラスを組み合わせて新たなクラス定義を記述することができるため開発コストが減少する。

magP では表3.1のように情報の収集手段を4つに分類する。

タイプ	内容
外部コマンド型	外部のコマンドを利用して情報を収集
外部ファイル型	外部のファイルから情報を収集
SYSLOG 型	SYSLOG に提供される情報を利用
SNMP 型	SNMP によって情報を収集

表 3.1 magP の情報収集の手段を4つに分類

外部コマンド型は、`ping`、`traceroute`、`netstat` のような管理作業で広く用いられているコマンドである。外部ファイル型は、例えば各種サービスのログファイルや設定ファイルであり、ファイルであればすべて情報源として利用できる。同様に、SYSLOG は報告されるログをファイルとして記録するが、UNIX では重要な管理情報を SYSLOG を利用して記録することが多く利用価値が高いため SYSLOG 型として別のタイプに分類した。SNMP 型は、管理対象から情報を収集する手段として非常に一般的な SNMP を用いて多くの機器から情報を取得できる。

このように情報を収集するために多くの手段を利用でき、本来異なるインタフェースを持ったこれらの手段に対して、知識ライブラリ記述言語によって共通のインタフェースを与えることができる。

分散システムをより効果的に管理するクラス定義を記述するために、知識ライブラリ記述言語では、リモートシステム上でのインスタンスの生成、およびメソッドの呼び出しを可能にしている。これによって、ローカルシステム上での情報の収集・分析作業とほぼ同様にして作業内容を記述することができる。

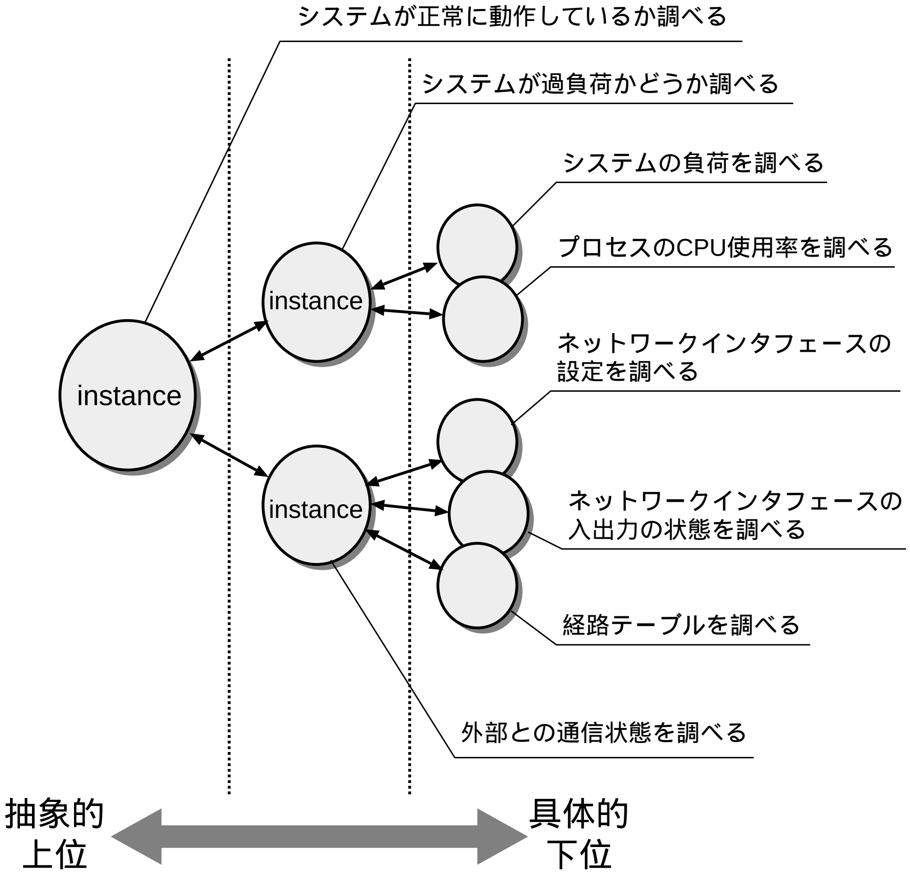


図 3.13 上位のオブジェクトと下位のオブジェクト

また、一般のプログラミング言語の持つデータ型に加え、システム管理に特有のデータ型を提供し、効率良く分析処理を記述できるようにしている。

3.8.3 記述例

ユーザは、次のような手順で知識ライブラリにクラス定義を追加する。

1. 必要なメソッドの定義を記述する。
2. 情報収集が必要な場合は必要な情報手段のタイプを指定し、情報収集に必要な定義も記述する。
3. ライブラリコンパイラ (`magpc`) で中間コードへ変換する。
4. 知識ライブラリに登録する。

クラス定義は次のものを含む。

- クラス外へのインタフェースとしてのメソッド定義

- 情報収集の方法に関する定義

クラス自身が外部の情報収集手段によって情報を収集しない場合、情報収集の方法に関する定義を記述する必要はない。ただし、その場合、情報を収集するために他のクラスを利用することになる。また、magP の実行系はクラスのインスタンスを生成すると、はじめに生成したインスタンスの `run()` メソッドを呼び出す。そのため、実行系から呼び出される可能性のあるクラスは、必ず `run()` メソッドが定義されていなければならない。

```
 1: class Uptime {
 2:
 3:   info uptime {
 4:     type = ExternalCommand;         // 外部コマンド型を指定
 5:     comm = "uptime";                // コマンド名は uptime
 6:     form = (time:str state:ident users:int tmp:",.*d" nul load:float);
 7:   }
 8:
 9:   method boolean isheavy() {
10:     return (get_load() >= 2.0);     // 負荷の値が 2.0 以上であれば true
11:   }
12:
13:   method void run() {
14:
15:     init();                         // 初期化
16:     if (isheavy()) {                // 負荷が大きければ...
17:       Report rep = new Report();
18:       rep.report_to = "yoshiaki@magp.co.jp";
19:       rep.writeln(system.hostname() +
20:                   ": load average is " +
21:                   get_load() +
22:                   ", too high! ");
23:       rep.sendMail();               // 電子メールによって通知する
24:     }
25:   }
26:
27: }
```

図 3.14 システムの負荷を監視するクラスの定義

ここでは、「システムの負荷を監視するクラス」を例にクラスの定義方法を簡単に説明する。図 3.14が定義例である。1 行目のキーワード`class` に続いてクラス名を記述し、続くブロック内に定義内容を記述する。この例では、情報収集の手段として外部コマンド型を用いている。使用する外部コマンドは`uptime` である。3 行目から 7 行目が、情報の収集手段に関する定義である。この部分を info 宣言部と呼び基本的な構造は図 3.15である。図 3.15

中の「名前」は情報収集の手段の名前でありクラス内で一意なものを指定する。

```
info 名前 {
  キーワード = 値 ;
  キーワード = 値 ;
         :
}
```

図 3.15 info 宣言部の構造

キーワードは収集手段のタイプによってさまざまなものが用意されており、その値によって情報の収集方法を定義する。各キーワードの意味には表 A.1の通りである。この例では、4 行目で外部コマンド型 (`ExternalCommand`) を指定し、5 行目でコマンド名 (`"uptime"`)、続く 6 行目でコマンドの実行結果のフォーマットを定義している。`uptime` コマンドをシェル上で実行した結果は図 3.16である。magP は実行結果を行ごとに解析するため、info 宣言部の`form` の値は、この実行結果の 1 行分のフォーマットに対応している。`form` の値は次のよう形式で記述する。

（ 項目名:型名 項目名:型名 ... 項目名:型名 ）

1 行分の結果を解析する際の基本的な流れは次の通りである。

1. 行が区切り文字によって単語に区切られているとみなし、単語を 1 つ分取り出す。
2. 適当な型に変換し、指定された項目名で参照できるようにする。この時、用いる項目名と型とは`form` で指定される。
3. `form` の値に含まれる要素の個数だけ以上を繰り返す。

以上の手続きを実行結果の行数分繰り返すことによって、最終的に 1 つの項目名に行の数と同じ個数だけ値が対応付けられることになる。

また、区切り文字の初期値は空白文字[4] であり、型名は図 3.2の通りである。さらに、実行結果のフォーマット内では「項目名:型名」の代わりに、図 3.3ような特別な記述も許す。

この例では、図 3.16の`uptime` の実行結果を次のようにして解析している。

1. 空白文字で単語に区切る。
2. 1 番目の単語を文字列型とし項目名`time` に対応付ける。
3. 2 番目の単語を文字列型で項目名`state` に対応付ける。
4. 3 番目の単語を整数型で項目名`user` に対応付ける。

---

[4] この場合の空白文字は、スペース・タブ・改行・復帰文字。

型名	内容
`int`	整数値
`float`	浮動小数値
`str`	文字列 (『」で囲まれたもの)
`ident`	文字列
`IPaddr`	IP アドレス
`time`	時刻
`host`	ホスト名
`email`	電子メールアドレス
`left`	行末までの残り文字すべて
`".*"`	正規表現

表 3.2 実行結果フォーマット内で指定可能な型名

`nul`	任意の単語にマッチするが無視する
`"文字列"`	それ以後の区切り文字を変更する

表 3.3 実行結果フォーマット内での特別な記述

5. 4 番目の単語 (`days,`) から 8 番目の単語 (`load`) までを文字列型で項目名`tmp` に対応付ける (正規表現で単語を指定する)。
6. 次の単語 (`avarages:`) は`nul` で無視する。
7. 次の単語を浮動小数型で項目名`load` に対応付ける。

このようにして、図 3.16のような実行結果が行ごとに解析され、各項目の値が 5 つの項目名 (`time`, `state`, `users`, `tmp`, `load`) で参照できるようになる。図 3.17は図 3.16の実行結果の解析方法を図で表したものである。

9 行目から 25 行目までがメソッドの定義である。この部分を method 宣言部と呼ぶ。この例では 2 つのメソッドが定義されている。

1 つ目のメソッド `isheavy()` は、システムの負荷が高いかどうか調査するメソッドである。具体的には、`uptime` コマンドによって得られた負荷の値が 2.0 以上である場合に真偽値`true`、そうでない場合に真偽値`false` を返す。負荷の値は項目名`load` に対応付けられており、`get_load()` という手続きによって得ることができる。また、`get_load(`整数値`)` と記述することによって、指定した行の値を取り出すことができる。今回の例のように整数値を省略した場合は最後に解析を行った行 (最終行) の値が得られる。このように、各項目に対応付けられた値を得るためには、次のように記述する。

```
get_項目名 (index)
```

`index` の値は 0 以上の整数値であり、0 がはじめの行に対応する。また、次のように記述すると、最終行の値が得られる。

`get_項目名 ()`

この他にも項目名に対応付けられている値にアクセスするためのさまざま手続きが用意されている。表3.4内の手続きは、`str` 型 ・`ident` 型 ・`host` 型 ・`email` 型 ・`left` 型で利用できる。一方、表3.5) は、`int` 型 ・`float` 型で利用することができる手続きである。

型名	内容
get_項目名 (s)	項目内の値のうち文字列 s を含むものを 1 つ得る
get_all_項目名 (s)	項目内の値のうち文字列 s を含むものをすべて得る
get_index_項目名 (s)	項目内の値のうち文字列 s を含むもの 1 つ分のインデックス得る
get_index_all_項目名 (s)	項目内の値のうち文字列 s を含むものをすべてのインデックス得る

表 3.4 各項目名の値へのアクセスに利用できる手続き (文字列の場合)

型名	内容
get_max_項目名 ()	最大値を得る
get_min_項目名 ()	最大値を得る
get_max_項目名 (n)	値が大きい方から n 個まで得る
get_min_項目名 (n)	値が小さい方から n 個まで得る
get_index_max_項目名 ()	最大値のインデックスを得る
get_index_min_項目名 ()	最小値のインデックスを得る
get_index_max_項目名 (n)	値が大きい方から n 個分のインデックスを得る
get_index_min_項目名 (n)	値が小さい方から n 個分のインデックスを得る
get_gt_項目名 (x)	値が x よりも大きい要素をすべて得る
get_gte_項目名 (x)	値が x 以上の要素をすべて得る
get_lt_項目名 (x)	値が x よりも小さい要素をすべて得る
get_lte_項目名 (x)	値が x よりも小さい要素をすべて得る
get_index_gt_項目名 (x)	値が x よりも大きい要素のインデックスをすべて得る
get_index_gte_項目名 (x)	値が x 以上の要素のインデックスをすべて得る
get_index_lt_項目名 (x)	値が x よりも小さい要素のインデックスをすべて得る
get_index_lte_項目名 (x)	値が x よりも小さい要素のインデックスをすべて得る
get_avarage_項目名 ()	項目内の平均値を得る
get_variance_項目名 ()	項目内の分散を得る
get_deviation_項目名 ()	項目内の標準偏差を得る

表 3.5 各項目名の値へのアクセスに利用できる手続き (数値の場合)

2 つ目のメソッドの定義は `run()` メソッドである。`Uptime` クラスは magP の実行系によってインスタンスの生成および実行が行われるため、`run()` メソッドが定義されている必要がある。この `run()` メソッドでは、`Uptime` クラスのインスタンスを初期化し (15 行

目)、`isheavy()` メソッドによって負荷を調べている (16 行目)。この時、負荷が高い場合には、電子メールによって報告書の通知を行っている (18 行目から 23 行目)。

17 行目では、報告用クラスのインスタンスを生成し、18 行目から 22 行目で宛先および報告内容を登録している。23 行目では、手段に電子メールを指定して実際に報告を行っている。

知識ライブラリ記述言語の仕様については付録 A に記した。

```
% uptime
12:07PM  up 10 days,  6:10, 3 users, load averages: 0.05, 0.10, 0.08
%
```

図 3.16 uptime コマンドの実行例

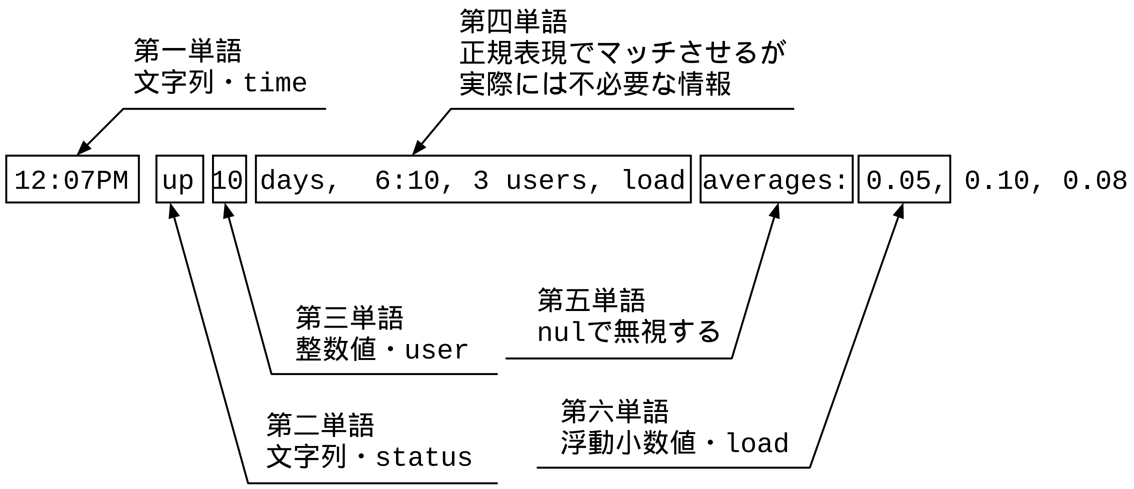


図 3.17 uptime コマンドの実行結果を解析する

3.9 実装に利用したオペレーティングシステムとツール

現在、magP は次のオペレーティングシステム上での動作が確認されている。

- Solaris 2.5.1 (sparc, x86 edition)
- FreeBSD 2.1.0R, FreeBSD 2.1.5R, FreeBSD 2.1.6R
- BSD/OS 2.1

ただし、BSD/OS では今のところリモートホスト上にインスタンスを生成しメソッドを呼び出すことができない。

また、magP の実行系・知識ライブラリ・報告機構・ユーザインタフェースは、Java 言語 [17](Java(tm) Developers Kit version 1.0.2)[5] および、HORB 言語 (HORB version 1.3.b1)[6]によって実装されている。ただし、一部のコードは perl 言語 (perl version 5.002) [7] によって記述されている[8] が、将来的には Java 言語によって再実装される予定である。また、ライブラリコンパイラの実装には、C 言語 (GNU C コンパイラ version 2.6.3)、bison(GNU Bison version 1.25)、flex(GNU flex version 2.5.3) を用いている。

---

[5] `http://java.sun.com/products/JDK/1.0.2/index.html`

[6] `http://ring.etl.go.jp/openlab/horb/`

[7] `http://www.perl.com/perl/index.html`

[8] これは Java インタプリタの動作がシステムによって異なることを回避するために行っている。おそらく Java インタプリタのバグである。

第 4 章

magP の実行例とその考察

この章では magP の実行例を示す。例として、次の 4 つについて述べる。

- ファイルシステムの使用率の監視 (4.1節)
- 複数計算機の監視 (4.2節)
- 計算機の負荷および仮想メモリの使用状況の監視 (4.3節)
- 組織外からのログインの監視 (4.4節)

4.1　ファイルシステムの使用率の監視

この例では、システムのファイルシステムの使用率の監視を行う。

4.1.1　実行例の説明

UNIX ではファイルシステムの使用率を調査するために`df` コマンドを用いる。`df` コマンドの実行結果の例は図 4.1である。

```
% df -t ufs
Filesystem 1K-blocks   Used  Avail Capacity Mounted on
/dev/sd0a      19471  11313   7184    61%   /
/dev/sd0g     297759 263365  19506    93%   /usr
/dev/sd0e      95311   9668  80877    11%   /var
/dev/sd0h     414591 374227  19634    95%   /export/src.i386-bsdos
/dev/sd1h     934315 661110 226489    74%   /export/local.i386-bsdos
/dev/sd1g      98911  79935  14030    85%   /export/infosys
%
```

図 4.1 `df` コマンドの実行結果

この出力は、6 つのファイルシステムの状態を示している。出力の左から 5 つ目の項目が各ファイルシステムの使用率を表している。このコマンドを用いて、ファイルシステムの状

態を調査する `FileSystem` クラスの定義例が図 4.2である。この定義例には、info 宣言部のみで method 宣言部はない。4 行目で情報の収集手段に外部コマンド型 (`ExternalCommand`) を指定し、5 行目に外部コマンドの名前および引数を記述している。図 4.1からもわかる通り、`df` コマンドの実行結果は 1 行目に項目名が出力される。そこで、6 行目では実行結果の最初の 1 行分を無視することを指定している。7 行目は、外部コマンドがなんらかの原因で終了しない場合、15 秒間で強制終了することを意味している。8 行目では、実行結果のフォーマットを定義している。このフォーマットに従って外部コマンドの実行結果を解析し、項目名と値の対応付けが行われる。例えば、図 4.1の実行結果を解析した場合、表 4.1 のようにして値を参照することができる。

```
 1: class FileSystem {
 2:
 3:   info df {
 4:     type = ExternalCommand;     // 外部コマンド型
 5:     comm = "df -t ufs";         // df コマンド
 6:     skip = 1;                   // 実行結果の最初の 1 行分を無視
 7:     timeout = 15;               // 15 秒でタイムアウト
 8:     form = ( name:ident size:int used:int avail:int cap:int mnt:ident );
 9:   }
10:
11: }
```

図 4.2 ファイルシステムの状態を調査するクラス

記述例	得られる値	説明
`get_name(0)`	`"/dev/sd0a"`	項目 `name` の最初の値
`get_size()`	`98911`	項目 `size` の最後の値
`get_max_used()`	`661110`	項目 `used` の最大値
`get_index_max_avail()`	`4`	項目 `avail` の最大値のインデックス
`get_max_gte_cap(90)`	`95, 93`	項目 `cap` の値が 90 以上の要素
`get_avarage_cap()`	`69`	項目 `cap` の値の平均

表 4.1 各項目の値へのアクセス例

`FileSystem` クラスを用いて、ファイルシステムの使用率の監視を行なう `DiskCapacity` クラスの定義例が図 4.3である。

このクラス自身は外部からの情報収集を行なわず、他のクラスを利用して情報を収集するため、定義内に info 宣言部はない。`DiskCapacity` クラスは `run()` と `report()` の 2 つ

のメソッドを持つ。

`void run()`
magP の実行系によってインスタンスの生成 および実行が行なわれる際に呼び出される。メソッド `report()` を呼び出してる。

`boolean report()`
ファイルシステムの使用率を調査し 99%以上のファイルシステムが存在する場合に報告を行なう。また、使用率が 99%以上のファイルシステムが存在する場合は真偽値 `false` を返し、そうでない場合は `true` を返す。

`report()` メソッドついて簡単に説明する。9 行目で `FileSystem` クラスのインスタンスを生成し、10 行目で初期化を行なっている。`FileSystem` クラスでは、この初期化によって実際に外部コマンド (この場合は `df` コマンド) が実行され情報が収集される。12 行目から 15 行目では報告書の準備を行なっている。17 行目では、項目名`cap` に対応付けられている値のうち、99 以上のもののインデックスを得ている。これによって、使用率が 99%以上のファイルシステムのインデックスが得られる。20 行目から 23 行目まででは、17 行目で得られたインデックスをもとにして、ファイルシステム名とその使用率を報告書の本文に書き込んでいる。24 行目、25 行目では、実際に報告書の送信を行なっている。また、`report()` メソッドの返り値は、使用率が 99%以上のファイルシステムが存在した場合は`false`、存在しなかった場合は`true` である (26 行目、28 行目)。

4.1.2 実行例の考察

magP では、`FileSystem` クラスのような情報の収集方法を一度記述してしまえば、以後はそのクラスを利用してさまざまな分析条件の記述に利用できる。

例えば、図 4.3の 17 行目では、使用率が 99% 以上であるファイルシステムのリストを得ている。これと同様の作業を、仮にシェルスクリプトによって実現する場合、`df` コマンドの出力を処理するために、`awk` や `sed` といったテキスト処理のためのツールを利用したとしても非常に繁雑なものになる。また、`FileSystem` クラスはこの例のような使用率の監視以外の用途にも利用することが可能であり、このように利用しやすい一貫したインタフェースの提供は、シェルスクリプトでは実現が難しい。

この実行例の場合、単純に使用率が 99%以上であるファイルシステムを検出している。しかし、使用率の変化が著しく大きいファイルシステムの検出など、より複雑な条件の記述を行うことによって、さらに実用度を向上させることができる。その際、`FileSystem` クラスの収集する情報へアクセスする手段は、図 3.5のように豊富に揃っているため条件の記述は容易である。

```
class DiskCapacity {

  method void run() {
    report();
  }

  method boolean report() {

    FileSystem fs = new FileSystem();    // 情報収集のために
    fs.init();                           // FileSystem クラスを用いる

    Report r = new Report();                  // 報告書の生成
    r.report_to = "yoshiaki@magp.co.jp";      // 宛先の設定
    r.report_title = "Disk capacity report";  // 題名の設定
    r.println("Warning!!!");                  // 報告書の本文を書き込み

    int[] index = fs.get_index_gte_cap(99);   // 項目名 cap から値が 99 以上の
                                              // もののインデックスを得る
    if (index.length > 0) {
      for (int i = 0; i < index.length; ++i) {
        r.println(fs.get_name(index[i]) + "\t" +
                  fs.get_cap(index[i]) + "%");   // 報告書の本文の書き込み
      }
      r.sendMail();     // 報告書を電子メールとして送信
      r.sendMagpCon();  // 報告書を監視ブラウザへの送信
      return (false);
    }
    return (true);

  }

}
```

図 4.3 ファイルシステムの状態を調査するクラス

4.2 複数計算機の監視

この例では、4.1節で説明した`DiskCapacity`クラスを用いて複数の計算機のファイルシステムを監視する。

4.2.1 実行例の説明

図4.5が複数の計算機のファイルシステムの監視を行なう`MultiSystems`クラスの定義例である。

6行目の「`arguments`」は`String`クラスの配列であり、実行系はこの配列を通じて各クラスのインスタンスに引数を渡すことができる。`MultiSystems`クラスは、引数として与えられた配列を管理対象となるホストのリストとする (図4.5の6行目)。

9行目から12行目では、図4.4のように、`rnew` 演算子を用いて各システム上に`DiskCapacity`クラスのインスタンスを生成している。`rnew`演算子は`new`演算子と同様にクラスのコンストラクタを呼び出してインスタンスを生成するが、指定された計算機上にインスタンスを生成する。11行目では、配列`hosts`の要素を計算機名として、`DiskCapacity`クラスのインスタンスを各計算機に生成している。

20行目から26行目では、各計算機上のインスタンスの`report()`メソッドを呼び出し、ファイルシステムの使用率が正常であるかを調査している。それぞれの結果に従って、報告書の本文に結果を書き込み、29行目から30行目で報告書の送信を行なっている。

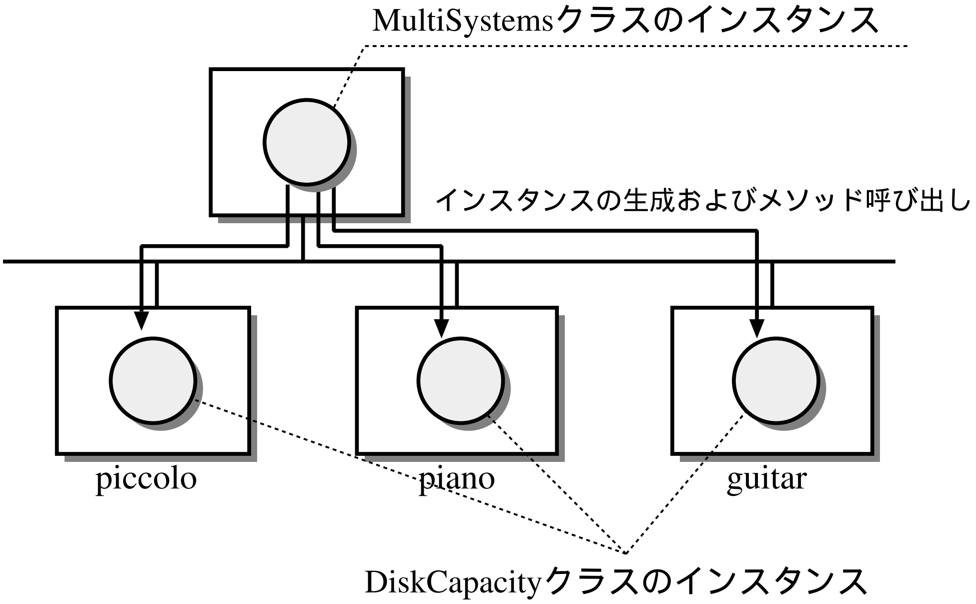


図4.4 リモートシステム上に`DiskCapacity`クラスのインスタンスを生成

```
class MultiSystems {

  method void run() {

    // 引数をもとに管理対象となるホストのリストを設定
    String[] hosts = arguments;

    // DiskCapacity クラスのインスタンスを各計算機上に生成
    DiskCapacity[] dc = new DiskCapacity[hosts.length];
    for (int i = 0; i < hosts.length; ++i) {
      dc[i] = rnew hosts[i] DiskCapacity();
    }

    // 報告書の準備
    Report r = new Report();
    r.report_to = "yoshiaki@magp.co.jp";
    r.report_title = "Multi Systems’ File System Report";

    // 各計算機を調査し結果を報告書に書き込む
    for (int i = 0; i < hosts[i].length; ++i) {
      if (dc[i].report()) {
        r.println(hosts[i] + " is good. (^_^)\n");
      } else {
        r.println(+ hosts[i] + " is bad. (;_;)\n");
      }
    }

    // 報告書を送信する
    r.sendMail();
    r.sendMagpCon();

  }

}
```

図 4.5 複数の計算機のファイルシステムを調査するクラス

4.2.2 実行例の考察

magP ではリモートシステム上にインスタンスを生成しメソッドの呼び出しを行なうことができる。この実行例を見ると、複数の計算機に関する情報の収集・分析作業をローカルシステムとリモートシステムの区別をほとんど意識せずに記述できることが分かる。ま

た、情報の収集だけでなく比較的時間のかかる分析作業は複数の計算機上に生成したインスタンスに処理させることによって、負荷を分散させることも可能である。

4.3 計算機の負荷および仮想メモリの使用状況の監視

この例では、計算機の負荷および仮想メモリの使用状況の監視を行う。

4.3.1 実行例の説明

この例の具体的な監視内容は次の通りである。

1. 計算機の負荷を調査する。負荷が高い場合は、CPU使用率が大きいプロセスをリストアップする。
2. 仮想メモリの使用状況を監視する。メモリ不足である場合、メモリ使用率の大きいプロセスをリストアップする。

これらの監視作業において、負荷が高いか、メモリ不足であることが判明した場合、適当な報告手段によって管理者へ通知を行う。図4.6および図4.7はこの監視作業を行なう`SysStat`クラスの定義例である。`SysStat`クラスは、計算機の負荷を調査するために`LoadAvarage`クラスを利用し、仮想メモリの状況を調査するために`VirtualMemory`クラスを利用する(図4.6・12~15行目)。

図4.6の18行目では、`LoadAvarage`クラスの`isHigh()`メソッドを用いて計算機の負荷を調査している。`isHigh()`メソッドは、呼び出されると負荷が異常に高い場合は真偽値`true`を、正常な場合は`false`を返す。

図4.7の35行目では、`VitualMemory`クラスの`isShort()`メソッドを用いて仮想メモリの状態を調査している。`isShort()`メソッドは仮想メモリが著しく不足している場合に`true`を、正常な場合に`false`を返す。

また、CPU使用率が高いプロセス、メモリ使用率が高いプロセスのリストを得るために、`ProcStat`クラスを利用している。`ProcStat`クラスの`getHeavyProcReport()`メソッドは、CPU使用率が高いプロセスのうちその上位から3つ分について、CPU使用率・所有者・プロセスID・コマンド名のレポートを作成し文字列配列として返す。`SysStat`クラスは、このレポートを報告書の本文に書き込んでいる(図4.6・23~31行目)。また、メモリ使用率の高いプロセスのリストについても、同様の処理を`ProcStat`クラスの`getBigProcReport()`メソッドを用いて行なっている(図4.7・40~48行目)。

また、`LoadAvarage`クラス、`VirtualMemory`クラス、`ProcStat`クラスの記述例は、それぞれ付録Bの図B.1、図B.2、図B.3である。

4.3.2 実行例の考察

`SysStat`クラスを記述する際には、計算機の負荷の調査方法や、具体的にどのような値である場合に計算機の負荷が高いとみなすかなどの知識を必要しない。これら知識は、

LoadAvarage クラス内に隠蔽されているためである。

同様に、仮想メモリに関する情報の収集方法やメモリ不足の判断規準などは VitualMemory クラス内に隠蔽されているため、やはり SysStat クラスを記述する際にこれらの情報は必要とされない。

また、LoadAvarage クラスや VirtualMemory クラスに用意されているメソッドを利用することによって、より具体的な値を用いた分析条件を記述することも可能である。

このように、管理ライブラリ記述言語では、ユーザのシステム管理に関する熟練度に合わせたインタフェースを提供できる。

4.4 組織外からのログインの監視

この例では、架空のドメイン「magp.co.jp」内において、組織外からのログインを監視する。

4.4.1 実行例の説明

組織外からのログインを検知するために、login というプログラムが SYSLOG を利用して記録するログを利用する。図 4.8は login プログラムが記録するメッセージの例である。このメッセージは、morning.co.jp というドメインの kame という名前のホストから、ユーザ名 poteta によってログインが行われたことを示している。この場合、ドメイン名が magp.co.jp ではないため、組織外からのログインである。

SYSLOG に報告される login プログラムのメッセージを収集する LoginSuccess クラスの記述例が図 4.9である。まず、4 行目では情報の収集方法として SYSLOG 型を指定する。次に、5 行目～6 行目では、SYSLOG を通して記録されるメッセージのうち、対象とするメッセージの条件を指定している。例えば、5 行目では login プログラムのメッセージであること、6 行目でメッセージが文字列「"login from"」を含んでいることを条件として指定している。7 行目は図 4.8に示されるメッセージのフォーマットを定めている。

図 4.10は、LoginSuccess クラスを用いて実際に組織外からのログインの監視を行う Login クラスの記述例である。このクラスについて簡単に説明する。

まず、9 行目では、SYSLOG から作業に必要な login プログラムのメッセージを得るために、LoginSuccess クラスのインスタンスを生成する。次に、10 行目で、init() メソッドによって、生成されたインスタンスを初期化しメッセージを待つ。init() メソッドは、SYSLOG に図 4.8のようなメッセージが記録され次第、処理を呼び出し元に戻す。

12 行目では、得られた情報をもとにログインが組織外からのものであるかを判定している。ログイン元のホスト名は get_from() メソッドによって得ることができる。また、図 4.9の 7 行目からもわかるように、host 型 (ホスト名型) の情報として定義されているため host クラスの持つ in() メソッドを利用して、ドメイン名の照合を行うことができる。この例では、magp.co.jp というドメイン名以外のホストを組織外のホストとみなしている。

もし、組織外からのログインである場合は、13～19 行目で報告書を生成し管理者へ通知

```
class SysStat {

  method void run() {

    Report r = new Report();
    r.report_to = "yoshiaki@magp.co.jp";
    r.report_title = "System Status Report";

    boolean status = true;

    // 負荷およびメモリの状況を調査する
    LoadAvarage load = new LoadAvarage();
    VirtualMemory vm = new VirtualMemory();
    load.init();
    vm.init();

    // システムの負荷は高いか ?
    if (load.isHigh()) {
      status = false;
      r.println("Load avarage is high! (" + load.get_value() + ")");

      // CPU 使用率の高いプロセスを調べる
      ProcStat ps = new ProcStat();
      ps.init();
      String[] ret;
      if ((ret = ps.getHeavyProcReport()) != null) {
        r.println("Proccess List:");
        for (int i = 0; i < ret.length; ++i) {
          r.println("\t" + ret[i]);
        }
      }
    }
```

図 4.6 計算機の負荷および仮想メモリの使用状況を監視するクラス (続き)

する。報告される電子メールは例は図 3.12であり、ユーザインタフェースへ送られる報告書の例は図 3.11である。

4.4.2 実行例の考察

従来のシステム管理で、この実行例と同様の作業を行う場合、ログインに関する情報を特定のファイルに記録するように SYSLOG を設定し、これを定期的に参照することにな

```

    // メモリ不足か ?
    if (vm.isShort()) {
      status = false;
      r.println("Memory shortage!");

      // メモリ使用率の高いプロセスを調べる
      ProcStat ps = new ProcStat();
      ps.init();
      String[] ret;
      if ((ret = ps.getBigProcReport()) != null) {
        r.println("Proccess List:");
        for (int i = 0; i < ret.length; ++i) {
          r.println("\t" + ret[i]);
        }
      }
    }

    // 異常が発見された場合のみ報告を行なう
    if (!status) {
      r.sendMail();
      r.sendMagpCon();
    }

  }
}
```

図 4.7 計算機の負荷および仮想メモリの使用状況を監視するクラス (続き)

```
login from kame.morning.co.jp as poteta
```

図 4.8 `login` プログラムがログに記録するメッセージの例

る。また、情報が記録されたファイルに対しては、`sed` や `awk` といったテキスト処理のためのツールを利用した解析を行い、組織外からのログインの記録を抽出する。組織外からのログインの記録が抽出された場合、電子メールやグラフィカルユーザインタフェースなどのなんらかの監視ソフトウェアを用いて、システム管理者へ通知する。

従来のシステム管理においては、上記のような作業をシェルスクリプトを用いて記述し、定期的に実行することによって行うことが多かった。しかし、SYSLOG の利用方法、シェ

```
class LoginSuccess {

  info login_success {
    type = Syslog;
    cond_prog = "login";
    cond_msg  = "login from";
    form = ( nul nul from:host nul user:ident );
  }

}
```

図 4.9 `login` プログラムが SYSLOG を通して記録する検索するクラス

ルスクリプト、テキスト処理に関してなど、多くの知識が必要とされる。さらに、一度記述された、シェルスクリプトが他の作業にも再利用可能であること少く、再利用できるように記述すること自体も難しい。

この例からもわかる通り、知識ライブラリ記述言語では収集結果に対する解析方法を容易に記述することができる。さらに、解析結果はさまざまなメソッドを利用して、分析条件の記述に自由に利用できる。このように、従来のようなシェルスクリプトの記述によって行われていた作業を大幅に改善することができている。

また、ログインに関する情報を得るために、`LoginSuccess` クラスを用いてる。このクラスを記述する際には、もちろん SYSLOG に関する知識が必要とされる。しかし、クラスの記述以後、`LoginSuccess` クラスの利用者は SYSLOG に関する知識を必要とされない。

以上のように、magP の知識ライブラリ記述言語は 3.8.1 節で述べた目標を満たしていることが分かる。

```
class Login {

  method void run() {

    LoginSuccess ls;

    while (true) {

      ls = new LoginSuccess();   // SYSLOGから情報を収集するクラス
      ls.init();                 // メッセージが得られるまでブロック

      if (!ls.get_from().in("magp.co.jp")) {  // 組織外からか？
        Report r = new Report();
        r.report_to = "yoshiaki@magp.co.jp";
        r.report_title = "Login Warning";
        r.println("User " + ls.get_user() +
                  " login from " + ls.get_from().toString());
        r.sendMail();
        r.sendMagpCon();
      }

    }

  }

}
```

図 4.10 組織外からのログインを監視するクラス

第 5 章

magP による管理作業の改善

この章では、magP によって達成される管理作業の改善について説明する。

5.1 さまざまな情報収集手段の統合

現在、システム管理を支援するための管理アプリケーションはさまざまなものが存在する。しかし、あらゆる問題に適応可能な万能な管理アプリケーションは存在せず、実際にシステム管理作業を行なう場面では、作業に必要な情報を、さまざまな管理コマンドの実行結果や各種サービスのログファイルなどから得ている。そのため、対象となる情報ごとに収集方法が異なり、効率良く情報収集を行なうことは容易なことではない。

magP は、このようなさまざまなインタフェースを持った情報収集の手段を組み合わせて統合するとともに、あらかじめ標準的なインタフェースを提供することにより、新たなツールの開発を容易にする。

ネットワークの性能管理に関連する作業を支援するためのシステムとして ipsh[22] がある。この ipsh は、システム管理者が管理作業において頻繁に用いる外部コマンド (`ping`, `traceroute`, `netstat`) や SNMP を用いて情報を収集しネットワークの性能管理に関する定期的な情報の収集や整理を支援するシステムである。また、その動作の内容は利用者があらかじめ記述するスクリプトよって与えられる。このスクリプト言語は C 言語に近い手続き型言語で、プログラミングにある程度熟練している利用者であれば、比較的容易に理解することができる。しかし、利用できる情報収集の手段が固定であり、自由に拡張することができないこと、過去に記述したスクリプトを利用して新たな作業内容を記述するような開発形態の実現が難しいことなどが問題である。

5.2 管理者の熟練度に合わせたインタフェースの提供

計算機やネットワークが普及し、その規模も拡大しているため、管理者の不足はますます深刻な問題となっている。そこで、管理アプリケーションは熟練した管理者にも熟練してない管理者にも適切なインタフェースを提供する必要がある。magP の知識ライブラリでは、類似した情報を収集する際にも、異なるインタフェースを提供することが容易であるため、この問題を解決することができる (3.8節および図 3.13)。

例えば、4.3節の実行例では、計算機の負荷が高いことを確かめるために、`LoadAvarage` クラスの `isHigh()` メソッドを使用している。しかし、`LoadAvarage` クラスの `get_value()` メソッドによって実際に負荷の値を確かめる方法も考えられる。前者の方法は計算機の負荷に関する知識を必要としないため、熟練していない管理者も容易に利用することができる。一方、熟練した管理者は後者の方法によって具体的に負荷の値を用いてクラス定義を記述するもできる。

このように、熟練した管理者はより正確な詳しい情報をもとにして作業を行うことが可能になり、熟練していない管理者は少ない知識だけでも最低限の管理作業を行うことが可能になる。

5.3　インテグレーションが容易

従来、システム管理において、個々の問題ごとに専用に開発された管理アプリケーションは、さまざまな方針で設計され、異なるインタフェースを備えている。そのため、それらを互いに組み合わせて新たな管理アプリケーションを開発することは容易なことではなかった。

一方、magP では知識ライブラリ内の既存のクラス定義を用いて新たなクラス定義を記述することができるため、この問題を解決することができる。例えば、4.2節の実行例で利用した複数の計算機のファイルシステムの使用率を監視する `MultiSystems` クラスは、4.1節の実行例で用いた `DiskCapacity` クラスを利用して定義されている。このように、既存のクラスを再利用することによって、新たなクラスの開発に要するコストを減少させることができる。

5.4　高いポータビリティ

magP は、実行系を中心としたシステムの大部分が Java 言語によって実装されているため、Java 仮想マシンが動作する多くのオペレーティングシステム[1]で動作可能である。また、3.8.2節で述べた通り、知識ライブラリは知識ライブラリ記述言語によってシステムに依存する部分を分離することができるため、多くのクラスを異なるシステム間で共有できる。

ネットワークが多種多様なオペレーティングシステムによって構成されることが一般的である現在、このような magP の高いポータビリティは有効であると言える。

5.5　リビジョン管理

分散システムやエージェントシステムにおいて、分散して配置された各種サービスのリビジョン管理は特に重要である。分散して動作している複数のアプリケーションが協調して動作する場合、版が異なることによって正常に動作することができな可能性がある。特

---

[1] 現在動作が確認されているオペレーティングシステムについては3.9節を参照されたい。

に組織の規模が拡大するほど、この問題は深刻化する。magP は、知識ライブラリを集中管理することによってこの問題を解決している。具体的には、知識ライブラリをネットワークロードあるいは NFS によって共有することによって、magP のバージョン管理が容易にしている。また、組織内の設定変更などに柔軟に対応することができる (図 5.1)。

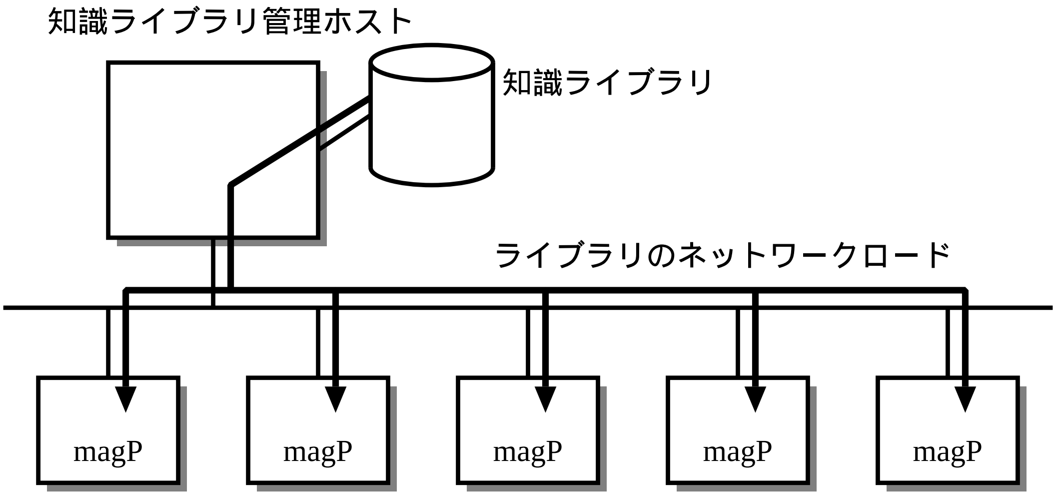


図 5.1 知識ライブラリのネットワークロード

5.6　システム管理に特有なデータ型の利用

magP はシステム管理に特有なデータ型を使用できるため、収集した情報の整理・分析作業を効率良く行うことができる。例えば、ホスト名・電子メールアドレス・時刻情報などは、シェルスクリプト、Tcl/Tk[5]、Java 言語などではそのまま扱うことは難しく、さらにその意味の解釈などは専用の手続きを用意する必要がある。一方、magP ではこれらに専用の型を用意しているため、ある組織内に所属する計算機であるかどうかを判定する、どのような組織に所属するユーザであるか調査する、各種サービスがログに記録するさまざまな形態の時刻情報を比較するなど、システム管理に頻繁に必要とされる処理を容易に記述することができる。

このようにして、magP では、言語の適用範囲をシステム管理アプリケーションの開発に限定することによって、記述力を向上を実現している。

5.7　障害の検知

インターネットの普及によって、システムが 24 時間正常に運用されることが重要になっている。しかし、従来の管理アプリケーションではポーリングなどの手段によって、障害

を検知する方法が一般的で、通信コスト、障害発生検知までの遅延時間が大きかった。また、ログに記録される情報についても、管理者が気付くまでの遅延時間が大きく無視できない。

magP では、さまざまな情報収集手段を自由に組み合わせて利用できるため、ポーリング以外の方法で障害の発生を検知できる。例えば、4.4節の実行例では、システムへのログインを受け付ける `login` というプログラムが記録するログを情報源とし、組織外からのログインを即座に検出することができる。また、この他にも SNMP のトラップを受け付ける機能を現在実装中であり、より効果的な障害検知作業の実現が可能になる。

第 6 章

今後の展望

この章では、magP をより効果的な管理支援システムとして運用してくために、研究開発が必要とされる要素について議論する。

6.1 記述インタフェース

前にも述べた通り、magP の知識ライブラリ記述言語は記述量減少のため多くの試みを行っているが、より多くのシステム管理者を支援するためには、プログラミング経験の少ないシステム管理者にも利用できる、知識ライブラリを開発環境を提供する必要がある。そのためには、たとえば、現在、広く普及しはじめているビジュアルプログラミングの技術を利用することによって、知識ライブラリをより容易に構築できるようにするなど、各分野の技術を効果的に取り入れていく必要がある。

6.2 セキュリティの向上

magP は、ユーザ名とホスト名によるアクセス制御、Java インタプリタによるセキュリティの考慮以外には、特にセキュリティ向上のための仕組みを備えていない。しかし、システムの管理情報を扱うため、通信の暗号化、ライブラリの正当性の認証などによって安全性を高める必要がある。また、HTTP で CMIP や SNMP をカプセル化し、ファイアウォールを通過させる技術 [7][9] なども開発利用されており、セキュリティ面の議論は引続き行ってゆきたい。

6.3 動作履歴の利用

動作履歴の記録をもとに、人工知能などの技術を用いて、あらかじめ想定していない障害の検知も可能になるのではないかと考えている。また、分析作業をトレースする機能など、magP の動作状況をより細かく把握できる仕組みも必要である。障害の原因が特定できない場合でも、どのような分析の過程を経てきたのかを履歴情報をもとに提示するのみでも、障害復旧作業に大きく貢献できる。

6.4　データベースの設計と実装

現在のmagPの実装では、収集した情報を記録・管理するためのデータベースの実装を完了していない。システム管理に必要な情報は、大量かつ類似した情報が多いという特色を持つ。このような情報の検索や不要なものの削除など、情報を効率良く扱う方法を考案する必要がある。また、過去に収集された情報を分析するには、どのような問い合わせが必要とされるか、汎用のデータベースでも適用可能であるか、あるいは専用のデータベースを開発する必要があるのかなど、引続き議論および実験を行う必要がある。

6.5　より柔軟な報告手段

分析結果をユーザにより把握しやすい形で提供する必要がある。現在、magPが用意している報告機構は、あらかじめ定められたフォーマットに従って報告書を生成し、メール、ページャ、FAXなどでユーザに通知するものである。

3.7節でも述べた通り、報告はシステム管理者の状態に応じて最適な報告手段を用いて行い、報告書も報告手段に応じて自動生成される必要がある。しかし、より情報の把握がしやすい報告書を自動生成するためには、magP内に存在する情報を平文に展開し人間の理解しやすい形に変換する機構の開発が必要である。

そのために、現在、大野研究室[1]で開発されている「メッセージ標準表現形式を用いた情報伝達機構」を利用することが考えられる。これは、あらゆる形態の情報を、一度「メッセージ標準表現形式」と呼ばれる共通構造に変換して扱い、さまざまな形態の情報に自由に再変換し伝達することができるシステムである。magPが分析結果の報告をこのメッセージ標準形式に従って出力することができれば、電子媒体・音声・画像などさまざまな媒体を利用した情報伝達が可能になる。

この他にもさまざまなメディアを利用した情報伝達に関連する研究は多くの分野でなされており、引続き研究開発を行っていく予定である。

6.6　実行系の再実装

現在、magPの実行系はJavaインタプリタで代用されている。また、知識ライブラリ自体もJavaインタプリタで処理される中間言語で記述されたクラスライブラリである。Java言語は、オブジェクト指向、ポータビリティ、インタオペラビリティ、すぐれた簡易性を備えた言語であり、汎用のプログラミング言語として非常にすぐれている。しかし、本論文では、管理アプリケーションの開発のみを対象にして特化した言語として設計した、知識ライブラリ記述言語を提案した。Javaインタプリタから不要な機能を取り除き、知識ライブラリ記述言語の処理に必要な部分のみを残す形で実行系を再実装することによって、処理速度の改善、システムの安定を実現する計画がある。

---

[1] 東京工業大学 情報理工学研究科 数理・計算科学専攻 大野研究室

6.7　他研究との連携

ここでは、今後の magP の開発において連携を予定している他の 3 つの研究について述べる。

6.7.1　NMW System

ネットワークワームを用いてネットワーク管理を行うエージェントシステムとして NMW System がある [16][15]。NMW System では、ネットワークワームがエージェントとして遠隔地の管理対象間を移動しながら調査を行い、システム管理者に結果を報告する。調査の間は非同期で処理が行われるため、システム管理者とエージェント間のコネクションは必要ない。そのため、ネットワーク上の障害に影響を受けることなく作業を続けることができる (図 6.1)。また、強力な暗号化によって安全に情報を運搬することができることも特徴の 1 つである。

NMW System では、ネットワークワームの各管理対象上での作業内容、管理対象間の移動経路などを Scheme 言語によって記述する。しかし、外部コマンドの実行やコマンドの出力結果の解析などは、Scheme 言語が提供するインタフェースをそのまま利用すること、スクリプトの再利用性がほとんど考慮されていないなど、magP の知識ライブラリ記述言語にくらべ記述力が高いとは言えない。また、複数のネットワークワームが協調して作業するなど、分散システムを効率良く管理するための仕組みは現状では提供されていない。

一方、magP では分散環境下でのオブジェクトアクセスが可能であり、知識ライブラリ記述言語によって分散システムにおけるさまざま情報収集・分析作業を記述することができる。しかし、システムに発生した障害の種類によっては、magP が正常に動作することができない場合がある。例えば、リモートメソッド呼び出しを行った直後に通信障害が発生した場合などである。

そこで、magP 間の通信手段として NMW System を採用し、より障害に強い安定した通信手段の獲得を考えている。

6.7.2　ネットワーク管理情報可聴化システム stetho

ネットワーク管理に必要な情報を音によって表現し、管理対象の状態を把握する作業を支援するシステムである [13]。音による情報表現は、特に、動的に変化し続ける情報、同時に複数の事象を表現する情報、種類の多い情報などに適している。そのため、システム管理においては、ネットワーク上を流れるパケットを音で表現することによって、トラフィックの変化を把握する作業において大きく貢献することが分かっている。

現在開発中の新しい stetho では、システムが観測部と表現部の 2 つに分離されている。観測部は、管理対象に関する情報の収集および分析を行い、表現部は観測部の分析結果をもとに管理対象の状態を音で表現し stetho のユーザであるシステム管理者に伝える。

システムの分離によって、magP は stedho の観測部を情報の収集手段として、表現部をより汎用的な情報表現モジュールとして利用できるようになる。特に、表現部を報告機構

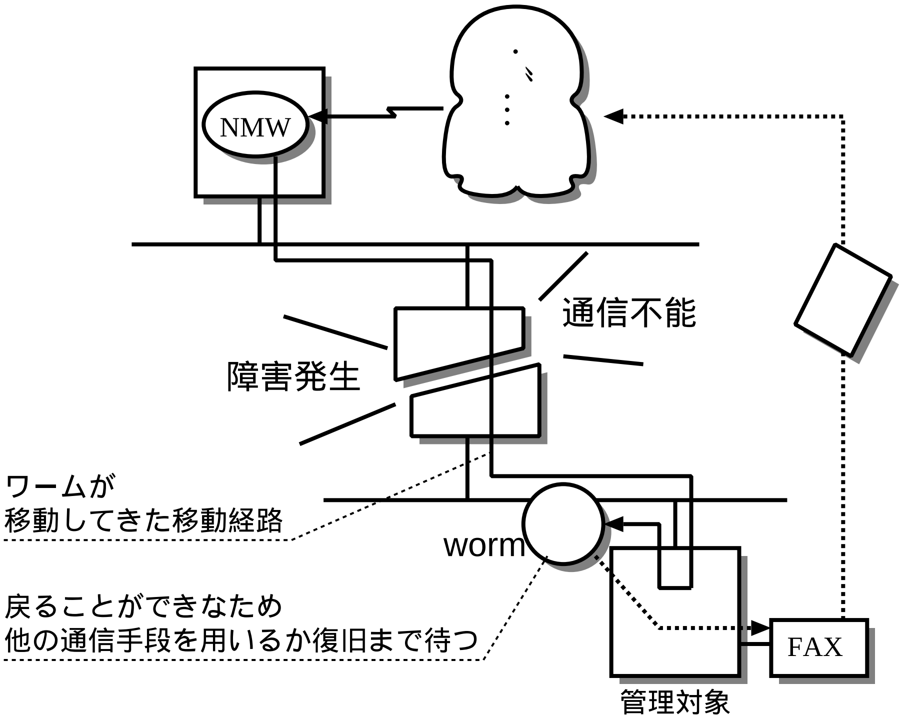


図 6.1 NMW System

の一部として利用し、音によってユーザに障害の発生や管理対象の状態を伝えることができれば、音の特性を利用した新しい監視体系の構築が可能である (図 6.2)。

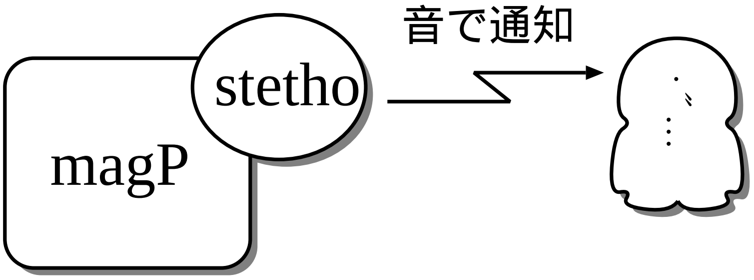


図 6.2 stetho との連携

6.7.3 PICKLES プロジェクト

PICKLES プロジェクトは、街角に公衆端末を多数設置することによって、いつでもどこでもネットワークを利用した情報交換できる環境を構築している [6]。多数の公衆端末を

一括管理するために、よりシステム管理者の負荷を軽減するような管理システムが望まれている。そこで、magP を PICKLES に搭載しより安定した公衆端末を実現する。

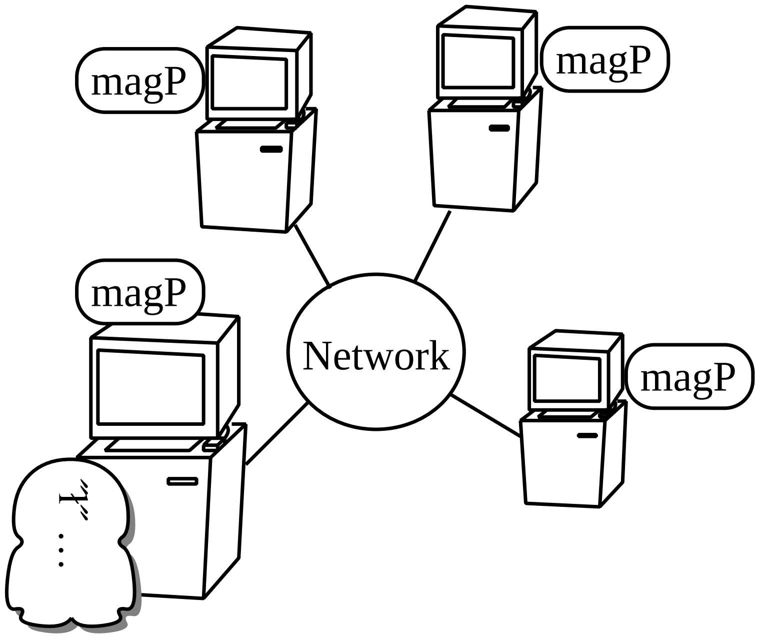


図 6.3 PICKLES プロジェクトとの連携

第7章

まとめ

本論文では、システム管理支援システムであるmagPの機能および構成、その実行例について述べ、magPによって従来のシステム管理をどのように改善することが可能であるかについて述べた。

magPでは、実際にシステム管理者が作業において用いるさまざまな道具や情報源に対して、共通のインタフェースを持たせることによって、情報の収集方法や分析方法の記述を容易にしている。また、階層化された知識ライブラリによって効率良く作業内容を記述することができる。これによって、システム管理者が繰り返し同じような順序でコマンドを実行するなどして行っていた、従来の情報収集および分析作業をほぼ自動化することができる。また、システムに発生する個々の問題ごとにシェルスクリプトを記述することに対して、magPは記述内容の再利用や情報収集手段の豊富さなどの面で優位性を持っている。

2.2節で述べた管理アプリケーションが満たすべき要素については、4章および5章の議論からも分かる通り、magPは「必要な計算機リソースが少い」「セキュリティに優れている」という二点を除いてはほぼ満たしている。

必要な計算機リソースが多いことの原因の1つに現在の実装がJavaインタプリタを用いていることが挙げられる。実用上十分な処理を行うために必要な計算機リソースの見積りを行うとともに、実装を見直す必要がある。また、セキュリティについて、現在は実行系へのアクセス制御以外に特別なことはしてないため、引続き議論および開発が必要である。

さらに、6章で述べたようなさまざまな課題を残している。そのような点を考慮しつつ、今後も開発を進めて行くつもりである。

謝辞

まず、学部4年生から修士1年生にかけての2年間、多くのことを教えて頂いた木村泉教授に心より感謝致します。

多くの助言を頂いた慶應大学SFCの新美誠さん、Titanet運用センターの飯島昭博さん、応用物理学科の山下雄三さん、WIDEプロジェクトのみなさんに感謝致します。

大野研究室・佐々研究室・柴山研究室の皆さんには、研究および論文作成にあたって、さまざまにお世話になりました。牛嶋哲さんからは研究テーマとなったシステム管理について、佐々木毅史さんからは食生活の大切さを学びました。白石昌靖さんには公私ともにお世話になりました。志築文太郎さんには、研究テーマの基礎部分からシステムの設計および論文作成に至るまで本当にお世話になりました。また、ともに3年間研究を行ってきた、木本雅彦君、成田哲也君、小野木渡君にも感謝致します。その他、卒業生の方々をはじめ多くの方々にお世話になりました。

そして、この3年間、研究にあたって熱心な御指導を頂いた大野浩之講師に心より感謝致します。

参考文献

[1] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstall, and James R. Davin. A Simple Network Management Protocol (SNMP). Request for Comments 1157, May 1990.

[2] D. Geiger, W. allen, A. Majtenyi and P. Reder. IBM cmipWorks: Technical Paper IBM. March 1997.

[3] David Flanagan. *Java in a Nutshell: A Desktop Quick Reference for Java Programmers*. O'Reilly & Associates, Inc., 1996.

[4] ISO/IEC. Common Management Information Protocol (CMIP) - Part 1: Specification ISO/IEC 9596-1. In *CCITT Recommendation X.711*. CCITT, 1991. Information Technology-OSI.

[5] J. Ousterhout. Tcl and the Tk Toolkit. Addinson-Wesley ISBN-0-201-63337-X.

[6] 木本 雅彦. 大学内公衆情報端末 (PICKLES) を用いたネットワーク利用環境の構築に関する研究. 平成 8 年度修士論文, 東京工業大学 大学院 情報理工学研究科数理・計算科学専攻, January 1997.

[7] L.Deri. Droplets: Breaking Monolithic Applications Apart. In *IBM Research Report RZ 2799*, September 1995.

[8] L.Deri. Network Management for the 90s. In *Proceedings of ECOOP '96 Workshop on Systems and Network Management*, July 1996.

[9] L.Deri. Surfini Network Management Resources Across the Web. In *Proceedings of 2nd Intl. IEEE Workshop on Systems and Network Management*, June 1996. Toronto.

[10] L.Deri. Static vs. Dynamic CMIP/SNMP Network Management Using CORBA. In *4th International Conference on Intelligence in Services and Networks*, 1997. `http://www.zurich.ibm.com/~lde/Corba_short.html`.

[11] Keith McCloghrie and Marshall T. Rose. Management Information Base for Network Management of TCP/IP-based internets. Request for Comments 1156, May 1990.

[12] Keith McCloghrie and Marshall T. Rose. Management Information Base for Network Management of TCP/IP-based internets: MIB-II. Request for Comments 1213, March 1991.

[13] 成田 哲也. ネットワークトラフィック可聴化システム stetho のネットワーク管理支援システムへの拡張. 平成 8 年度修士論文, 東京工業大学 大学院 情報理工学研究科数理・計算科学専攻, January 1997.

[14] Object Management Group. The Common Object Request Broker: Architecuture and Specification. July 1995. Revision 2.0.

[15] 小野木 渡. NMW System によるネットワーク管理とその評価. 平成 8 年度修士論文, 東京工業大学 大学院 情報理工学研究科数理・計算科学専攻, January 1997.

[16] 清水 亮博. 寄生プログラム (ネットワークワーム) を用いたコンピュータネットワークの管理方式. 平成 4 年度卒業論文, 東京工業大学理学部情報科学科, February 1993.

[17] Sun Microsystems. Java Application Programming Interface Documentation. 1996. `http://java.sun.com/products/JDK/CurrentRelease/api/`.

[18] X/Open Company Ltd. OSI-Abstract-Data Manipulation API (XOM). February 1994. X/Open Document C315.

[19] X/Open Company Ltd. Systems Management: Management Protocol API (XMP). March 1994. X/Open Document C306.

[20] 大鐘 久生. TCP/IP と OSI ネットワーク管理 ~SNMP と CMIP~. ソフト・リサーチ・センター, 1993. ISBN4-915778-21-5.

[21] 小野沢博文. 分散オブジェクト指向技術 CORBA. ソフト・リサーチ・センター, April 1996. ISBN4-915778-70-3.

[22] 門林 雄基, 山口 英, 宮原 秀夫. 利便性の向上と省力化を目的としたネットワークの 性能管理システム. インターネットコンファレンス'96 論文集. 日本ソフトウェア科学会・インターネットテクノロジ研究会・日本 UNIX ユーザ会・WIDE プロジェクト, March 1997.

付録 A
知識ライブラリ記述言語仕様

ここでは、知識ライブラリ記述言語の仕様を簡単に述べる。知識ライブラリ記述言語の文法は、そのが大部分が Java 言語に準拠しているため詳しくは Java 言語仕様を参照されたい。

A.1　字句

知識ライブラリのソースコードは、字句によって形成される。字句は、キーワード、名前、コメント、値、演算子などのことである。それぞれについて簡単に説明する。

キーワード キーワードは次のものである。

```
boolean  break   byte      case     catch
char     class   continue  default  do
double   else    finally   float    for
if       info    int       long     new
null     return  rnew      short    switch
try      void    while
```

名前 名前は、英数字およびアンダーバーによってなる文字列である。名前の先頭の文字は英字でなければならない。名前は「変数名、クラス名、メソッド名、ラベル」などを表すために用いる。

値 値には整数値、小数値、真偽値 (`true`,`false`)、文字、文字列がある。これらは Java 言語と同様の仕様であるので説明を省略する。

コメント 知識ライブラリ記述言語では、//から行末までをコメントとし、空白文字一つ分として解釈する。

演算子 演算子は次のものである。

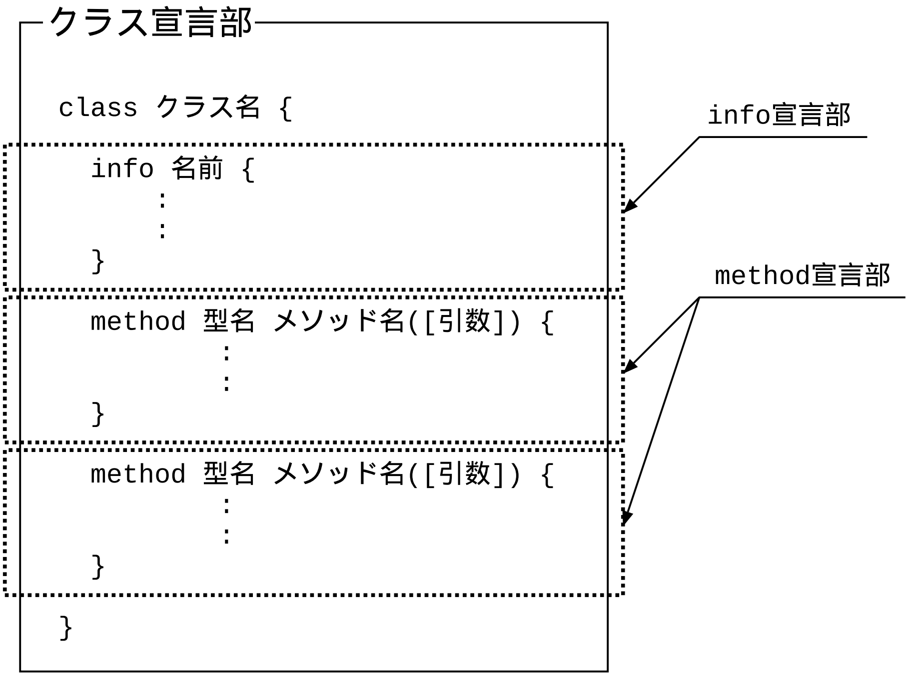
```
= > < ! \\ == <= >= != && || ++ -- + - * / & | ^ % << >> >>>
```

A.2　型および型の変換

Java 言語仕様を参照されたい。

A.3　class 宣言部

単一のソースコードファイルは、コメントと 0 個以上のクラス宣言部から構成される。クラス宣言部は、0 個以上の info 宣言部と 0 個以上の method 宣言部を含む。次のような構造になる (図 A.1)。



```
class クラス名 {
    info 名前 {
         :
         :
    }
    method 型名 メソッド名([引数]) {
             :
             :
    }
    method 型名 メソッド名([引数]) {
             :
             :
    }
}
```

図 A.1 クラス定義の構造

A.4　info 宣言部

info 宣言部は情報の収集方法に関する定義である。次のように記述する。

```
info 名前 {
  キーワード = 値 ;
  キーワード = 値 ;
          :
}
```

キーワードは図 A.1の通りである。表中の項目名 COM、FILE、SNMP、SL は情報収集のタイプを示し、それぞれ、外部コマンド型、外部ファイル型、SNMP 型、SYSLOG 型を表している。それぞの項目に書かれている○ は、キーワードがそのタイプで指定可能であることを示し、× は指定不可能であることを示している。

キーワード	値の型	内容	COM	FILE	SNMP	SL
type	シンボル	情報収集のタイプ	○	○	○	○
timeout	整数値	収集作業の制限時間	○	○	○	○
sep	文字列	結果の区切り文字	○	○	○	○
form	form リスト	結果のフォーマット	○	○	○	○
skip	整数値	無視する結果の数	○	○	○	×
comm	文字列	外部コマンド名	○	×	×	×
exec	シンボル	実行の方法（Complete，Step）	○	×	×	×
file	文字列	外部ファイル名	×	○	×	×
host	文字列	通信先のホスト名	×	×	○	○
mib	文字列	MIB 変数	×	×	○	×
cond_fac	文字列	ファシリティ	×	×	×	○
cond_pri	文字列	プライオリティ	×	×	×	○
cond_flags	整数値	フラグ	×	×	×	○
cond_btime	文字列	検索開始時刻	×	×	×	○
cond_etime	文字列	検索終了時刻	×	×	×	○
cond_host	文字列	ホスト名	×	×	×	○
cond_regex	文字列	正規表現	×	×	×	○
cond_prog	文字列	プログラム名	×	×	×	○
cond_pid	整数値	プロセス ID	×	×	×	○
cond_rev	真偽値	条件の反転	×	×	×	○

表 A.1 info 宣言部内で用いるキーワード

A.5 method 宣言部

method 宣言部はクラス外に対するインタフェースの定義である。次のように記述する。

```
method 名前 ([引数]) {
  キーワード = 値 ;
  キーワード = 値 ;
         :
}
```

A.6　配列

Java 言語同様の配列を使用することができる。

A.7　ブロック・文・式

ブロックは 0 個以上の式を中括弧 (’{’,’}’) で囲んだものである。文には次のようなものがある。

空文 セミコロン (;) のみの文である。

変数宣言 現在のブロック内でのみ有効な変数を宣言する。

```
型名 変数名 [配列指定] [= 初期値] ;
```

ラベル付き文 ラベル付き文には次のようなものがある。

```
case 定数式:  ブロック
default:  ブロック
```

式文 任意の式に続けてセミコロン (;) を書くことによって式文を作ることができる。

選択文 (`if`, `switch`) `if` 文は次のようなものである。

```
if (式) ブロック 1
if (式) ブロック 1 else ブロック 2
```

(式) は真偽値でなければならない。(式) の値が `true` である場合、ブロック 1 を実行する。後者の場合、(式) の値が `false` ならばブロック 2 を実行する。

`switch` 文は次のようなものである。

```
switch (式) {
case 定数式:  ブロック
case 定数式:  ブロック
case 定数式:  ブロック
        :
default:  ブロック
}
```

(式) の値と同じ値の定数式を持つ `case` 文へジャンプする。そのような `case` 文がない場合に、`default` 文があればそこへジャンプする。

繰り返し文 (`while`, `do`, `for`) `while` 文は次のようなものである。

```
while（式）ブロック
```

(式) の値が true である場合にブロックを実行する。ブロックの実行終了後、再び (式) を評価し (式) の値が `false` になるまでブロックの実行を繰り返す。

`do` 文は次のようなものである。

```
do ブロック while（式）
```

ブロックを実行し終了後に (式) を評価する。(式) の値が `true` である場合、再びブロックを実行する。(式) の値が `false` になるまでブロックの実行を繰り返し行う。

`for` 文は次のようなものである。

```
for（初期化式; 条件式; 増加式）ブロック
```

はじめに初期化式を実行する。続いて、条件式の値が `true` である場合、ブロックを実行する。ブロックの実行が終了すると、増加式を実行し条件式の値が `true` であれば、再びブロックを実行する。以後は、条件式の値が `false` になるまでこれが繰り返す。

ジャンプ文 (`continue`, `break`, `return`) `continue` 文は現在のブロックの最後にジャンプする。`break` 文は現在のブロックを抜け、次の文に制御を移す。`return` 文はメソッドを抜け、メソッドの呼出もとに返り値を返す。

```
continue ;
break ;
return ;
```

付録 B
サンプルコード

ここに、本論文中の実行例で用いたクラスのコードを載せる。

B.1　ファイルシステムの使用率の監視

この例で使用したクラスは次の通りである。

`FileSystem` クラス　図 4.2

`DiskCapacity` クラス　図 4.3

B.2　複数計算機の監視

この例で使用したクラスは次の通りである。

`FileSystem` クラス　図 4.2

`DiskCapacity` クラス　図 4.3

`MultiSystems` クラス　図 4.5

B.3　計算機の負荷および仮想メモリの使用状況の監視

この例で使用したクラスは次の通りである。

`SysStat` クラス　図 4.6および図 4.7

`LoadAvarage` クラス　図 B.1

`VirtualMemory` クラス　図 B.2

`ProcStat` クラス　図 B.3

```
class LoadAvarage {

  info load {
    type = ExternalCommand;        // 外部コマンド型を指定
    comm = "uptime";               // コマンド名は uptime
    form = (time:ident state:ident users:int tmp:",.*d" nul value:float);
  }

  method boolean isHigh() {
    return (get_value() >= 2.0);    // 負荷の値が 2.0 以上であれば true
  }

}
```

図 B.1 システムの負荷を調査するクラス

B.4 組織外からのログインの監視

この例で使用したクラスは次の通りである。

`LoginSuccess` クラス 図 4.9

`Login` クラス 図 4.10

```
class VirtualMemory {

  info proc {
    type = ExternalCommand;
    skip = 3;
    comm = "vmstat -c 4 1";
    form = ( run:int[2] blk:int[2] wait:int[2] avm:int);
  }

  method boolean isShort() {
    if (get_max_wait() > 0) {
      return (true);
    } else {
      return (false);
    }
  }

}
```

図 B.2 仮想メモリの状況を調査するクラス

```
class ProcStat {

  info proc {
    type = ExternalCommand;
    skip = 1;
    comm = "ps -auxww";
    form = ( name:ident pid:int cpu:float mem:float
             nul nul nul nul nul nul cmd:left );
  }

  method int[] getBigProcs() {
    return (get_index_max_mem(3));
  }

  method String[] getBigProcReport() {
    int[] index = getBigProcs();
    String[] ret = new String[index.length];
    for (int i = 0; i < index.length; ++i) {
      ret[i] = get_mem(index[i]) + "% " + get_name(index[i]) + " " +
               get_pid(index[i]) + " " + get_cmd(index[i]);
    }
    return (ret);
  }

  method int[] getHeavyProcs() {
    return (get_index_max_cpu(3));
  }

  method String[] getHeavyProcReport() {
    int[] index = getHeavyProcs();
    String[] ret = new String[index.length];
    for (int i = 0; i < index.length; ++i) {
      ret[i] = get_cpu(index[i]) + "% " + get_name(index[i]) + " " +
               get_pid(index[i]) + " " + get_cmd(index[i]);
    }
    return (ret);
  }

}
```

図 B.3 プロセスの状況を調査するクラス