管理情報を自動的に収集・分析する
ネットワーク管理支援系の設計と実装

東京工業大学
理学部情報科学科
中嶋 良彰
(91-2744-6)
平成六年度卒業論文

指導教官 木村 泉 教授
平成7年2月6日

目 次

図の一覧

表の一覧

第 1 章

はじめに

現在、計算機や計算機ネットワークの管理作業は、大部分管理者の人手によってなされているのが実情である。それは、管理者個人の知識や経験に頼る部分が多い複雑で時間のかかる作業である。また、障害の内容や調査結果に対して作業内容がはっきりと決定しているとは限らず、情報を整理し、状況に応じて作業しなければならない。管理者の「勘」に頼る部分も多い。その上、管理に用いる「道具」の数も多く、これによって管理対象から得られる情報の形態も多様である。それらすべてを把握し、効率よく管理を行なうためには、複雑な知識や経験が必要である。このように、現在の管理作業には「道具は存在するが、複雑な知識や豊富な経験が必要とされるため、多大な労力を必要とする」という問題がある。しかも、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現在、この問題はますます深刻化しつつある。この研究の目的は、熟練した管理者にとっては管理作業が軽減され、熟練していない管理者にもある程度管理作業が可能になるような、なんらかの手段を提供することである。

そこで、まず一般の管理作業のモデル化を行ない、このモデルをもとに管理作業支援系を設計した。特に、管理作業における「情報の収集」と「情報の分析」に注目し、この部分を中心に自動化を行なうことによって問題の解決を行なうことにした。そのためには、「収集する情報の内容」「情報の収集方法」「情報の分析方法」を決定するための知識、つまり管理作業に関する知識の表現方法が大きな問題となる。そこで、管理者が情報収集に用いる道具に注目し、その道具それぞれに対して情報の収集・分析のための規則を記述し、管理者の知識を表現することを考えた。本研究で実装した管理作業支援系は、このような必要な情報や規則が付加された管理者の道具を管理者の知識の単位として扱い、「管理知識データベース」に登録・管理することにより、管理知識を表現している。管理作業支援系は、この管理知識データベースに登録された管理知識を指定し実行することによって動作を始めると、その際管理知識に記述された情報や規則をもとに自動的に作業を行なう。

本論文の管理作業支援系は情報調査の結果が一定の形態で統一されていれば、大部分の情報について分析規則を記述できる。また、分析結果をもとに報告書を生成し、適切な手段で報告するという部分についても、いくつかの実験を行ない期待した通りの結果を得た。ただし、分析規則の記述文法は十分に熟成していないため、分析条件が満たされなかった場合の行動が記述できない、分析結果を報告に反映しにくいなどの問題が残っている。また、管理作業支援系の作業履歴が記録されていないため、過去の作業履歴を用いた分析を行なうことができないなど、管理知識データベースの自動生成に関しても課題が残っている。本論文ではオペレーティングシステムとしてUNIXを仮定し、UNIXを用いたシステムの管理作業について述べているが、管理作業の自動化のもととなるモデルの設計は、他のオペレーティングシステムにも応用可能であると考えている。

第 2 章

管理作業とその問題点

この章では、計算機や計算機ネットワークによって構成されるシステムの管理作業とその問題点について述べる。さらに、問題点を解決するためにどのような援助が必要であるのかについて述べる。

2.1　一般の管理作業

一般に管理作業とは次のことを目的として行なわれる作業であると考える。

- 問題の発見と解決 ... システム上の問題点や将来の問題となる点を発見し解決する
- 環境の改善 ... より良い環境を目指してシステムの改善を行なう

これら 2 つの作業の違いは作業の必要性にある。前者は、作業を行なわなければシステムに存在する問題を取り除くことができない、またはそのような問題が生じる可能性がある場合の作業である。それに対して後者は、作業によってシステムが改善されより良い環境にはなるが、現在の状態を維持できれば良い場合は特に必要ではない作業である。以下で、両者について詳しく述べる。

2.1.1　問題の発見と解決

管理者は計算機や計算機ネットワーク上に存在する問題点を発見し、これを何らかの手段によって解決することを目的として作業を行なう。例えば、「ある遠隔地の計算機と通信ができない」という問題が生じた場合、管理者はネットワークインタフェースの状態やネットワーク上のトラフィックなど必要な情報の収集・分析を行なう。そして、原因や対策を決定し問題の解決を行なう。ただし、このような現在生じている障害に対して迅速に対応し処置を行うことも必要であるが、定期的な監視作業によって問題点を発見し、将来起こり得るであろう障害を未然に防ぐことも必要である。

また、一口に問題といってもその原因はさまざまである。ネットワーク、メモリ、ディスク、カーネル、計算機上の様々なサービスなど、それぞれさまざまな問題の原因になり得るため、それぞれについて調査・分析するための知識が必要となる。 例えば、ネットワーク障害はシステムにとって致命的なものになる。このネットワーク障害としては、「外部ホストとの通信障害」「通信の盗聴」などがあり、「通信障害」の原因として以下のものが考えられる。

- ルーティング異常
- 物理的接続異常
- インターフェース異常

- 入出力パケット異常
- ネットワークの過重トラフィック

このように、障害の原因はさまざまなものが存在するため、問題発見のために必要な情報の収集・分析は、管理者の持っている知識や経験に依存する部分が大きく、熟練していない管理者にとっては困難な作業となる。また、ネットワーク障害以外にも、メモリやディスクの障害など、利用者の環境を大きく左右する要因はさまざまなものが存在し、管理に必要な知識や経験は膨大なものになる。

2.1.2 より良い環境を構築するためのシステムの改善

管理者は、より良い環境の構築を目的として、システムの改善を行なわなければならない。このとき、どのような環境を「良い環境」と考えられるかが問題である。しかも、これは立場や価値観などによってさまざまに変化する。そこで、筆者の考える「良い環境」について以下に示し、それぞれについて考えを述べる。

1. 計算機資的要因
    - (a) 処理が高速である
    - (b) 保安性 (セキュリティ) に優れている
    - (c) 利用できるアプリケーションが豊富
    - (d) 計算機、ネットワーク資源が豊富
2. 人的要因
    - (a) 利用者の要求に迅速に対応できる
    - (b) システムへの適切な投資が行われている

処理が高速 利用者はある作業を行なおうとしたとき、「実行時間が長い」という事態に直面すると、システムの性能が悪いと思うかもしれない。このとき問題となるのは、処理が高速であるかどうかである。システムの処理が高速であれば、実行時間は短くなり利用者にとって良い環境となる。ただし、利用者が行なおうとした作業自体に原因がある場合も考えられ、「実行に時間がかかる」原因を追求し解決することによって、処理の高速化が可能となる。

保安性 (セキュリティ) に優れている データが無許可のユーザーによって読まれたり書き換えられたりせず、予期せぬ災害や誤操作などによってデータが失われることがなければ、そのシステムは安全である。また、そのようなシステムであれば、利用者にとって環境が良いといえる。

利用できるアプリケーションが豊富 利用者が安心して利用できるアプリケーションが豊富であることは、利用者にとって良い環境であるといえる。このためには、常に必要と思われるアプリケーションを導入し、古いものや問題のあるものを修正・更新する作業が必要になる。

システム資源が豊富 システムの資源とは、CPU・メモリ・入出力資源といったものである。入出力資源とは、ディスクやネットワークのようなもののことである。これらを利用者全体に十分に公平に提供することできれば、システムの性能あるいは環境の改善を行なうことができる。また、逆に十分に提供できなかったり、何らかの障害や問題を抱えている場合、利用者にとって良い環境であるとはいえない。

利用者の要求に迅速に対応できる　利用者が管理者に、「改善要求」「質問」などシステムに関する要求を容易に行なうことができ、管理者がこの要求に迅速に対応することができれば、利用者にとって良い環境であるといえる。また、利用者が要求を行なう前に、その要求に対応することができればさらに良い。

利用者の要求には、システムに生じた障害の解決要求や、システムの性能改善の要求などさまざまなものが考えられる。また、ネットワーク上の計算機の接続や移動のような拡張・変更作業に関する要求も考えられる。いずれにしても、管理者がどのような要求に対しても迅速に対応できるようなシステムであれば、利用者は安心して使用することができるはずである。

システムへの適切な投資が行われている　システムの状態を的確に判断し必要な投資を行なうことで、システムの性能を改善することができる。もちろん、どこにどの程度の投資を行なうのかが問題である。例えばメモリの場合、システムがメモリをどの程度必要とするのかを見積もらなければならない。そのためには、システムのピーク時におけるメモリの使用量を調査することになる。その結果、現在のメモリ量では十分でないと判断された場合、メモリの拡張を施さなければならない。

また、管理者はシステムの他の部分についても現在の状態を判断し、適切に必要な投資を行い、可能な限り性能の改善をはからなければらない。また、利用者のシステムの利用の仕方も監視する必要があると考えられる。

2.1.3　管理作業の手順

前節で述べたように、管理作業にはさまざまな作業が存在するが、どの管理作業も以下の3段階の作業で構成される。

1. 管理ツールを用いて調査を行なう
2. 得られた情報を分析し対策を考える
3. 実際に作業を行ない、必要であれば1.2.3.を繰り返す

まず調査すべきと判断した対象を適当な方法によって調査し、得られた情報を分析し状況を把握する。その結果、問題や改善すべき点が発見されれば、それに応じて実際に作業を行なうことになる。その後、必要であれば再び調査・分析を行ない作業を繰り返す。このとき、調査方法を決定するための知識や、得られた情報をもとに問題や障害を発見するため知識が特に重要といえる。

2.2　管理作業の問題点

現在、管理作業には次のような問題が存在する。

1. 管理に関する知識が複雑なこと
2. 管理情報の収集や分析を行なうために豊富な経験が必要とされること
3. 管理作業の手作業が占める割合が高いこと
4. 管理ツールが多種多様ですべてを把握することが難しいこと

5. 管理対象の拡大によって管理者が不足すること
6. さまざまな利用者が存在するため、利用者の立場にたった判断基準が定まらないこと

システム管理を行なうために必要とされる知識は複雑である。また、これらの知識は文献 [3] などから得ることもできるが、管理者のそれまでの経験によるところが大きいため、管理知識の蓄積は容易な作業ではない。そのため、計算機が普及し管理すべき対象が増加すればするほど、十分な知識を持つ管理者は不足することになる [10]。

また、管理作業は管理者の手作業の形になることが多く、同じような作業が続くこともあるため、作業総量はかなり大きなものとなる。

管理作業に用いるツールが多種多様であることも問題である。管理者は様々な管理ツールを状況に応じて使い分けることになるので、管理ツールに関して必要な知識も膨大なものになる。そのため、効果的に作業を行なうために、数多くのツールに関する知識や経験が必要である。

そして、最も大きな問題は、複雑なシステム管理のための知識や経験により得られる知識をすべて用いたとしても、必ずしも正しい結論が得られるとは限らないという点である。管理作業において、ある状況に対して必要な作業が一意に定まるということはほとんどないため、時には管理者の「かん」に頼らざるを得ないこともある。

このように、現在の管理作業は様々な問題を抱えており、これら問題を解決するために管理者を援助する方法を考える。

2.3 管理作業の援助

2.2節で述べた管理作業の問題を解決するためには、どのような援助を行なえば良いだろうか。例えば、障害発生時の作業について考える。

1. 障害の発生を知らせる
2. 発生した障害の原因、対策を決定する上で必要となる情報を提供する
3. 過去の類似した障害の原因、対策の情報を提供する

まず、障害の発生を管理者に知らせ、発生した障害の「内容」およびその「原因・対策」など、障害に関する情報を提供する。また、障害の原因や対策ではなく、管理者が原因や対策を決定する上で必要となる情報を提供するだけでもよい。例えば、過去の類似した障害の原因やその障害に対して行なった作業内容のような情報である。逆に原因や対策が明らかであれば、ある程度自動的に作業を行なわせることも可能であろう。

提供する情報が適切に整理・分析され、管理者が利用しやすいような形で提供されること、不必要な情報は提供されないことも重要な要素である。また、最終的に行なった作業記録を整理保管すれば、後の作業で利用することができる。

このように、管理作業では情報の収集・分析が基本であり特に重要である。しかし、この情報の収集作業では、調査対象、調査方法などを決定するための知識が必要である。また、分析作業では情報を分析するための知識も必要であろう。つまり、通常、管理者がこのような作業の際に用いている知識をどのように表現するかが重要な問題である。

本論文では、この情報の収集・分析を中心に自動化することによって管理作業の援助を行なうことを考える。次章では管理作業のモデル化を行ない、管理作業支援系の設計について述べる。

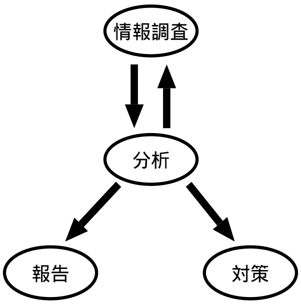
第 3 章

管理作業支援系の設計

この章では、システム管理作業のモデル化と、このモデルに基づいて行なった管理作業支援系の設計について述べる。

3.1 管理作業のモデル化

管理者によるシステム管理作業のモデル化を行なった。まず、管理作業を次のように「情報調査」「分析」「報告」「対策」の 4 つの部分に分割する。



情報調査 情報の調査を行なう。

分析 調査結果をもとに分析を行ない、再調査・報告・対策のいずれかを行なう。

報告 分析の結果を適当な手段を用いて報告する。

対策 分析の結果に従って対策を講じる。

「情報調査」では、「入出力の状態」「ネットワークの状態」「メモリの状態」など、管理作業において必要となる情報を収集する。管理者は収集すべき情報ごとに、適当な管理ツールや調査対象を選び作業を行なう。通常はある程度決まった対象を定期的に調査し監視する。このときに得られた情報の分析結果によっては、通常は調査しない対象も含めてさらに何度か調査を繰り返す場合もあり、次第にシステムの正確な状況を得ることができる。

「分析」では、得られた情報からシステムの状況を診断する。また、診断に必要な情報が十分に揃っていない場合は、適当な手段を決定し再調査を行なう。システムの状況を診断することができた場合、異常がないことが認められれば、その報告を行なうための作業に移る。また、問題点が発見されれば、状況を正確に把握し対策を考えることになる。その際、さらに情報が必要であれば再調査を行なう。このように、「分析」は管理者による思考作業である。

「報告」では、分析の結果、報告すべきであると判断された事実を適切な手段によって報告する。通常の管理作業においては、電子メール、ページャーシステム、FAX などが用いられる。

「対策」では、分析の結果決定された対策に従って処理を行なう。分析部分ですべての思考作業を行ない、情報調査・分析・報告・対策の 4 つの部分を明確に分離するため、「対策」部分では分析結果に従って手順通りに作業を行なうだけで良いという状態になる。

本論文では、以上のような4つの部分で構成される管理作業モデルをもとに、管理作業支援系の設計と実装を行なった。以降では、これらをセクションと呼び、それぞれ「情報調査セクション」「分析セクション」「報告セクション」「対策セクション」と呼ぶ。2章でも述べた通り、特に「情報調査セクション」「分析セクション」を中心に自動化を行なう。

3.2 管理作業支援系の設計方針

この節では、管理作業支援系の設計方針について述べる。ただし、本論文では実装の容易さを重要視し「中央見張りモデル」に従って管理作業支援系を設計し実装した。分散型で設計する場合、支援系それぞれの協調のさせ方や分散する情報の扱い方などを決定する必要があり、これらは今後の課題とする。

3.2.1 中央見張りモデル

ある1つの管理ホストが、自分自身及びネットワーク上の他の管理対象を管理する場合である。1つの管理ホストがネットワークや他の管理対象の情報を調査・分析し、自分自身を含めたシステム全体の管理を行なう。この管理ホストにあらゆる情報が集中され、管理ホストの運営にのみ注意を払えば良い点や、後で述べる「分散生き残りモデル」(3.2.2節)に比べて、実装が容易である点が長所である。一方、管理ホストが何らかの障害によって管理不能な状態に陥った場合、それ以後システム全体が管理不能な状態となってしまう点が大きな問題である。この問題については、緊急時に管理ホストの代わりを勤めるホストを用意することによって解決しようという考え方[4]もあるが、管理ホスト切替えの困難さなど問題も多く、根本的な解決策にはなっていない。この方針に従って実装する場合、管理作業支援系は管理ホスト上で動作することになる。

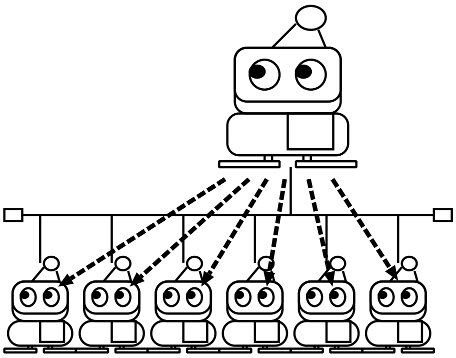


図 3.1 中央見張りモデル

3.2.2 分散生き残りモデル

このモデルは、「中央見張りモデル」(3.2.1節) のように、システムの生命線となってしまうような管理ホストを作らないことが特徴である。システム上の 1 つ 1 つのホストが独自に「調査・分析・報告・対策」による管理作業を行ない、自分自身やその周辺に関する情報の調査・分析を行なう。各ホストは、結果的に、生き残ることを第 1 の目的として作業を行なうことになる。また必要であれば、各ホストが互いに協調し合いながら管理作業を行なうこともできる。

さらに、このモデルはあるホストが何らかの障害によって使用不能な状態に陥ることがあっても、システム全体としては完全に管理不能な状態に陥るわけではないため、「中央見張りモデル」に比べ障害に強いといえる。ただし、各ホストが独自に調査を行なうため、情報が分散する可能性があることや、協調作業を行なうための実装の困難さなどが問題である。また、「中央見張りモデル」は管理作業を行なうホストが 1 つで良いのに対して、「分散生き残りモデル」では各ホストで管理作業が行なわれるため、システム全体としての効率が悪い面も問題である。また、この方針で実装する場合、管理作業支援系はシステムの各ホスト上で動作する。

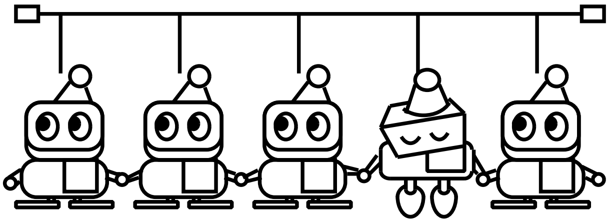


図 3.2 分散生き残りモデル

3.2.3 分散見張りモデル

このモデルは「中央見張りモデル」「分散生き残りモデル」の両者の考え方を組み合わせたものである。通常は複数存在する管理ホストが分担して管理を行なうが、1 つの管理ホストが障害によって管理不能な状態に陥った場合、他の管理ホストが管理を代行する。管理ホストの数を増やす程システム全体としては障害に強くなるが、分散生き残りモデルとほぼ同様な形態となってしまう。つまり、管理ホストの数によって中央見張りモデルと分散生き残りモデルのどちらの性質がより多く現われるかが決まる。この方針で実装する場合、管理作業支援系は管理ホスト上で動作する。

3.3 管理者の知識

この節では、管理作業支援系での管理者の知識の表現方法について述べる。

通常の管理作業において、管理者は何らかの手段によって情報の収集を行なう。例えば、その手段はシェル[1] 上で実行するコマンドであり、得られる情報はそのコマンドの出力である。管理者はこの情報を分析

---

[1] コマンド・インタープリタ。ユーザとオペレーティングシステムの間の仲立ちをする。

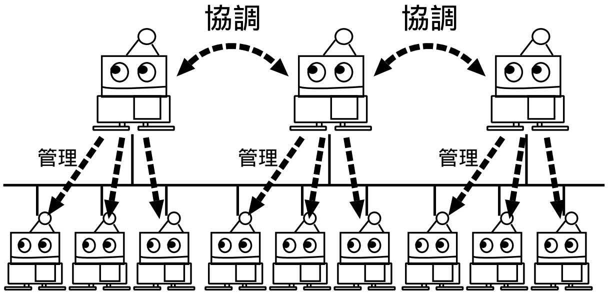


図 3.3 分散見張りモデル

し、次の作業を決定する。管理者がある初期状態 A でコマンド 1 を実行したとする。管理者はこのコマンド 1 によって得られた情報をある規則に従って分析する。分析の結果、新たにコマンド 2 によって別の情報が必要になる可能性もあれば、この 1 回の調査で結論に達する可能性もある。

いずれにしても、この時点で管理者の頭の中には簡単な調査計画ができあがっている。この計画には明確な結論があるわけではないが、「このような出力であれば次はこの作業を行なう」という程度のものはすでにできあがっている。次の作業が情報の再調査である場合は同様に分析が繰り返され、管理者の頭の中の調査計画はさらに正確なものになり、これを何度か繰り返した後、初期状態 A からの情報の収集・分析の手順が定まる。

さらに、別の管理作業の機会に初期状態 A に類似した状態で作業を行なう場合、最初に実行するコマンドがコマンド 1 でも、情報の分析結果によってはその後の行動が異なってくる。そのため、何度かの調査・分析が繰り返され後、再び別の情報の収集・分析の手順が決まる。

そして、このような情報の収集・分析の手順をまとめることによって、初期状態 A やそれに類似した状態から開始される管理作業における分析規則ができあがる。これを表わしたものが図 3.3である。

この時、ある状態から次にどの状態への遷移するのかは、コマンドの実行結果によって決まる。つまり、このコマンドごとに、コマンド自体の性質、実行結果の形態、実行結果の分析規則などを記述しまとめることができれば、これをコマンドに関する知識と考えることができる。そして、このようなコマンドに関する知識のつながり合いで管理者の知識を表現することができる。

また、管理者の情報収集のための手段は、シェル上で実行するコマンドだけでなく、後述するするリモートシェル、SNMP [1, 7, 11]、NMW [8] などを用いることもできる。そこで、本論文ではこのような情報収集の手段を「管理コマンド」と呼び、この管理コマンドごとに管理者の知識を表現したものを「管理知識」と呼ぶ。例えば、ローカルシステムの負荷監視作業を行なうためには、必要な管理コマンドが`uptime` であること、`uptime` の実行結果の形態、その分析方法などの知識が必要である。そこで、これらの知識をまとめたものがローカルシステムの負荷監視作業に関する管理知識となる。

本論文の管理作業支援系は、この管理知識を解釈し自動的に情報を収集・分析することによって、管理者の援助を行なう。また、管理作業支援系はある 1 つの管理ホスト上で動作し、ネットワーク上の管理対象の監視を行なう。

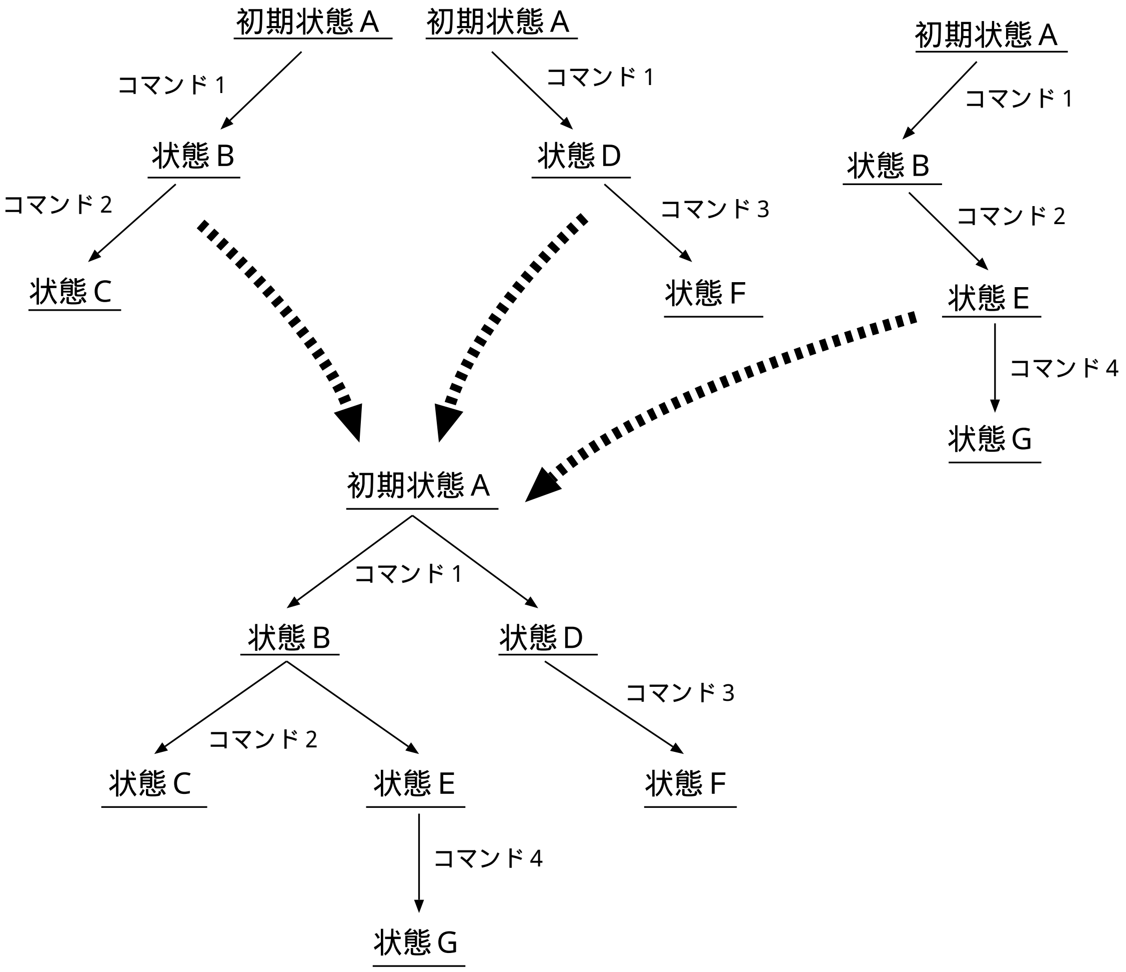


図 3.4 管理者の知識

ある初期状態 A からのいくつかの分析過程をまとめることによって、初期状態 A からの分析ルールができ上がる。このとき、最初に実行したコマンドに注目すると、このコマンドごとに、管理知識が記述できると考えられる。

第 4 章

管理作業支援系の実装

この章では、まず、管理作業支援系を構成について述べる。続いて、本論文で実装を行なった管理作業支援系実行部 magP について述べ、magP の動作の中心となる管理コマンドと管理知識について述べる。

4.1 管理作業支援系

管理作業支援系は次の 3 つのプログラムで構成される。

実行部	管理知識データベースに登録されている管理知識を 1 つ指定することで動作を始める。管理知識記述文法に従って記述された管理知識をもとに、情報の調査・分析を行ない分析結果に基づいて作業を行なう。本論文で実装を行なった部分である。プログラム名は「magP」[1]。
常駐部	常に動作を続け、magP のスケジュール管理と実行、管理知識データベースの管理および自動生成、magP の動作記録の管理などを行なう。magP の実行は知識名を 1 つ指定して行なう。現在は未実装である。プログラム名は「magPbase」。
操作部	管理作業支援系のユーザインタフェースである。利用者 (管理者) はこれを用いて常駐部の操作を行ない、実行部「magP」を直接操作することはない。現在は未実装である。プログラム名は「magpp」。

4.2 管理作業支援系実行部 magP の動作

本論文で実装した管理作業支援系実行部 magP は 3.3節で述べた管理知識をもとに動作を行なう。この管理知識は、その作業に必要な管理コマンド、その管理コマンドの出力形態、出力結果の分析規則などを含んでいる。また、管理知識データベースはこのような管理知識の集まりである。

まず、現在の実装では magP はシェル上で

```
% magP 管理知識名
```

のように実行される。このようにコマンドライン引数に行なう管理作業を表わす管理知識の名前を指定する。magP は動作を始めると、指定された管理知識を管理知識データベースから取り出す。次に、この管

---

[1] magP の由来は「magpie(カササギ)」という、黒白の羽毛で鳴き声がやかましく様々な小さなもの (管理情報) を巣に集めてくる動物である。

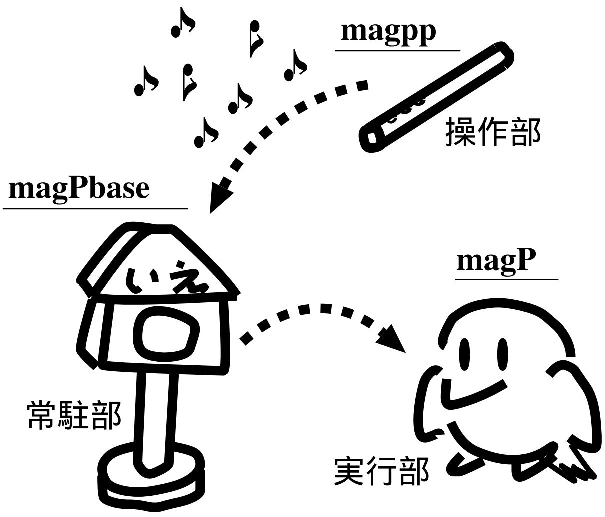


図 4.1 管理作業支援系の構成

理知識に従って管理コマンドの実行を行ない、情報収集を行なう。これが図 3.1の情報調査セクションでの作業にあたる。

magP は管理コマンドの実行結果を得ると、管理知識に含まれる分析規則に従って実行結果の分析を行なう。このとき、分析規則によっては 3.3節で述べたように情報の収集・分析を繰り返し行なうこともある。これは分析の過程で、別の管理知識で表現される作業によって得られる情報が必要なった場合である。この場合、この作業に関する管理知識を、管理知識データベースから取り出し、この知識に従って情報の収集・分析が行なわれるため、ある管理知識から別の管理知識が呼び出される形になる。そして、呼び出された管理知識で表される作業の結果をもとに、呼出側の分析が継続される。このように最終的な結論が得られるまで情報の収集・分析が繰り返される。また、これは図 3.1のモデルでは、情報調査セクションと分析セクションを行き来することに相当する。

結論に達した場合、ここでいくつかの作業が選択される。選択の対象となる作業は「報告」「対策」「再調査」であるが、この中から複数を選択することが可能である。「報告」は、分析の過程やその結論についての報告を行なう。報告先が自由に指定可能であることはもちろん、報告のための手段も電子メール、ページャーシステムが利用できる。報告内容はあらかじめ記述されている報告書の定義をもとに、その都度報告書が生成される。「対策」については現在は未実装であり将来の課題であるが、これについては後で述べることにする。「再調査」を選択した場合、新たに管理知識データベース中の管理知識が呼び出され、作業が行なわれる。

以上のように、管理作業支援形実行部 magP は、管理知識データベースに基づいて動作する。そこで以降の節では、この管理知識の中心である「管理コマンド」と「管理知識の記述方法」について述べる。

4.3 管理コマンド

本論文で、情報を収集するために管理者が用いる手段を管理コマンドと呼ぶことは 3.3節ですでに述べた。管理作業支援系で扱う管理知識は、この管理コマンドごとに記述される。この管理コマンドは、本論文で新たに設計・実装したものではなく、すでに広く管理作業に用いられているものである。また、前節

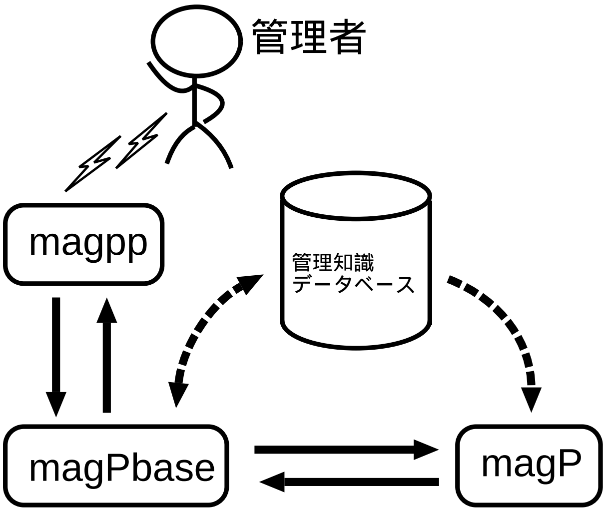


図 4.2 管理作業支援系と管理知識データベース

で述べた通りシェル上で実行するコマンドだけでなく、リモートシェルや SNMP を用いることもでき、将来は NMW System を利用することも考えている。そこでここでは、これらの管理コマンドについて詳しく述べる。

4.3.1 シェル上で実行するコマンド

シェル上で実行するコマンドで BSD UNIX [2] で利用可能なもののうち、主なものをその用途によって分類したが表 4.1である。例えば、`uptime` コマンドによって計算機の負荷を調べることができる。図 4.3にその使用例を示す。また、図 4.4はネットワークに関する情報を得ることができる`netstat` コマンドの使用例である。このように、UNIX では管理に用いるコマンドは文字情報で情報を出力することが多い。そこで、これらのコマンドの出力から、文字列や数値を読みとり分析を行なう。

表 4.1 ローカルホストで用いる UNIX の管理コマンド

調査場所	コマンドの例
プロセス	`uptime, ps`
メモリ	`pstat, vmstat`
ディスク	`df, du, iostat disklabel, fstat, repquota`
入出力	`iostat, slstat, nfsstat`
カーネル[3]	`dmesg, pstat`
ネットワーク	`ifconfig, tcpdump,netstat`

---

[2] AT&T ベル研究所で開発されている UNIX を、カリフォルニア大学バークレー校で 1977 年以来独自に強化、改良し、配布している UNIX。

[3] カーネルとは、UNIX の心臓部であり、メモリ管理、プロセススケジューリング、入出力の管理などシステム上の低レベルの仕事をするプログラムのことである。

```
% uptime
  5:35pm  up 2 days,  1:35,  14 users,  load average:0.43, 0.27, 0.01
%
```

図 4.3 uptime の使用例

```
% netstat -i
Name  Mtu   Network/Dest    Address              Ipkts Ierrs    Opkts Oerrs  Coll
en0   1500  131.112.40      nsk                3550305     0  3084545     4 35468
en1   1500  131.112.40.64   iskim-gw           2454589     0  2194182     0 19289
sl0*  296   none            none                     0     0        0     0     0
sl1*  296   none            none                     0     0        0     0     0
lo0   1536  127             localhost           353826     0   353826     0     0
at0   576   131.112.40.192  islt-gw               6771     0     3924     0     0
%
```

図 4.4 netstat の使用例

4.3.2 リモートシェル

リモートシェルは BSD UNIX に用意されているネットワーク機能の 1 つである。リモートシェルはリモートホスト上で指定したコマンドを実行するために用いられる。具体的にはシェル上で

```
% rsh ホスト名 コマンド
```

のように実行する。このとき、リモートホスト上にはこのrsh コマンドの要求にこたえるためのサーバプログラム (rshd など) が動作する。リモートシェルの使用が可能であれば、ローカルホストと同様にリモートホスト上で管理ツールを用いることができる。ただし、リモートホストへのログインの際にパスワードの照合の省略ができる必要があり、ローカルホストとユーザがリモートホストにとって信用できるものとして設定されていなければならない。そうでない場合はリモートシェルは使用できない。例えば、図 4.5がリモートシェルの使用例である。この例では、ホスト anne からホスト mike 上でコマンドdf を実行している。

4.3.3 SNMP (Simple Network Management Protocol)

SNMP は 1990 年に Jeffrey D. Case、Mark S. Fedor、Martin L. Schoffstall、James R. Davin によって提案されたネットワーク管理のためのプロトコルであり、ネットワーク上の管理対象 (ホストマシン、ルータなど) の情報をネットワークを通じて交換するために用いられる。管理対象には通常snmpd などのエージェントが存在し、このエージェントとのメッセージ交換によって管理対象の情報を得ることができる。つまり、リモートホスト上に SNMP エージェントを用意すればリモートホストの情報を得ることができる。

```
% hostname
anne
% rsh mike df
Filesystem           512-blocks    Used   Avail Capacity  Mounted on
/dev/wd0a                 15892   12596    1706    88%    /
/dev/sd1h               1922648 1009456  720920    58%    /usr
mfs:13                    15768    3104   11080    22%    /tmp
nsb:/export/sa          3923420 3340743  190335    95%    /am/nsb/export/sa
amd:15540                     0       0       0   100%    /auto
amd:15540                     0       0       0   100%    /home
nsb:/var/spool/rwho      282198   21768  232210     9%    /am/nsb/var/spool/rwho
nsk:/export/home.nsk    3923420 3296649  234429    93%    /am/nsk/export/home.nsk
%
```

図 4.5 リモートシェルを用いた情報調査の例

SNMPで扱う情報はMIB(Management Information Base) [5] [6]と呼ばれるオブジェクト識別子によって指定され、これは図4.7のような木構造の形で定義されている。そして、管理オブジェクトすべてに対して単一の識別子が対応している。例えば「システムの説明 (system description)」は、

`.iso.org.dod.internet.mgmt.mib-2.system.sysDescr.0`

という識別子で指定される。

`.1.3.6.1.2.1.1.1.0`

のような略記も可能である (図4.7参照)。

SNMPエージェントに値を問い合わせる場合は、この識別子を用いる。SNMPパッケージの1つであるCMU-SNMPでは、MIB識別子で指定された情報を得るためにsnmpgetというコマンドを利用できる。snmpgetは図4.6のように、エージェントホスト名、コミュニティ名、MIB識別子を指定して実行され、識別子に対応する値を実行結果として出力する。コミュニティ名とは一種のパスワードのようなもので、これを用いてエージェントへのアクセス制限を行なう。エージェントは受け取ったコミュニティ名が未登録のものである場合は要求を拒否する。例えば、図4.6で用いているpublicというコミュニティ名では、特に重大な影響を与えないような一般的な要求 (変数の値を参照するなど) のみ受け付ける。このように、コミュニティ名によってエージェントが受け付ける要求の制限を行なう。また、SNMPはMIBが示す値の参照だけでなく変更も可能である。MIBはASN.1(Abstract Syntax Notation 1: 抽象構文記法1) [2]と呼ばれる構文で記述される。この記法に従えばMIBを自由に拡張することができる。例えば、MIBには

```
% snmpget -v 1 nsb public .iso.org.dod.internet.mgmt.mib-2.system.1.0
system.1.0 = "SONY NEWS-Workstation, NEWS-OS 4.2; SNMP MIB II"
%
```

図 4.6 snmpgetによってMIBの値を得る

標準 MIB(MIB、MIB-II など) の他に、各ベンダーが固有の機器のために独自に拡張を行った MIB が存在し、このような MIB を用いてさまざまな機器を SNMP でコントロールすることが可能になる。 SNMP に

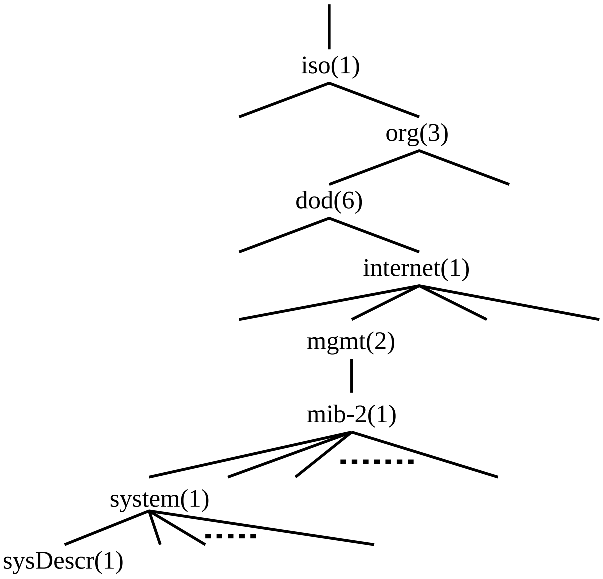


図 4.7 MIB ツリー

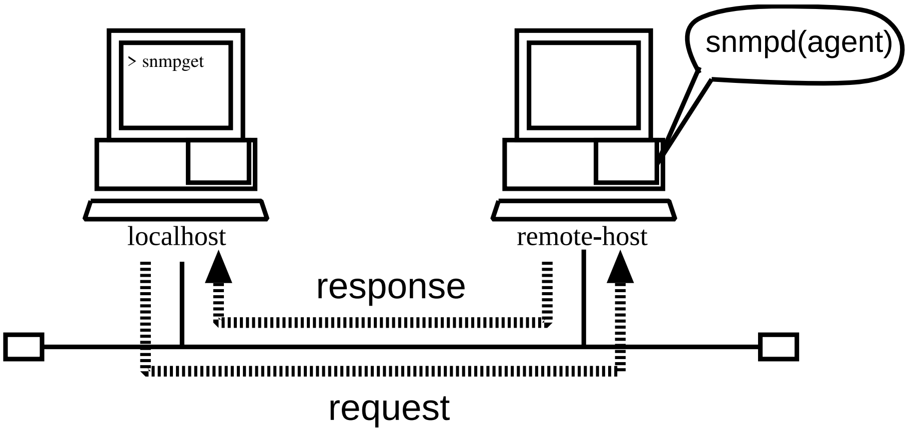


図 4.8 SNMP

よって管理対象の情報収集や状態の変更を行なうためには、管理対象が SNMP をサポートしている必要がある。SNMP の実装にはさまざまなものが存在するが、現在 UNIX で利用できる主な実装を次に示す。

1. CMU-SNMP version 2.1.2
   実装が比較的単純であるため使いやすいことが特徴である。MIB はテキストファイルに ASN.1 に従って記述されたものを利用するため拡張が容易である。エージェントとメッセージ交換を行なう際に、クライアントに指定する MIB 識別子の記述法も自由度が高く利用しやすい。現在は SunOS 4.1.3, BSD/OS 1.1, FreeBSD-1.1.5, FreeBSD-2.0R で動作確認済みである。

2. ISODE-SNMP version 8.0
   OSI の上位層を研究するために開発された ISO 開発環境 (ISODE:ISO Development Environment) を用いて実装されている SNMP パッケージである。 MIB は ASN.1 文法に従って記述されたものを MOSY(Managed Object Syntax-compiler) というコンパイラを用いて、実行時のライブラリが使用するファイルに変換する。このため、テキストファイルを解釈する CMU-SNMP に比べ解析速度が速い。また、GNU の awk を SNMP 対応化したものを使用することができる。SunOS 4.1.3 において動作確認済みである。
3. SONY-SNMP
   NEWS-OS に付属する SNMP パッケージである。NEWS-OS が動作しているワークステーション上にはすでにインストール済みであるためすぐに利用できる。ただし、MIB 識別子の指定法にくせがあり、自由度が低いなど難点も多い。

本論文で実装した管理作業支援系では CMU-SNMP version 2.1.2 を用いている。この CMU-SNMP パッケージに含まれる実装のうち本論文で利用したものを次に示す。

`snmpd`	SNMP のエージェント
`snmptrapd`	SNMP のトラップ[4]を受け付ける。受け付けたトラップを syslog を用いて記録する。
`snmpget`	SNMP エージェントに対して MIB の問い合わせを行なう。

4.3.4 NMW(Network Management Worm)

ネットワークワーム [9] によってネットワーク管理を行なうためのシステムである。ネットワーク上の管理対象上にワームサポートデーモン (wsd) が存在し、命令を与えられたワームがこれらの wsd でを渡り歩きながら管理作業を行なう。つまり、リモートシステム上に NMW システムを構築することができれば、リモートホスト上の情報調査が可能である。

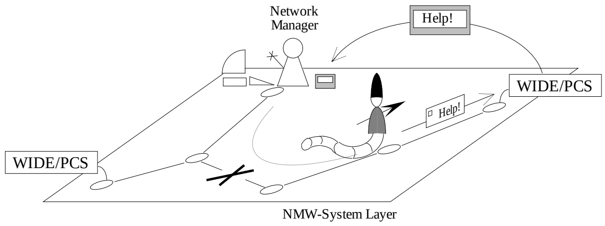


図 4.9 NMW システム

---

[4] 割り込みのこと。管理対象で報告すべき例外事象が起きた時に、管理対象は管理ホストにトラップを送る。

4.4 管理知識

ここでは、管理知識の記述方法について述べる。magP は4.4.1節以降で述べる「管理知識記述文法」に従って記述された管理知識をもとに動作を行なう。現在の実装では、この管理知識はエディタで記述され、テキストファイルとして保存される。

4.4.1 管理知識記述文法

管理作業支援系で扱う管理知識は、管理知識記述文法に従って記述される。例えばローカルシステムの監視作業は、表4.1中の管理コマンドごとに記述できる。この節では、この管理知識記述文法の設計において注意すべき点について述べる。

情報調査で得られる情報は、数値や文字列などさまざま形態をとる。また、複数の情報が1つにまとめられた形で提供される場合(例えば表4.1中のps,netstatなど)や、ファイルの形で出力される場合なども考えられる。そこで、多くの情報を調査するためには、どのような形態の情報でも可能な限り自由に扱える必要がある。

情報の評価方法もさまざまである。システムにとって正常な値が特定の1つの値である場合もあれば複数存在する場合もあり、数値型の情報であれば、正常な状態の候補が連続値である場合、離散値である場合などが考えられる。このように、情報の評価方法も自由に表現できなければならない。

また、管理知識は支援系内部に持つのではなく、外部にデータベースとして蓄積することによって、支援系の利用者である管理者が自由に管理知識を追加できなければならない。さらに、この管理知識の自動生成が容易になるように文法を設計しなければならない。

以上をまとめると管理知識記述文法の設計方針は次のようになる。

1. さまざまな形態の情報を扱えなければならない
2. 情報や分析条件を可能な限り自由に記述できなければならない
3. 支援系の利用者である管理者が知識を自由に追加できなければならない
4. 支援系自体が知識の自動生成を行なえなければならない

そして、このようなことを考慮に入れ設計した、管理知識記述言語に従って記述される管理知識は表4.2の要素によって構成される。この表中のラベルは、実際に管理知識を記述する際に用いられる。各要素の詳細は以降の節や付録Aで述べる。例えば、管理知識の記述例は図A.2である。

4.4.2 管理知識名と管理コマンドに関する記述

この節では、管理知識名と管理コマンドに関する記述方法について述べる。ただし、それぞれの詳しい記述方法は付録Aで述べる。

管理知識名はmagPに管理知識を指定するときに用いる名前である。管理コマンドについては次の3項目について記述しなければならない。

- 管理コマンドの種類

表 4.2 管理知識を構成する要素

構成要素	ラベル
管理知識名	CNAME
管理コマンドの種類	CTYPE
管理コマンドの内容	COMMAND
実行制限時間	TIMEOUT
履歴破棄時間	OBSOLETE
データ区切り記号	DATA_MARK
データフォーマット	DATA_FORMAT
フィールド名定義	DATA_ALIASES
分析規則部	IF, THEN, RETURN, ENDIF
報告書定義部	任意

- 管理コマンドの内容
- 実行制限時間

管理コマンドの種類は4.3.2節で述べたもののうち「シェル上で実行するコマンド」「リモートシェルを用いてリモートホスト上で実行するコマンド」「SNMP を用いた方法」が利用できるので、このうちから選択する。管理コマンドの内容は具体的な内容を記述する。例えば、管理コマンドに「シェル上で実行するコマンド」を選択した場合は、シェル上で実行する際と同様にコマンドを記述する。実行制限時間は、終了しない管理コマンドやあまりにも実行に時間がかかる管理コマンドを、強制的に終了させるために指定する。

4.4.3 管理コマンドの実行結果に関する記述

実行する管理コマンドごとに実行結果の形態はさまざまである。そこで、このようなさまざまな出力形態に対応するために、管理作業支援系で用いた方法について述べる。例として、ディスクの空き状況を調べるために BSD UNIX のコマンド`df` を用いる場合を考える。例えば、`df` の出力は図 4.10のようになる。

```
% df -kt ufs
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0a       19631    14153     4496    76%    /
/dev/sd0e       19695     2216    16494    12%    /var
/dev/sd0g      792109   576662   175841    77%    /usr
/dev/sd0h      112623    62047    44944    58%    /home/anne
%
```

図 4.10 df の出力の例

管理コマンドの出力の分割

管理作業支援系では図4.10のような管理コマンドの出力を「区切り文字」によって文字列に分割して扱っている。区切り文字は、管理者が特に指定しない限り「空白」「タブ」である[5]ため、図4.10の`df`の出力は図4.11のように分割される。ただし、分割されてできる文字列を「"」で囲み、「,」(カンマ) で区切って並べている。このように分割されてできる各文字列をフィールドと呼ぶ。管理者はディスクの空き状況を調べるために、ファイルシステム名 (`Filesystem`) と使用率 (`Capacity`) に注目すれば良いとすれば、図4.11では各行の第1フィールドのファイルシステム名、第5フィールドの使用率に注目することになる。しかし、使用率のフィールドには「%」記号が付与されており、このままでは数値として扱えない形になっている。また、図4.10の出力の1行目には「項目名」が出力されており、最後の項目名は「`Mounted on`」である。しかし、4.11ではこの「`Mounted on`」が2つのフィールドに分割されてしまっている。

```
"Filesystem", "1K-blocks", "Used",   "Avail",  "Capacity", "Mounted",   "on"
"/dev/wd0a",  "19631",     "14153",  "4496",   "76%",      "/"
"/dev/sd0e",  "19695",     "2216",   "16494",  "12%",      "/var"
"/dev/sd0g",  "792109",    "576662", "175841", "77%",      "/usr"
"/dev/sd0h",  "112623",    "62047",  "44944",  "58%",      "/home/anne"
```

図4.11 dfの出力をスペースとタブで分割した場合

以上のように、管理コマンドの出力を単純に「区切り文字」で分割しただけでは意図した通りにならない場合があるため、細かい出力形式の指定のために「データフォーマット」を用意する。このデータフォーマットの例が図4.12である[6]。このデータフォーマット中で使用される記号の意味を表A.3に示す。例えば、1行目の第5フィールドまでは「区切り文字」で分割することできるので「`f`」を指定し、第6フィールドは正規表現で定義している。また、2行目以降の第5フィールドは「%」記号を除いた形で取り出したいので、正規表現によって定義する。正規表現によって定義されたフィールドは、正規表現中の値として取り出したい部分に対応する部分を「(,)」によって囲む必要がある。「(,)」によって囲まない場合は値を取り出すことはできない。

```
f * "(Mounted .*)"
f * "([0-9]*)%" f
*
.
```

図4.12 データフォーマットの例

また、「*」記号は直前のフィールド定義の繰り返しを意味するが、第1フィールドとして使用した場合と、第2フィールド以降で使用した場合でその意味が異なる。第1フィールドとして使用した場合 (図4.12の定義中では3行目) は直前の行の繰り返しを意味し、第2フィールド以降で使用した場合は直前のフィー

---

[5] 必要であれば、管理知識を記述する際に、この区切り文字を指定することができる。

[6] ただし、`df`の出力はファイルシステムの名前が長い場合、ファイルシステム欄が2行に分割され、このフォーマットでは扱うことができない。

ルド定義の繰り返しを意味する。

図 4.12のデータフォーマットを用いて、図 4.10の出力を分割すると、図 4.13のようになる。「使用率(`Capacity`)」の値から「%」記号が除去され、1 行目の第 6 フィールドが「`Mounted on`」となったことが分かる。

```
"Filesystem", "1K-blocks", "Used",    "Avail",  "Capacity", "Mounted on"
"/dev/wd0a",  "19631",     "14153",  "4496",   "76",       "/"
"/dev/sd0e",  "19695",     "2216",   "16494",  "12",       "/var"
"/dev/sd0g",  "792109",    "576662", "175841", "77",       "/usr"
"/dev/sd0h",  "112623",    "62047",  "44944",  "58",       "/home/anne"
```

図 4.13 df の出力の分割例

フィールド名

フィールドの値を分析規則部などで参照するためには、各フィールドに名前をつけなければならない。これをフィールド名定義と呼び、フィールドにつけられた名前をフィールド名と呼ぶ。

フィールド名の定義は次のよう記述する。

フィールド名　　整数値　整数値

フィールド名に続く 2 つの整数値はフィールドの位置を表わし、0 以上の値か「*」記号である。例えば、図 4.10の出力を図 4.12のデータフォーマットによって図 4.13のように分割した場合、

```
capacity    1 4
```

のように記述すると 2 行目の第 5 フィールドの値を「`capacity`」という名前で参照できる。また、この場合の「`capacity`」値は「76」となる (図 4.4.3)。ただし、図 4.4.3中の各四角形はフィールド 1 つ分を意味し、斜線のフィールドが「`capacity`」で参照されるフィールドである。

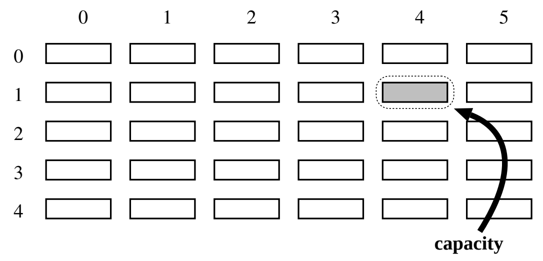


図 4.14 フィールド名によるフィールドの参照

行や列に全体にフィールド名をつけることもできる (図 4.15、4.15)。縦 1 列に並ぶ各行の第 5 フィールドを参照するためのフィールド名の定義は次のようになる。

```
capacity * 4
```

「*」記号は任意の整数を意味し、フィールド名「`capacity`」は一種の配列のように働く (ただし、「*」をフィールド名の定義 1 つに 2 つ使用することは許されない)。例えば、2 行目の第 5 フィールドの値を参照

する場合は「`capacity[1]`」と記述し、その値は「76」となる。同様に、1 行目を参照する場合は次のように定義する。

```
keyword 0 *
```

この場合は、「`keyword[3]`」と記述すると 1 行目の第 4 フィールドの値が参照され、値は「Capacity」となる。表 A.4のように記述すれば、一部分を取り出すこともできる。

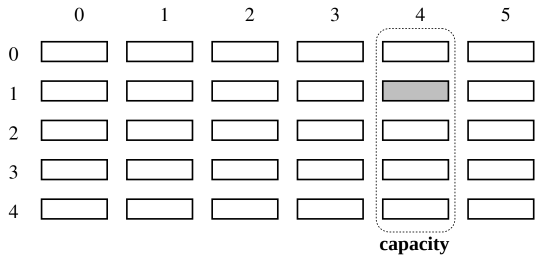


図 4.15 縦 1 列に並ぶフィールドの参照

破線で囲まれた部分に`capacity` というフィールド名をつけた場合、`capacity[1]` で斜線のフィールドの値が参照できる。

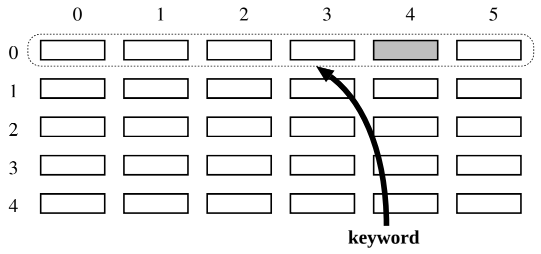


図 4.16 横 1 行に並ぶフィールドの参照

1 行目の破線で囲まれた部分に`keyword` というフィールド名をつけた場合、斜線のフィールドの値は`keyword[3]` で参照できる。

4.4.4 分析規則部の記述

管理知識の記述において分析規則が記述されている部分を「分析規則部」と呼ぶ。magP はこの分析規則部に従って管理コマンドの実行結果を分析する。分析規則部は構造の構造は次のようになる (図 4.17)。

- 分析規則部は、1 個以上の分析規則によって構成される
- 分析規則は、1 個以上の条件部と 1 つの行動部で構成される
  - 条件部は 0 個以上の条件式で構成される
  - 行動部には「報告」「対策」「再調査」「関数`call()`による呼び出しの戻り値」ついて記述される。

これを拡張 BNF[7]で定義すると次のようになる。

分析規則部 ::= 分析規則 { 分析規則 }
分析規則 ::= 条件部 { 条件部 } 行動部
条件部 ::= { 条件式 }

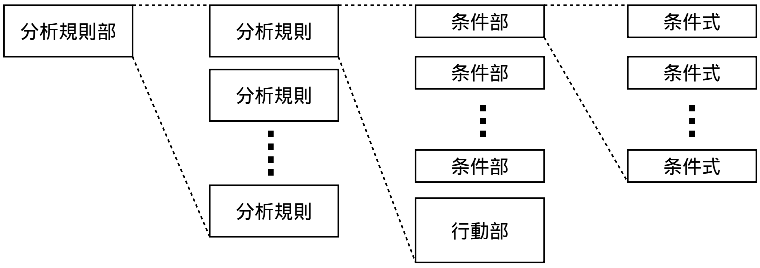


図 4.17 分析規則部の構造

また、この分析規則部はつぎのルールに従って評価される。

- すべての分析規則の評価を行なう
- 分析規則に含まれる条件部のうちのどれかが真のとき、その行動部が実行される
- 条件部のすべての条件式が満たされたとき、その条件部は真となる。

分析規則部に複数の分析規則が含まれる場合はすべてを評価する。分析規則部が持つ 1 個以上の条件部のうち 1 つでも満たされるものがれば、その分析規則の行動部が実行される。また、ある条件部内の条件式がすべて満たされるとき、その条件部は満たされていると考える。ただし、条件式を 1 つも持たない条件部は常に満たされているとする。

管理知識が関数`call()`によって呼び出されたものであり、その管理知識の分析規則部が 2 つ以上の分析規則を持つ場合は、1 つ目の分析規則のみ評価を行ない、行動部に記述されている戻り値がその評価結果に従って呼び出しもとに返される。そして、2 つ目以降の分析規則は評価しない。

図 4.18が、このような構造を持った分析規則部の例である。この 31 行で構成される分析規則部の構造は以下のようになる。ただし、図 4.18は、分析規則部の他に報告書定義部も含まれている。

第 1 行 ~ 第 17 行 分析規則部

---

[7] {a} は a の 0 個以上の繰り返しを表す。

第 1 行 ~ 第 16 行	分析規則
第 1 行 ~ 第 3 行、第 4 行 ~ 第 6 行	条件部
第 2 行、第 5 行	条件式
第 7 行 ~ 第 16 行	行動部
第 19 行 ~ 第 31 行	報告書定義部

この例では、2 つの分析規則が定義されている。1 つ目の分析規則の条件部は 2 つの条件を含んでいる。また、1 つの条件には複数行記述することができるが、全て満たされたければならない。条件部は「.」(ピリオド) によって記述の終了を示す。

条件式は「フィールド名」「関数」「演算子」「定数」などによって構成される。フィールド名は管理コマンドの実行結果 (出力) を参照するために用いる。

分析規則でフィールドの値を参照するためには「$」記号に続いてフィールド名を記述する。関数は、履歴情報との比較や他の管理知識の呼出しなどに用いる。履歴情報や関数については 4.4.5節、4.4.6節や付録Aで詳しく述べる。ただし、使用できる関数は表 A.6の通りである。分析規則で関数を使用するためには、「&」記号に続いて関数名を記述する。また、本論文の支援系の実装は perl を用いて行なったため、perl で使用できる比較演算子のほとんどがそのまま使用できる。

```
 1: IF:
 2:                $capacity[1..*] > 90
 3:                .
 4: IF:
 5:                &percentage($capacity[1..*]) >= 10
 6:                .
 7: THEN:
 8:                INFORM:
 9:                        WITH:    MAIL
10:                        TO:      root
11:                        TEXT:    WARNING_MAIL
12:                RESEARCH:
13:                        CNAME:          check_disk
14:                COUNTER:
15:                        COUNTER-CNAME:  clean_core
16:                .
17: ENDIF:
18:
19: WARNING_MAIL:
20:
21: ファイルシステムの使用率に異常が発見されました
22: ので報告します
23:
24: 各ファイルシステムの使用率は以下の通りです。
25: &filesystem[1..*]: &capacity[1..*] %
26:
27: 各ファイルシステムの使用率の前回調査時と比較し
28: た変化率は以下の通りです。
29: &filesystem[1..*]: &history(&capacity[1..*]) %
30:
31: .
```

図 4.18 分析規則部の定義例

行動部の記述

この節では行動部の記述について述べる。行動部は分析規則部に含まれ、分析規則の評価結果に従って実行される。行動部には「次の作業」または「関数`call()`による呼び出しの戻り値」が記述される。「次の作業」は報告、対策および再調査のうちのいずれかである。「関数`call()`の戻り値」は条件部が満たされた時と満たされない時の戻り値が記述される。図4.18の分析規則部の定義例では、「次の作業」が行動部である第7行～第16行に記述されている。例えば、電子メールによる報告の記述は次の部分である。

```
 8:                  INFORM:
 9:                          WITH:    MAIL
10:                          TO:      root
11:                          TEXT:    WARNING_MAIL
```

「`WITH:`」で報告方法、「`TO:`」で報告先、「`TEXT:`」で報告事項(この場合は報告書定義のラベル名)が記述されている。また、「対策」「再調査」についても次のように記述されている。

```
12:                  RESEARCH:
13:                          CNAME:          check_disk
14:                  COUNTER:
15:                          COUNTER-CNAME:  clean_core
16:                  .
```

ここでは、「`RESEARCH:`」で再調査のための作業を表わす管理知識を指定し、「`COUNTER:`」で対策のための作業を表わす管理知識を指定している。

次に、「関数`call()`による呼び出しの戻り値」は、図4.18の分析規則部の定義例では定義されていないが、後で述べる実用例3の用いられる管理知識`get_temperature`ではこれが定義されている。例えば、この管理知識`get_temperature`の定義のうち分析規則部は次のようになる。

```
1:  IF:
2:                 $temperature < 30
3:                 .
4:  THEN:
5:                  .
6:  RETURN:
7:                 "OK"
8:                 "FAULT"
9:  ENDIF:
```

「関数`call()`による呼び出しの戻り値」が定義されているのは、第6行目～第8行目である。定義には2つの値が記述され、この例の場合は、条件部つまり条件式「`$temperature < 30`」が満たさた場合、管理知識`get_temperature`の呼び出しもとには1つ目の値「`"OK"`」が返される。また、満たされない場合は2つ目の「`"FAULT"`」が返される。また、関数`call()`によって管理知識が呼び出された場合、呼び出された管理知識内の分析規則部は2つ以上の分析規則を含んでいても1つ目の分析規則のみ評価される。そのため、この「関数`call()`による呼び出しの戻り値」の定義は1つ目の分析規則の行動部に記述され、それ以外の記述は無効となる。

4.4.5 履歴情報

管理作業おいてシステムの状態変化を検出するためには、収集した情報の履歴が必要である。本論文の管理作業支援系においても、調査結果を履歴として記録し分析作業で参照している。この履歴情報を記録する上で注意すべき点は次の通りである。

1. 調査結果は全て記録されなければならない
2. あまりに古く役に立たない情報は除去されなければならない
3. 効率よく検索できなければならない

調査結果はいつどのような状況で必要になるか予測できないため、確実に不必要であると考えられる情報以外は、できる限り記録保存しなければならない。また、履歴の検索時に古すぎて無意味であると考えられる情報は検索の対象から除外する必要がある。さらに、履歴の検索が頻繁であるほど、検索の効率が重要となることも明らかである。そこで、本論文では次のように実装を行なった。

1. 管理コマンドの実行結果はすべて記録する
2. 記録事項は、管理知識名、管理知識指定時の引数、調査時刻および結果出力の 4 項目とする
3. テキストファイルとして記録する
4. 検索効率については今後の課題として保留する

例えば、管理知識`disk_free_space` を引数なしで指定し、図 4.10のような結果が得られた場合には、図 4.19のように記録が履歴ファイルに書き込まれる。ただし、時刻は 1970 年 1 月 1 日 0:00 からの秒数である。

```
CNAME:  disk_free_space
CARGS:
TIME:   790671524
DATA:
Filesystem  1K-blocks     Used    Avail Capacity  Mounted on
/dev/wd0a       19631    14153     4496    76%    /
/dev/sd0e       19695     2216    16494    12%    /var
/dev/sd0g      792109   576662   175841    77%    /usr
/dev/sd0h      112623    62047    44944    58%    /home/anne
/dev/wd0h      329135    77588   235090    25%    /usr/local/source
.
```

図 4.19 履歴に書き込まれる記録の例

4.4.6 関数

関数は、管理情報の分析の際にフィールドの値の参照や演算以外の分析規則の記述に用いられる。その一覧が表 A.6である。条件式中で関数を記述する時は、「&」記号に続いて関数名を記述する。

change 関数

関数`change()` はフィールド名 1 つを引数とし、そのフィールドの値の変化量を求める。例えば、

```
&change($capacity[*]) >= 90
```

と記述された条件式は、図4.20のように置換・展開される。まず、関数`change()`は履歴を検索し、履歴上の指定されたフィールド名の最新の値を求める。履歴の検索に失敗した場合の変化量は0とする。変化量は次のように計算される。

$$\left| \text{現在の値} - \text{履歴の値} \right|$$

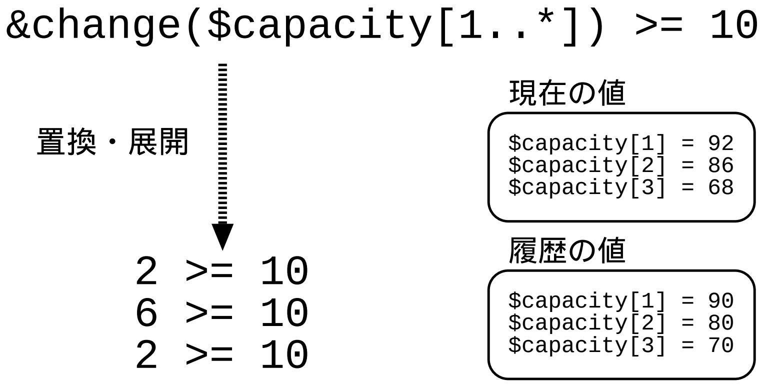


図4.20 関数`change()`を用いた条件式の置換・展開の様子

percentage 関数

関数`percentage()`は、別名1つを引数とし変化率を百分率で求める。実際には、履歴上の指定されたフィールドの最新の値を求め、この値と現在のフィールドの値の比較を行ない変化率を求める。また、履歴の検索に失敗した場合は変化率は0とする。この関数は、5.1節の実用例1で用いている。

$$\left| \frac{\text{現在の値} - \text{履歴の値}}{\text{履歴の値}} \right| \times 100$$

history 関数

関数`history()`は管理知識名、フィールド名、管理知識に対する引数の3つを指定し履歴上の値を得る。履歴検索には、指定された管理知識名と管理知識に対する引数を用いる。履歴の検索に失敗した場合は値はなしとする。この関数は5.2節の実用例2で用いている。

history_time 関数

関数`history_time()`は管理知識名、管理知識に対する引数の2つを指定し用いる。この関数は履歴を検索し、指定した管理知識を指定した引数で呼び出した最も最近の時刻を得る。この関数は5.2節の実用例2で用いている。

call 関数

関数`call()`は、管理知識名、管理知識に対する引数の2つを指定し用いられ、指定された管理知識を呼び出す。呼び出された管理知識は管理知識データベースから取り出され、支援系はこれに従って動作を行なう。ただし、呼び出された管理知識の分析規則部に分析規則が2個以上存在しても2番目以降は無効となり、1番目のみ評価される。分析規則の評価後は、その評価結果に従って`RETURN:`部の値を戻り値として呼び出し元に戻る。この関数は、5.3節の実用例3で用いている。

shell 関数

関数`shell()`は、指定されたコマンドをシェル上で実行し、その実行結果を値として返す。この関数は、5.2節の実用例 2 で報告書定義部の記述に用いている。

4.4.7 報告書定義部の記述

分析の結果、報告を行なうことが決定した場合、次の 3 項目を決定する必要がある。

- 報告先
- 報告方法
- 報告事項

管理作業に関する情報を報告する限り、報告先には管理者が必ず含まれる。また、障害の発生に伴う注意や警告、システムの変更などによる環境の変化などは、管理者だけでなくシステムの利用者に対しても必要に応じて報告が行うべきである。

報告方法は「電子メール」「ページャー」「FAX」などが考えられる。現在の実装では「電子メール」「ページャー」を利用できる。この「報告先」「報告事項」は分析規則部の行動部に記述する。

次に報告、報告事項を決定する。報告事項は、管理知識の報告書定義部に記述させる。管理作業支援系は、報告書定義部のデータから情報調査や分析規則部の評価結果をもとに報告書を生成する。報告書定義部では分析規則部と同様にフィールド名、関数を用いることができる。そして、これらのフィールド名や関数が適当な値で置換されることで報告書が生成される。また、「フィールド名」は「$」記号を先頭に書くことで記述されるが、これにくわえて、報告書定義部では「%」記号を先頭に書いて記述することが特別な意味を持っている。この違いについては後述する。

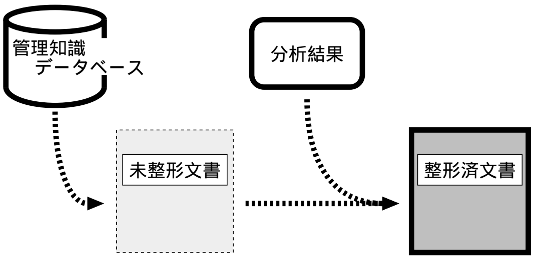


図 4.21 報告書生成の流れ

まず、報告書の生成は次のような流れで行なわれる。

1. 管理コマンドデータベースから報告書の元データを得る
2. 「%」記号によって指定されるフィールド名をその値で置換する
3. 「$」記号によって指定されるフィールド名をその値で置換する
4. 関数を評価し、その値で置換する

フィールド名が値を 1 つしか示さないような記述の場合、「$」「%」には特に違いはない。しかし、フィールド名が複数の値を示すように記述される場合は次のように扱われる。

`$name[...]`	要素を空白1つで連結した形の文字列で置換する
`%name[...]`	各要素で置換を行ない要素数分行を生成する

例えば、図4.22は、情報調査と分析で得られた結果をもとに、置換が行なわれる様子を示している。まず、「`$capacity[1..*]`」と記述された部分は、「`$`」記号で別名が指定されているので、要素を空白で連結した文字列で置換が行なわれている。また、「`%filesystem[1..*]`」は「`%`」記号でフィールド名が記述されているので、要素数分、つまり3行複製され、それぞれ各要素で置換が行なわれている。ただしこの例のように、同一の行で「`%`」記号にを用いてフィールド名が記述される場合、置換作業は最初のフィールド名の記述に依存するという規則に基づいている。

以上のようにして生成された報告書が、実際に管理者などへ報告される。

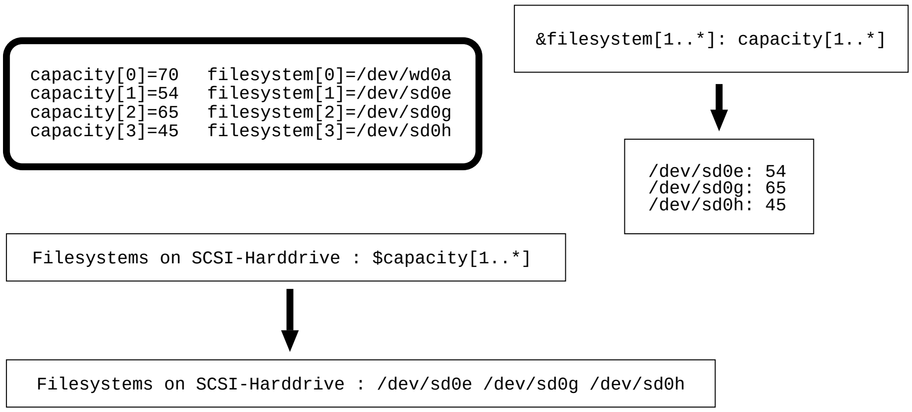


図4.22 フィールド名の置換

第 5 章

実用例

ここでは、管理作業支援系の実行部である magP の実用例について述べる。

5.1 実用例 1

この例の内容は次の通りである。

内容	ディスクの使用率が以下の条件を満たしているか監視を行なう
条件 1	使用率が 90%以下
条件 2	使用率の前回調査時から変化率が 10%以下

はじめに、管理知識データベースに必要となる管理知識が登録しなければならない。ディスクの使用率の監視は BSD UNIX では`df` コマンドを用いる。監視の対象となるディスクはローカルホストに存在するものと仮定すると、このコマンドをシェル上で実際に実行した結果が図 4.10である。この結果を見る限り、この実用例で必要となる情報は十分に`df` によって得られるため、この`df` を用いて管理知識を定義する。

管理知識のおおまかな内容は図 5.1のようになる。`df` コマンドの出力結果に対するデータフォーマットの定義は、すでに 4.4.3で述べており、図 4.12である。また、フィールド名は`df` コマンドの出力の列ごとに、左から`filesystem`、`total`、`used`、`avail`、`capacity`、`mounted` と定義する。

次に、分析規則部ついて述べる。この例で必要となるフィールドは 「使用率 (`capacity`)」である。また、報告書定義部では 「ファイルシステム名 (`filesystem`)」が必要となる。

まず、「使用率が 90%以下である」という条件はディスクが正常な状態であるための条件であり、ここでは異常時に報告を行なわせたいので、使用率が 90%を越えているかを検出することにする。この場合、条件式は次のようになる。

```
$capacity[1..*] > 90
```

図 4.10から使用率の値は 2 行目以降に出力されていることが分かる。フィールド名`capacity` は各行の第 5 フィールドを参照しており、第 1 要素は項目名 「`Capacity`」であるので、参照値のうち必要となるのは第 2 要素から最終要素までである。そこで、

```
$capacity[1..*]
```

と記述している。

「使用率の前回調査時からの変化率が 10%以上」であることを検出する条件式を記述するには、前回調査時の値を得るために履歴情報を用いる。履歴情報を得るために用意されている関数のうち、特に同一の管理知識による作業の結果との比較では、関数 「`change()`」「`percentage()`」を用いることができる。条件式は次のようになる。

```
&percentage($capacity[1..*]) >= 10
```

関数`percentage()`はフィールド名を引数として使用し、フィールドの現在の値と前回調査時の値の変化率を求めることができる。

次に、条件式が満たされた場合に実行される実行部を記述する。この例では実行部は「報告」のみとし、報告先を管理者 (root)、報告方法を電子メールとする。報告書では、少なくとも「ファイルシステム名」「使用率」を報告する必要がある。

以上をまとめると、この実用例で用いる管理知識`disk_free_space`は図 5.1のようになる。図 5.2は magP が管理知識`disk_free_space`に従って動作し、ディスクの使用量の変化率が 10%を越えていることを検知した場合に管理者に届く電子メールである。

表 5.1 実用例 1 で管理コマンドデータベースに登録するコマンドの内容

項目名	内容
コマンド名	disk_free_space
コマンド型	shell 型
コマンド内容	`df -kt ufs`
実行制限時間	特に制限しない
履歴破棄時間	1 時間以上経過したものは破棄する
出力区切り文字	空白 (スペース) とタブ
データ形式	欄外
別名定義	欄外
分析規則部	欄外
報告書原稿	欄外

5.2 実用例 2

この例では、経路情報の変化の検出を行なった。必要となる BSD UNIX のコマンドは`traceroute`である。`traceroute`は指定したインターネット上のホストにパケットを送信するとき、このパケットの通る経路を出力するためのコマンドである。例えば、図 5.3がこの`traceroute`の実行例である。これ図によると、各行に「ホスト名」「IP アドレス」「RTT[1]」が出力されていることが分かる。ただし、IP アドレスからホスト名を求めることができない場合、ホスト名の部分は IP アドレスが出力されてしまうため、IP アドレスの部分を用いて経路情報の変化を検出することにする。また、経路情報の変化を検出するためには、履歴を検索し以前の調査結果と比較を行なう必要がある。そこで、関数`history()`を用いて前回の調査結果を得る。条件式は次のようになる。

```
&history(check_routing, $ipaddr[*], "$argv[1]") ne $ipaddr[*]
```

ただし、`$argv[1]`は管理コマンド実行時の第 1 引数である。また、管理知識の内容は図 5.4、定義は図 5.6 のようになる。

---

[1] Round Trip Time。ネットワーク通信において、1つのパケットが始点から終点まで行き、再び始点へ戻って来るまでに要する時間。この時間には、終点で始点からのメッセージを処理し、応答を生成する時間も含まれている。

```
CNAME:
                disk_free_space
CTYPE:
                shell
COMMAND:
                df -kt ufs
TIMEOUT:
                0
OBSOLETE:
                01:00:00
DATA_MARK:
                "\t "
DATA_TYPE:
                f * "(Mounted .*)"
                f * "([0-9]*)%" f
                *
                .
DATA_ALIASES:
                filesystem    * 0
                total         * 1
                used          * 2
                avail         * 3
                capacity      * 4
                mounted       * 5
                .
IF:
                $capacity[1..*] > 90
                .
IF:
                &percentage($capacity[1..*]) >= 10
                .
THEN:
                INFORM:
                        WITH:    MAIL
                        TO:      root
                        TEXT:    WARNING_MAIL
                .
ENDIF:

WARNING_MAIL:
  ファイルシステムの使用率に異常が発見されました
  ので報告します

  各ファイルシステムの使用率は以下の通りです。
  %filesystem[1..*]: %capacity[1..*] %

  各ファイルシステムの使用率の前回調査時と比較し
  た変化率は以下の通りです。
  %filesystem[1..*]: %history(&capacity[1..*]) %
.
```

図 5.1 実用例 1 で管理知識データベースに登録する管理知識

この例では次のようなことを報告する必要がある。

- 始点ホストおよび終点ホスト
- 過去の経路およびその時刻
- 現在の経路およびその時刻

「終点ホスト」は、管理コマンドの引数として与えられているはずなので`$argv[1]`の値を使う。「始点ホスト」はローカルホストであるので、`hostname`コマンドを利用するため関数`shell()`を用いる。「過去の経路とその時刻」は、関数`history()`および`history_time()`によって得る。最後に「現在の経路」は

```
Subject: Check Filesystems
From: root (magP)
To: root
Date: Fri, 3 Feb 1995 07:27:36 0900 +

  ファイルシステムの使用率に異常が発見されました
  ので報告します

  各ファイルシステムの使用率は以下の通りです。
  /dev/wd0a: 80 %
  /dev/sd0e: 14 %
  /dev/sd0g: 76 %
  /dev/sd0h: 70 %
  /dev/wd0h: 25 %

  各ファイルシステムの使用率の前回調査時と比較し
  た変化率は以下の通りです。
  /dev/wd0a: 0 %
  /dev/sd0e: 53 %
  /dev/sd0g: 0 %
  /dev/sd0h: 0 %
  /dev/wd0h: 0 %
```

図 5.2 実用例 1 で報告される電子メールの例

```
% traceroute train-gw.noc.titech.ac.jp
traceroute to train-gw.noc.titech.ac.jp (131.112.246.2), 30 hops max, 40 byte packets
 1  nsk.is.titech.ac.jp (131.112.40.126)  1.163 ms  0.807 ms  0.780 ms
 2  nsa.is.titech.ac.jp (131.112.40.1)  2.147 ms  2.741 ms  2.273 ms
 3  131.112.248.7 (131.112.248.7)  4.672 ms  6.217 ms  5.052 ms
 4  train-gw.noc.titech.ac.jp (131.112.246.2)  17.211 ms *  13.702 ms
%
```

図 5.3 traceroute によってパケットの経路を調べる

`$ipaddr[*]` の値、「現在の時刻」は「始点ホスト」と同様に関数`shell()` によって`date` コマンドを用いる。以上をもとに報告書定義部の記述を行なった。

この例で記述した管理知識を指定し magP を動作させ、経路情報の変化を検知した場合に管理者に送られる電子メールの例が図 5.5である。

項目名	内容
コマンド名	check_routing
コマンド型	shell 型
コマンド内容	`traceroute $argv[1]`
実行制限時間	10 秒で実行を打ち切る
履歴破棄時間	15 分以上経過したものは破棄する
出力区切り文字	空白 (スペース) とタブ
データ形式	図 5.6参照
別名定義	図 5.6参照
分析規則部	図 5.6参照
報告書原稿	図 5.6参照

図 5.4 実用例 2 で管理コマンドデータベースに登録するコマンドの内容

```
Subject: Check Routing Information
From: Yoshiaki Nakajima <yoshiaki>
To: root
Date: Sat, 4 Feb 1995 00:37:31 +0900

ローカルホスト (anne) から
ホスト (msa) までの経路が
以下のように変化しました。

以前の経路 (時刻 Sat Feb  4 00:36:29 1995)
1: nsk.is.titech.ac.jp(131.112.40.126)
2: nisi7-gw.noc.titech.ac.jp(131.112.40.62)
3: fddi-gw1.noc.titech.ac.jp(131.112.1.68)
4: fddi-gw2.noc.titech.ac.jp(131.112.253.12)
5: minami1-gw.noc.titech.ac.jp(131.112.1.136)
6: msa.is.titech.ac.jp(131.112.52.1)

現在の経路 (時刻 Sat Feb  4 00:37:31 JST 1995)
1:  nsk.is.titech.ac.jp  (131.112.40.126)
2:  nisi7-gw.noc.titech.ac.jp  (131.112.40.62)
3:  fddi-gw1.noc.titech.ac.jp  (131.112.1.68)
4:  fddi-gw2.noc.titech.ac.jp  (131.112.253.12)
5:  minami1-gw.noc.titech.ac.jp  (131.112.1.136)
6:  fddi-gw2.noc.titech.ac.jp  (131.112.253.12)
```

図 5.5 実用例 2 で報告される電子メールの例

```
CNAME:
                check_routing
CTYPE:
                shell
COMMAND:
                traceroute $argv[1]
TIMEOUT:
                10
OBSOLETE:
                00:15:00
DATA_MARK:
                "\t "
DATA_TYPE:
                f *
                f f "\(([0-9\.]+)\)" f *
                *
                .
DATA_ALIASES:
                indexm     * 0
                hostname   * 1
                ipaddr     * 2
                .
IF:
                &history(check_routing, $ipaddr[*], "$argv[1]") ne $ipaddr[*]
                .
THEN:
                INFORM:
                        WITH:    MAIL
                        TO:      root
                        TEXT:    WARNING_MAIL
                .
ENDIF:
                .
WARNING_MAIL:

ローカルホスト (&shell(hostname)) から
ホスト ($argv[1]) までの経路が
以下のように変化しました。

以前の経路 (&history_time(check_routing, "$argv[1]"))
&history(check_routing, %index[*], "$argv[1]"): \
&history(check_routing, %hostname[*], "$argv[1]") \
(&history(check_routing, %ipaddr[*], "$argv[1]"))

現在の経路 (時刻&shell(date))
%index[*]: %hostname[*](%ipaddr[*])

.
```

図 5.6 実用例 2 で用いる管理知識の定義

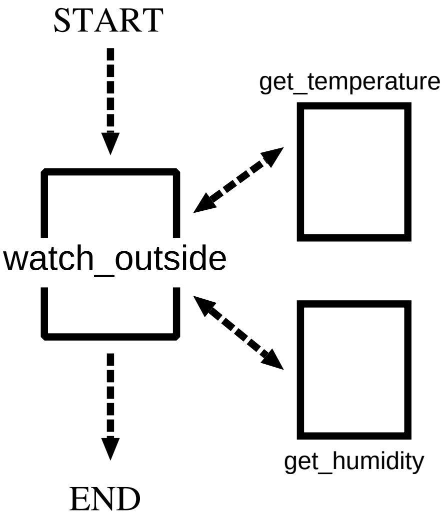
5.3 実用例 3

システム管理において、温度や湿度など、計算機の物理的な外部環境の変化がシステムに及ぼす影響を無視することはできない。そのため、このような外部環境を監視あるいは操作する手段が必要になってくる。最近はこのような目的に利用することができる機器がある。

APC UPS は無停電電源装置であり以下のような特徴を持つため、計算機の外部環境や電源周辺の監視を行なうことができる。

- SNMP を用いて電池内部の状態を調査あるいは操作することができる
- 外部の温度・湿度の計測ができる
- 4 つのドライ接点をセンサーやその他に利用することができる

この例では APC UPS を用いて温度・湿度を監視し、異常な値が検出された場合に報告を行なう。そこで、以下のように 3 つの管理知識を定義した。



`get_temperature`	温度の測定および判定作業を表わす管理知識
`get_humidity`	湿度を測定および判定作業を表わす管理知識
`watch_outside`	上記の 2 つの管理知識よる作業で得られる結果をまとめ、報告を行なう作業を表わす管理知識

`get_temperature` と`get_humidity` は、関数`call()` によって`watch_outside` から呼び出される管理知識である。これらは SNMP を用いて UPS の温度や湿度の測定を行ない、正常な測定値が得られた場合は`"OK"`、異常値の場合は`"FAULT"`を返す。そこで、`watch_outside` ではこれらの結果を次のような条件式で判定する。(ただし、紙面の都合上 1 行を途中で折り返している)

```
( &call(get_temperature, "$argv[1]") eq "FAULT" ) ||
                        ( &call(get_humidity, "$argv[1]") eq "FAULT" )
```

`get_temperature` と`get_humidity` はほぼ同様に記述されるので、`get_temperature` の定義のみ図 5.7 に示す (ただし、この図では紙面の都合上、MIB 識別子の記述を途中で折り返している)。この 2 つの管理知識は関数`call()` によって呼び出されるので、`RETURN` 部を定義して戻り値を設定している。

次に、SNMP による管理知識を定義するためには、次の 3 つを決定しなければならない。

1. エージェントホスト名
2. コミュニティ名
3. MIB 識別子

エージェントホスト名は、管理知識指定時の引数として与えられることにすれば`$argv[1]` で良い。APC UPS ではコミュニティ名`public` を用いることによって温度・湿度の測定値を得ることができる。また、MIB 識別子は、

温度: `.iso.org.dod.internet.private.enterprises.apc.1.1.2.1.1.0`
湿度: `.iso.org.dod.internet.private.enterprises.apc.1.1.2.1.2.0`

である。以上より、管理知識の定義内で「コマンドの内容」(ラベルは`COMMAND:`) には、次のように記述する。

```
$argv[1] public .iso.org.dod.internet.private.enterprises.apc.1.1.2.1.1.0
```

このように、SNMP ではコマンドの内容を記述する際に 3 つ要素を並べ、先頭から「エージェントホスト」「コミュニティ名」「MIB 識別子」を記述する。

5.4 実用例 4

システムの運用中にネットワーク上のホストと通信ができない状態になることがある。このようなとき、管理者が行なう原因の分析と同様の作業を、本論文で実装した支援系に行なわせる。

まず、一般にネットワーク上のリモートホストと通信ができない場合に考えられる原因を整理する。そのために、原因の所在を「ローカルホスト」「ネットワーク上」「通信対象のホスト」に分け、以下のように考えた。

- ping によって、ホスト A からホスト B と通信できないことが分かった。そこで、以下のような問題点を考えながら分析を行なう。図 5.8におおまかな分析の流れを示す。
  1. ホスト A の問題
     (a) ホスト B の IP アドレスが引けていない
     (b) ローカルホストのネットワークインタフェースに異常がある
  2. ネットワークの問題
     (a) ネットワークのトラフィックが高い
     (b) 経路情報が誤っている
     (c) 中継ホストの障害
  3. ホスト B の問題
     (a) ホスト B の負荷が高い
     (b) ホスト B のネットワークインタフェースに異常がある

これをもとに、人間が行なう管理作業ではどのような流れで調査・分析を行なわれるか考え整理したものが図 5.8であり、記述した管理知識名の一覧が表 5.2 である。また、これらの管理知識が図 5.8のどの部分に対応しているのかを示したものが、図 5.9である。例えば、管理知識`Ping` では、調査対象がホスト名で指定されていれば、正しくホスト名から IP アドレスを引くことができているかを確かめるために管理知識`CheckName` を呼び出す。また、調査対象が IP アドレスで指定されていれば、ネットワークインタフェースを調査するために管理知識`CheckIF` による作業へと移る。

```
CNAME:
               get_temperature
CTYPE:
               snmp
COMMAND:
               $argv[1] public .iso.org.dod.internet.private.
                                enterprises.apc.1.1.2.1.1.0
TIMEOUT:
               0
OBSOLETE:
               00:05:00
DATA_MARK:
               "\t "
DATA_TYPE:
               f *
               .
DATA_ALIASES:
               temperature      0 2
               .
IF:
               $temperature < 30
               .
THEN:
               .
RETURN:
               "OK"
               "FAULT"
ENDIF:
```

図 5.7 温度を測定し判定するための管理知識の定義

管理知識名	内容
Ping	通信を試みる
CheckName	名前の登録状況を調べる
CheckIF	ネットワークインタフェースを調べる
CheckNet	ネットワークの状況を調べる
RePing	再度通信を試みる
CheckRoute	経路情報を調べる

表 5.2 実用例 4 で必要となる管理知識

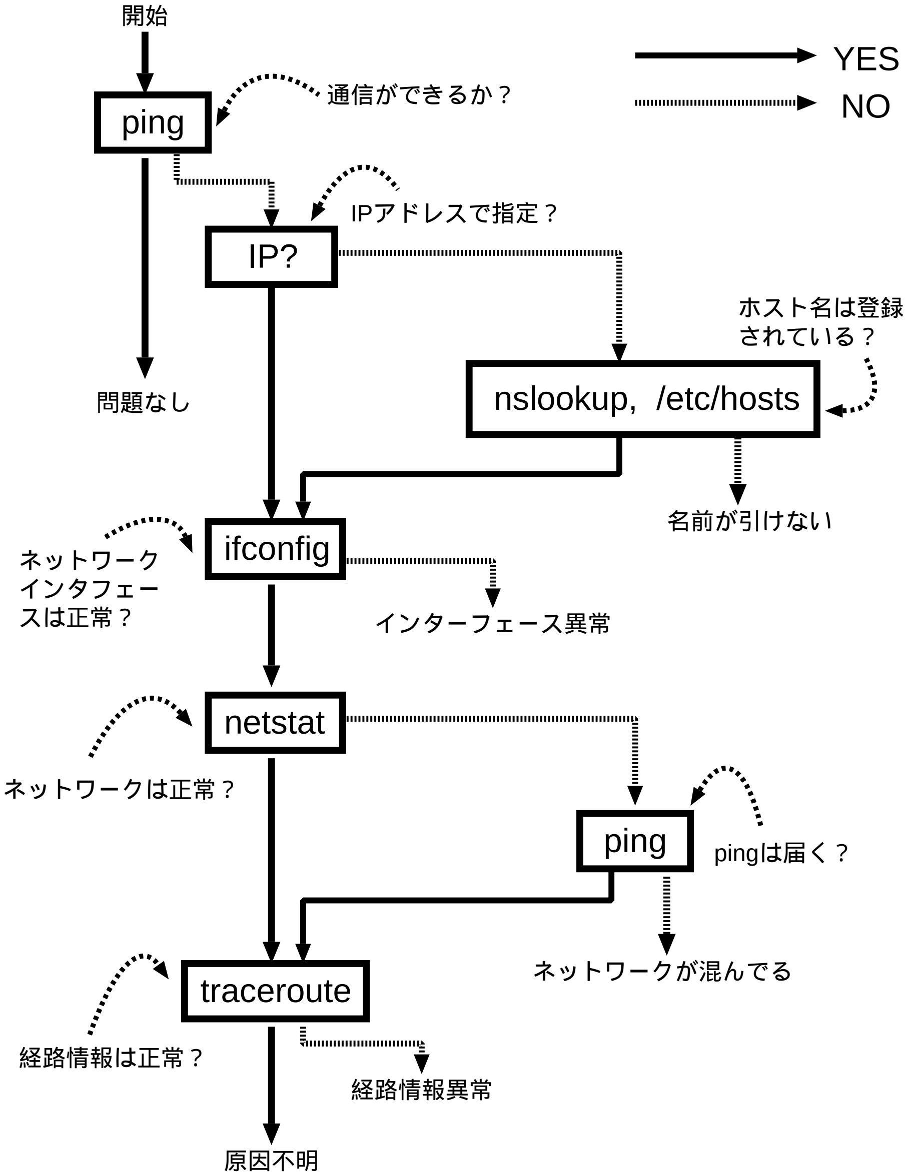


図 5.8 実用例 4:実際の管理作業における分析の流れ

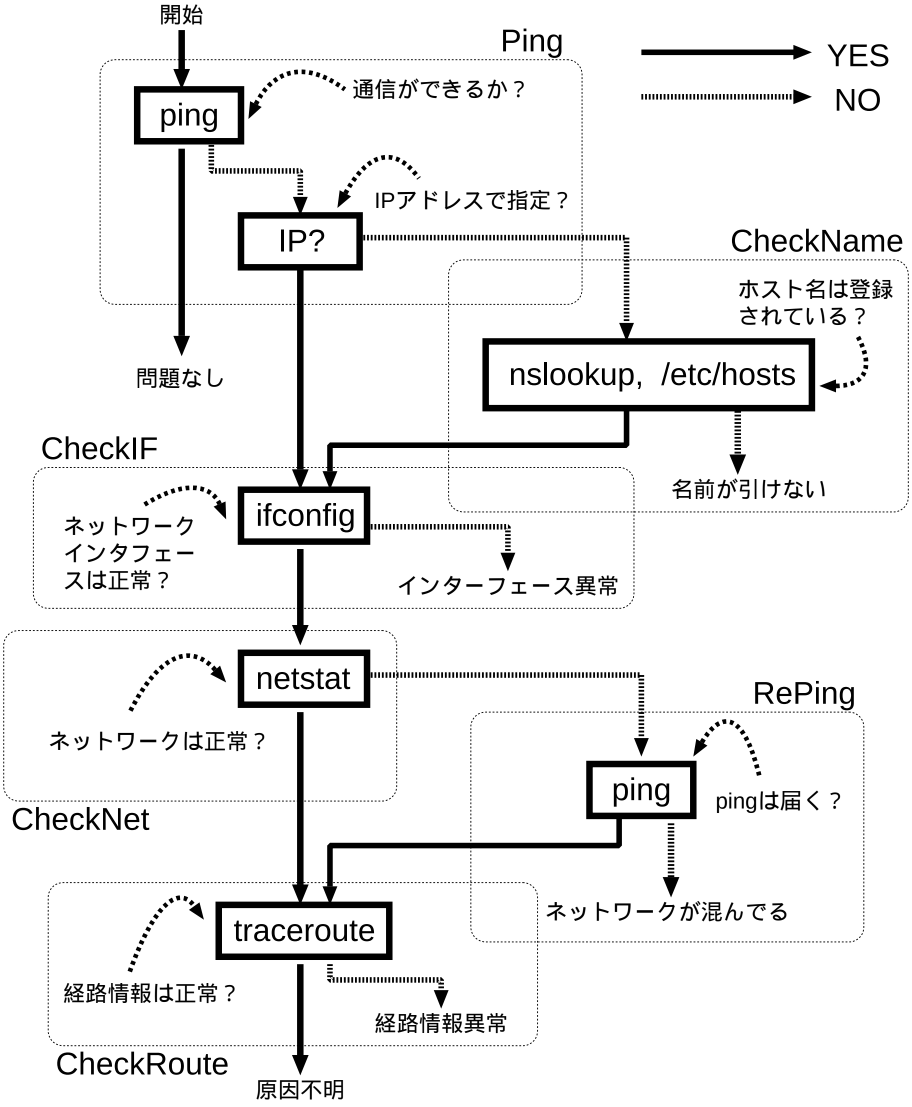


図 5.9 実用例 4:分析の流れに管理知識の対応関係を加えた図

第 6 章

考察

5.1節、5.2節、5.3節、5.4節で 4 つの実用例を示した。ここでは、これらの実用例を通じて分かったことについて述べる。

6.1　管理知識データベース

いずれの実用例も、あらかじめ管理知識データベースに必要な情報を記述することができれば、予想した通りに結果が得られた。管理作業に必要な管理知識をデータベースに登録することができれば、データベースの記述の範囲に限っては、自動的に情報を収集し分析することができる。このように、本論文で実装した支援系は、その能力の大部分を管理知識データベースに依存している。そこで、まず 4 つの実用例の中で問題となったことのうち、管理知識データベースに関連するものについて考える。主なものは以下の通りである。

1. 管理知識から他の管理知識を呼び出す方法がいくつか用意されているが使い分けが難しい
2. 条件規則部で条件が満たされなかった時の行動が記述できない
3. 分析結果を報告書の記述に活かせない
4. 出力形式が一定しないような管理コマンドは定義しにくい
5. 管理知識記述言語の設計の悪さによって記述の自由度を制限してしまっている

管理知識から他の管理知識を呼び出す方法には、「条件式中で関数を用いる方法」と「行動部で再調査を指定する方法」がある。前者の場合は関数`call()`と関数`history()`を利用する。また、後者の場合は、行動部に再調査として呼び出す管理知識名を記述すれば良いだけである。関数による方法と再調査による方法の大きな違いは、再び現在の管理知識に戻ってくるかどうかという点である。例えば、関数`call()`の場合は、呼び出された管理知識は分析規則部の評価結果に従って`RETURN`部の値を返し、最終的に呼び出しもとの管理知識へ戻ることになる。一方、再調査による呼び出しは、これまでの分析結果をすべて破棄し、処理が完全に呼び出された管理知識へ移ってしまう。このような、2 つの呼び出しを使い分けることが、管理知識データベースの記述の上での問題点 (困難さ) の 1 つといえる。この点に関して、熟練者が管理知識を記述すれば良いという考えもあろうが、管理知識の自動生成を目標とする場合はやはり大きな問題となる。

条件規則部の評価の際、行動部は条件部が満たされた場合に実行されるが、逆に条件部が満たされない場合は何も実行されない。しかし、これが場合によっては条件規則部の記述を妨げ、単純に記述できるはずの規則を複雑な記述にしてしまうことがある。ただし、関数`call()`によって呼び出された管理知識では、分析規則部の評価結果がどちらでも戻り値を設定することができるので、このような問題は生じない。

現在の管理知識データベースの記述文法では、分析規則部の評価結果を報告書で利用するような記述ができない。例えば、実用例 1 で「ファイルシステムの利用率の監視」を行なったが、報告書の定義では問

題があったファイルシステムのみ報告するような記述ができない。そのため、報告書の記述の自由度が大きく制限されている。

管理知識データベースに登録する管理知識を定義する場合、まず利用する情報収集の手段である管理コマンドを決定しなければならない。ここで問題となるのは収集される情報の形式である。例えば、実用例 1 の場合、コマンド`df`を用いており、このコマンド`df`のように出力結果が一定しているものは特に問題にはならない。反対に、出力結果が一定しないものは扱いにくいといえる。このような扱える出力結果の範囲を広げることが課題の 1 つである。

また、現在の実装における管理知識記述言語の設計が悪さによって、管理知識を記述する際の自由度を制限していたり、記述のしやすさを損なわせたりしているという問題がある。例えば、現在の管理知識記述言語は次のような点に問題がある。

1. 条件式で複数の値を意味するようにフィールド名を記述した場合の評価方法が 1 通りしかないこと
2. フィールド名の添字が 0 から始まること
3. 1 行を折り返して複数行に記述することができないこと
4. 定義に用いるラベル名が実際の意味を正しく表していないこと

条件式に複数の値を意味するようにフィールド名を記述した場合、現在の実装では要素ごとに条件式の評価を行なう。例えば、条件式で 3 つの値を意味するようにフィールド名を記述した場合、まずの 1 つ目の値で条件式を評価し、同様にして 2 つ目、3 つ目の値で評価を行なう。そして、その結果、真になるものがあれば、その条件式は真である (満たされている) としている。これは、実用例 1 のような場合には、条件式の記述が 1 つですむため非常に便利である。しかし、逆に全てのフィールド名が示すすべて要素について真になる場合に、条件式が満たされたとしたい場合も考えられ、このようなことは条件式の記述で自由に選択できるべきである。

また、他の 3 つの問題はこれによって管理作業支援形の能力自体が左右されるという種類のものではないが、管理知識の記述しやすさや考えやすさに多大な影響を与えるものである。ここにあげた以外にも、このような問題点を発見し改善することによって、より人間が記述しやすい言語にしなければならない。

6.2 管理作業支援系

現在の実装を管理作業支援系全体として眺めた時に、考えられる問題点について述べる。主なものは次の通りである。

1. magP の作業履歴 (ログ) がとられていないので、どのように作業が行なわれたかが分からないこと
2. 現在の実装が実験不足であること

現在の実装では、収集した情報は履歴として記録されているが、どのように動作したのかについては記録が残っていない。収集した情報に対して、どのように評価がなされ、どのような管理知識の呼び出しが行なわれ、どのような分析が行なわれたのかはまったく記録されていない。通常の管理作業において過去にどのような作業を行なったかの記録は非常に重要な情報源である。この通常の管理作業における過去の作業記録と同様に、本論文の支援系も動作の記録を管理し後の作業に利用することができなければならないと考えられる。特に、管理知識を自動生成することを考えた場合、このような作業履歴の管理が重要な意味を持ってくる。

また、現在の実装は実験不足あり、管理者の作業量をどの程度軽減できるかがはっきりとしていない。そのため、管理者が通常の作業を行なうことに比べ、管理知識を記述し管理作業支援系を利用することがどの程度の作業改善になるかははっきりと述べることはできない。

6.3 まとめ

本論文ではシステム管理作業の自動化を目標とし、その部分問題である管理情報の自動収集と自動分析を行なう管理支援系の設計と実装を行なった。現在の実装には、次のような問題が存在していることが分かった。

1. 管理知識記述文法が十分なものではないため、あらゆる分析規則を記述することできない。また、細部の設計が悪く記述しにくい。
2. 支援系の作業履歴が記録されていないため、現在の一連の作業を後に活かすことが難しい。
3. 管理知識の定義に利用できる管理コマンドは、その実行結果の形式によって利用しやすさが大きく異なる。
4. 支援系が実験不足であるため管理者の作業量をどの程度軽減できるのか分かっていない

管理作業支援系は管理知識データベースに依存しているため、あらゆる作業について完全に記述できる管理知識記述文法が存在すれば、管理情報の自動収集・分析作業を実現することができる。つまり、管理コマンドデータベースに関する設計と実装を見直す必要がある。

また、本論文では既存のオペレーティングシステムで構築されているシステムにおいて、管理作業の自動化を目指した。一方、既存のオペレーティングシステムではなく、より管理しやすいオペレーティングシステムを設計し利用することも、管理者の作業量を軽減する方法の 1 つと考えられる。ただし、そのためには、「管理の容易なオペレーティングシステムを設計・実装し、現存する多数の管理の難しいシステムのすべてをこれに移行させせることの労力」と、「現存するシステムの管理作業を援助する枠組を提案し実装する労力」のどちらがより小さいかを見極める必要がある。しかし、現状ではこの問題に対する明確な解答を得ることが困難であるため、本論文では現存するシステムの管理を自動化すること目標とした。

6.4 今後の課題

本論文では、「システム管理を容易にする」ことを目標に管理作業援助システムを設計と実装について述べてきた。そこで、ここでは、今後の課題について述べる。さまざまな課題が存在するが、特に次の 6 つをあげる。

1. magP の評価のための実験
2. 管理知識記述言語の改良及び拡張
3. magP の動作履歴の管理
4. 管理知識を記述するためのインタフェースの設計と実装
5. 管理知識の自動生成
6. 管理作業支援系の常駐部、操作部の設計と実装

すでに、1～3 については述べた。4 は、GUI などを用いてより容易に管理知識が記述できるようにすることを目的とする。また、5 を実現するためには管理作業支援系の動作の履歴の管理とその分析が特に重要である。6 については 4節で述べた通りである。

システム管理はこれまで、人間である管理者がその経験や知識を使って行ってきた作業である。そのため、これを自動化しようとする場合、人間の知能や知識をどのように実装するかがもっとも困難な問題である。そこで、今回は簡単な記述言語を用いて、分析部分の実装を試みた。しかし、汎用のエキスパート

システムを用いて実装するなど、他にも実装方法は考えられ、引続き分析作業についてはかなりの研究が必要であると考えられる。

また、管理作業支援系の実装は、時間的な制限もあり実装が比較的簡単な「中央見張りモデル」を採用した。すでに述べた通り、中央見張りモデルは中央監視ホストがなんらかの障害によって管理不能になった場合、手も足もでない状態になってしまう。将来、これを回避するためにも「分散生き残りモデル」を実装し比較することが必要であると考えられる。

最後に、本論文の目標である管理作業の自動化を通じて、将来どのようなことが実現され得るのかを以下に述べる。

1. 障害対策
   障害や問題が発見された場合、原因・対策を分析し管理者に報告する。対策が確実に正しいことが保証される場合や、特に必要である場合は自動的に(対策)作業を行う。
2. 管理者の不在対策
   管理者不在の場合の障害や問題にも対応できる。
3. 管理者の作業の軽減
   単純な作業の自動化によって作業量を軽減する。
4. 最適なシステム構築の援助(良い環境作り)
   過去の障害記録や現在のシステム状態の分析に基づいて、システムの性能や環境を改善を自動的に行なう。2章で述べた通りより良い環境を目指して改善を繰り返す。
5. 管理者の作業の単純化(未熟、多忙、操作制限などに対して...)
   管理者が未熟であったり多忙であったりする場合に、援助を行ない管理作業を容易にする。また、なんらかの理由によって管理者が直接操作を行なうことができない状態である場合でも、ある程度簡単な操作のみで管理が行なえるようにする。

謝辞

論文作成にあたって、さまざまな助言を頂いた木村研究室の皆さん、そして佐々研究室、柴山研究室の皆さん、Titanet運用センターの新美さんに感謝致します。

研究にあたって御指導を頂いた木村泉教授、大野浩之講師に心から感謝致します。

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] Information Processing Open Systems Interconnection. Specification of abstract syntax notation one (asn.1). International Organization for Standardization and International Electrotechnical Committee, 1987.

[3] Mike Loukides. *System Performance Tuning.* O'Reilly & Associates, Inc., 1991.

[4] 桝野弥生雄. ネットワーク管理における障害管理手法. 情報処理学会第46回(平成5年前期)全国大会, (株)富士通神戸エンジニアリング, 1993.

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

[6] 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.

[7] Marshall T. Rose. *The Simple Book.* Prentice-Hall, 1991.

[8] 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.

[9] E. H. Spafford, J. A. Rochlis, M. W. Eichin, D. Seeley, T.Eisenberg, D. Gries, J. Hartmanis, D. Holcomb, M. S. Lynn, and T. Santoro. 特集 ワーム・ストーリー — インターネット・ワーム事件の全容. *bit*, Vol. 21, No. 14, pp. 4–43, December 1989.

[10] 丸山不二夫・姫宮利融. UNIXスーパーユーザーの教育. 情報処理学会第46回(平成5年前期)全国大会, 稚内北星学園短期大学, 1993.

[11] 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.

付録 A

管理知識記述文法

本論文で実装した管理作業支援系で扱う管理知識は、管理知識データベースに登録される。ここでは、この管理知識を記述するための文法である「管理知識記述文法」についてまとめる。

A.1 管理知識の構成

管理知識は図 A.1のような構造で定義される。管理知識を構成する要素は、ラベル名とその定義の組合せで表わされる。ラベルの一覧は表 A.1の通りである。ラベルおよびその定義は以下に従って行なう。

- ラベルは行頭から記述し、最後に「:」記号を付加する。
- ラベル「CNAME」「CTYPE」「COMMAND」「TIMEOUT」「OBSOLETE」「DATA_MARK」「RETURN」を用いる場合は、定義部分は 1 行でなければならない。
- ラベル「ENDIF」は分析規則部の定義終了を示すため、定義部分はない。
- その他のラベルは定義部分に 1 行以上記述することができる。ただし、「.」(ピリオド) で定義部分の終了を示さなければならない。
- ラベル名に対応する定義の内容は、ラベル名の次の行から記述する。ただし、定義においていくつかの要素を区切って記述する場合は、空白文字およびタブ文字を用いる。

また、以上に基づいて記述された管理知識の例が図 A.2である。この管理知識の場合、最後に報告書定義部があり、ラベル名が「WARNING_MAIL」となっている。このように報告書定義部には他のラベル名と重複しなければ自由なラベル名を用いることができ、このラベルは、実行部の記述時に用いる。

次節以降では、それぞれのラベルの定義部について述べる。

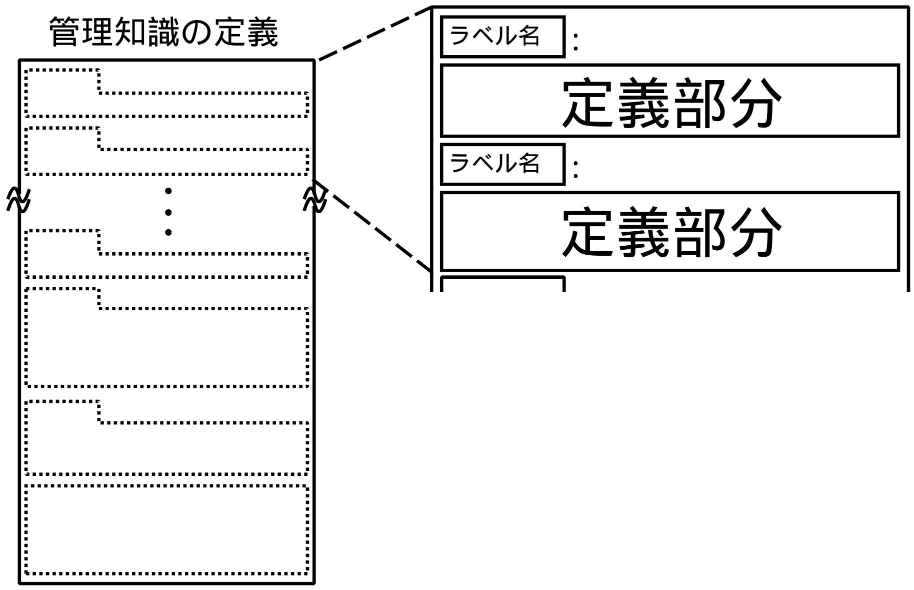
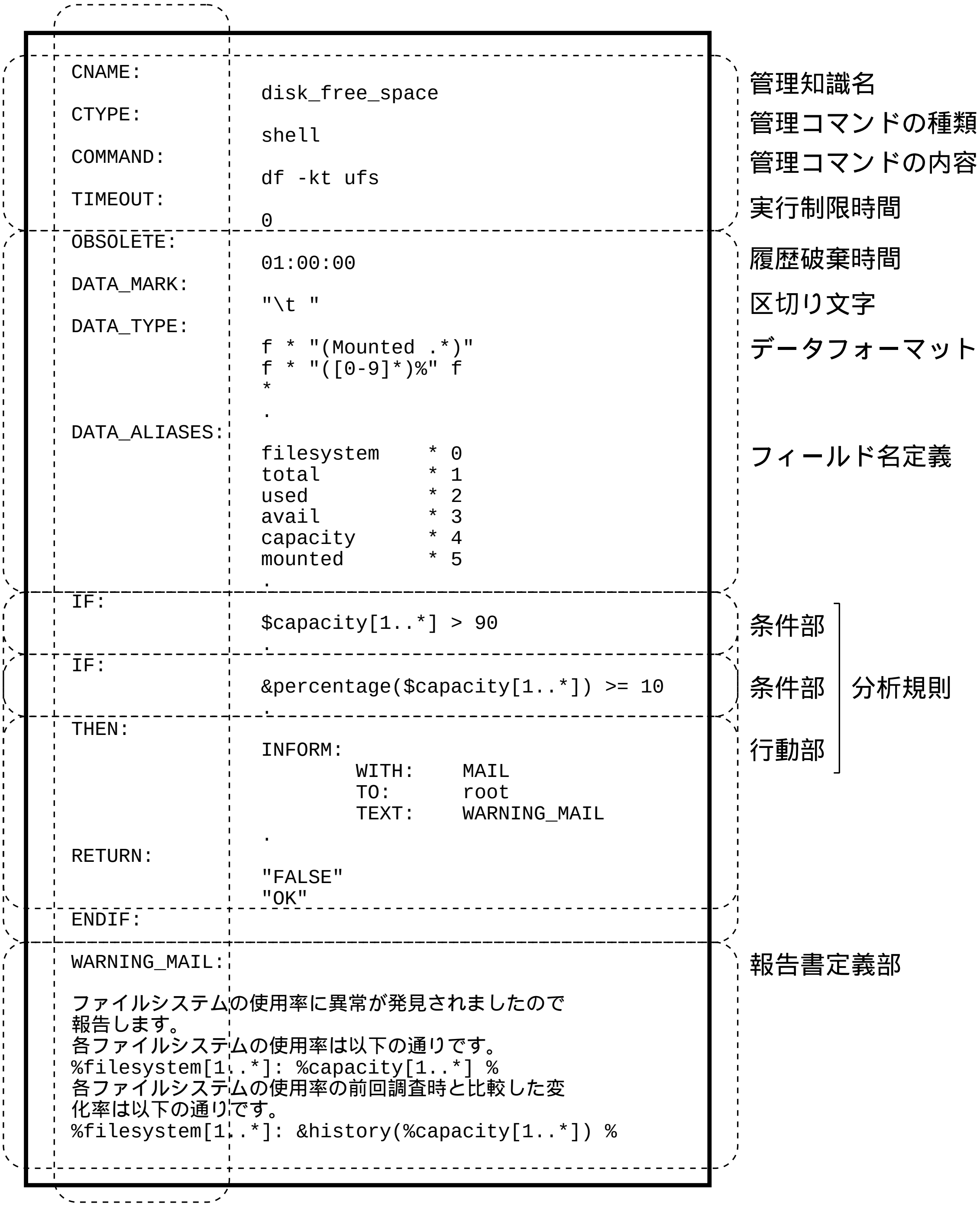


図 A.1 管理知識の定義の構造

表 A.1 管理知識の構成要素

ラベル名	内容
CNAME	管理コマンドの名称
CTYPE	管理コマンドの種類 (`shell`,`remote-shell`,`snmp` のいずれか)
COMMAND	管理コマンドの内容
TIMEOUT	管理コマンドの実行制限時間
OBSOLETE	履歴情報の破棄時間
DATA_MARK	管理コマンドの出力の区切り記号の指定
DATA_FORMAT	データフォーマット (管理コマンドの出力形式を定義する)
DATA_ALIASES	フィールド名定義
IF	条件式の記述
THEN	行動部の記述
RETURN	関数`call()` による呼び出し時の戻り値
ENDIF	条件規則部の終了を示す



```
CNAME:
                disk_free_space
CTYPE:
                shell
COMMAND:
                df -kt ufs
TIMEOUT:
                0
OBSOLETE:
                01:00:00
DATA_MARK:
                "\t "
DATA_TYPE:
                f * "(Mounted .*)"
                f * "([0-9]*)%" f
                *
                .
DATA_ALIASES:
                filesystem    * 0
                total         * 1
                used          * 2
                avail         * 3
                capacity      * 4
                mounted       * 5
                .
IF:
                $capacity[1..*] > 90
                .
IF:
                &percentage($capacity[1..*]) >= 10
                .
THEN:
                INFORM:
                        WITH:    MAIL
                        TO:      root
                        TEXT:    WARNING_MAIL
                .
RETURN:
                "FALSE"
                "OK"
ENDIF:
WARNING_MAIL:

ファイルシステムの使用率に異常が発見されましたので
報告します。
各ファイルシステムの使用率は以下の通りです。
%filesystem[1..*]: %capacity[1..*] %
各ファイルシステムの使用率の前回調査時と比較した変
化率は以下の通りです。
%filesystem[1..*]: &history(%capacity[1..*]) %
```

図 A.2 管理知識の定義例

A.2　管理コマンド部

A.2.1　管理知識名 (`CNAME`)

管理作業支援系で扱う管理知識の名前を指定する。名前は、英字、数字およびアンダースコア (_) によって構成される。ただし、先頭の文字は英字でなければならない。また、英字は大文字と小文字を区別する。

A.2.2　管理コマンドの種類 (`CTYPE`)

管理コマンドの種類を定義する。「`shell`」「`remote-shell`」「`snmp`」のいずれかを記述する。選択したものによって「`COMMAND`」の記述形式が異なる。

`shell`	シェル上で実行する管理コマンド。
`remote-shell`	リモートホストのシェル上で実行する管理コマンド。
`snmp`	SNMP によって指定のホスト上のエージェントと通信を行い MIB 識別子の値を得る。

A.2.3　管理コマンドの内容 (`COMMAND`)

管理コマンドの種類によって以下のように記述形式が異なる。

`shell`	シェル上で実行する場合と同様に管理コマンドを記述する。
`remote-shell`	リモートホスト名に続いて空白を空けた後、シェル上で実行する場合と同様に管理コマンドを記述する。
`snmp`	エージェントホスト名、コミュニティ名、MIB 識別子の順にを空白で区切り記述する。

A.2.4　管理コマンドの実行制限時間 (`TIMEOUT`)

管理コマンドの実行制限時間を秒単位で指定する。ただし、0 を指定した場合は制限なしを意味する。

A.3　管理コマンド出力部

A.3.1　履歴情報破棄時間 (`OBSOLETE`)

管理コマンドの出力 (実行結果) を履歴情報として管理する際に、意味をなさなくなった履歴を破棄する時間を指定する。指定方法は管理コマンド実行時からの経過時間であり、「時:分:秒」で指定する。

A.3.2　出力区切り文字 (`DATA_MARK`)

管理コマンドの出力 (実行結果) をフィールドに区切るために用いる文字を指定する。区切り文字は「`"`」記号の間に並べて記述し、この文字によって構成される文字列によって出力はフィールドに分割される。例えば、空白とタブを指定する場合は、「`"\t "`」のように記述する。ただし、「`\t`」はタブを示すエスケープシーケンスであり、使用できるエスケープシーケンスの一覧を表 A.2に示す。

表 A.2 エスケープシーケンス

表記	意味	表記	意味
\a	アラーム	\’	’
\b	バックスペース	\"	"
\e	エスケープ	\t	タブ
\f	フォームフィード	\033	8 進数表現
\n	改行	\x1b	16 進数表現
\r	リターン	\cX	コントロール文字

A.3.3 データフォーマット (DATA_FORMAT)

管理コマンドの出力 (実行結果) をフィールドに分割する際の形式を定義する。通常、フィールドは「区切り文字 (DATA_MARK)」以外の文字で構成される文字列であり、このようなフィールドを文字「f」で表わす。そこで、出力が「区切り文字」以外の文字で構成されるフィールドで構成され、3 × 5 個のフィールドからなる場合は、次のような定義になる。

```
f  f  f  f  f
f  f  f  f  f
f  f  f  f  f
```

また、フィールドを正規表現によって定義することも出来る。この場合、正規表現を「"」記号で囲む形で記述する。ただし、正規表現によって定義されたフィールドの値を得るためには、あらかじめ正規表現内でどの部分にマッチしたものをフィールドの値とするかを「(」「)」を用いて指定しなければならない。この指定を行わない場合はフィールドの値は常にヌル ("") となる。

繰り返しも記述できる。上記の例のように、同じ定義のフィールドが連続する場合、「*」記号によって繰り返しを指定することができる。この記号は、次の規則に従って解釈される。

1. 第 1 フィールド (先頭のフィールド) として記述した場合は、直前の行の繰り返しを意味する。
2. 第 2 フィールド以降のフィールドとして記述した場合は、直前のフィールドの繰り返しを意味し、次のフィールドの定義がマッチするまでの間繰り返される。
3. 「*」は連続したフィールドに記述してはならない。

また、データフォーマットの定義で用いる記号の意味は表 A.3の通りである。

表 A.3 出力フォーマットの定義に用いる記号の意味

記号	意味
f	通常のフィールド (区切り記号以外で構成される文字列)
*	0 個以上の繰り返し
”...”	正規表現によるフィールドの定義。 (...) によって指定された部分を値としてとる
. (ピリオド)	終了記号

A.3.4 フィールド名定義 (DATA_ALIASES) とフィールドの参照

フィールド値を参照する場合、フィールドに名前を付けなければならない。これをフィールド名と呼び、次のように定義する。

フィールド名　　整数値　整数値

フィールド名は英字、数字およびアンダースコア (_) によって構成される。ただし、先頭の文字は英字でなければならない。また、英字は大文字と小文字を区別する。また、フィールド名に続く 2 つの整数値はフィールドの位置を表し、第 2 行の第 3 フィールドの場合「1 2」と記述する。このように実際の値より 1 小さく、0 以上であることに注意しなければならない。

このように、あるフィールドを単一で参照する場合は整数値を直接 2 つ記述するが、行や列全体にフィールド名をつけ配列のように扱いたい場合は、以下のようにする。

行 (横方向) として扱う場合　...　フィールド名　整数値　*
列 (縦方向) として扱う場合　...　フィールド名　*　整数値

つまり、位置の指定で不確定である方を「*」記号で記述する。ただし、両方を「*」記号によって記述することはできない。フィールドの値を参照する場合は、「$」記号に続けてフィールド名を記述する。ただし、フィールド名が行や列を表す場合はフィールド名のあとに「[添字]」を付加する。添字はフィールド名定義時に省略した方の整数値を指定する。また、表 A.4のように記述すれば一部分を取り出すことができる。

表 A.4 フィールドの参照例

`capacity[2]`	第 3 要素
`capacity[2..4]`	第 3 要素から第 5 要素
`capacity[3..*]`	第 4 要素以降
`capacity[*]`	全要素
`capacity[$]`	最後の要素

A.4　分析規則部

分析規則部を拡張 BNF[1]で定義すると次のようになる。

分析規則部 ::= 分析規則 { 分析規則 }
分析規則 ::= 条件部 { 条件部 } 行動部
条件部 ::= { 条件式 }

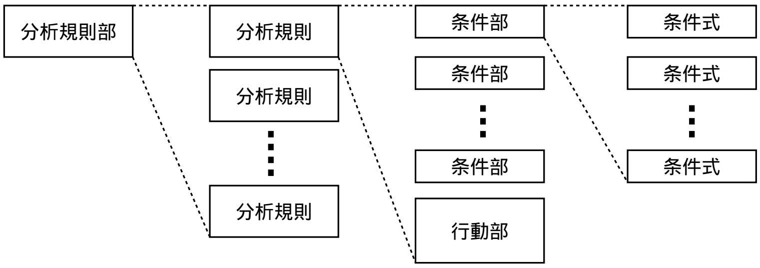


図 A.3 分析規則部の構造

また、この分析規則部はつぎのルールに従って評価される。

- すべての分析規則の評価を行なう
- 分析規則に含まれる条件部のうちいずれかが真のとき、その行動部が実行される
- 条件部の全ての条件式が満たされたとき、その条件部は真となる。

分析規則部はラベル「`IF`」で始まり、ラベル「`ENDIF`」で終りを示し、行動部の開始はラベル「`THEN`」で示す。以上をまとめると、以下のような構造となる。

A.4.1　条件式

条件式に用いることができる比較演算子を表 A.5に示す。また、この他に「`&&`」「`||`」が使用でき、それぞれの意味は「and」「or」である。また、算術演算子、正規表現によるパターンマッチなどすべて、perl と同様に記述できる。

表 A.5 比較演算子一覧 (数値と文字列の区別があるもの)

数値用	文字列用	意味	数値用	文字列用	意味
`==`	`eq`	等しい	`!=`	`ne`	等しくない
`<`	`lt`	小さい	`>`	`gt`	大きい
`<=`	`le`	以下	`>=`	`ge`	以上

---

[1] $\{a\}$ は a の 0 個以上の繰り返しを表す。

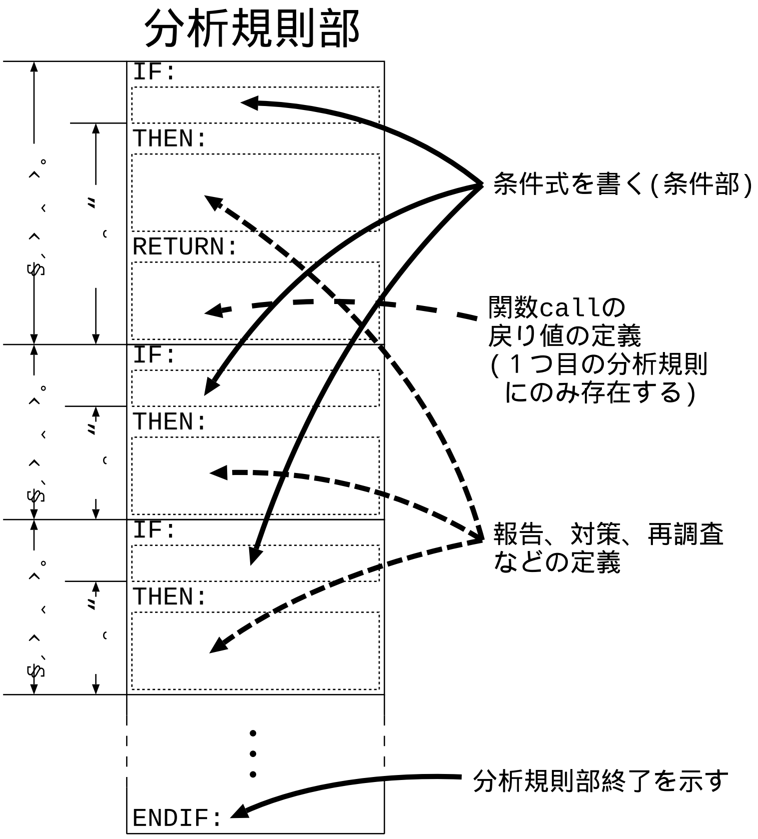


図 A.4 分析規則部の構造

A.4.2 行動部

行動部には、「次の作業」または「関数call() による呼び出しの戻り値」が記述される。それぞれ、ラベル「THEN」とラベル「RETURN」に続いて記述される。ラベル「THEN」に続いて記述するものは「報告」「再調査」「対策」のいずれかであるが、それぞれ以下のように記述する。

報告

報告を行うためには、「手段」「報告先」「報告」の 3 つの要素を決定しなければならない。それぞれ、ラベル「WITH」「TO」「TEXT」に続いて次のように記述する。

```
INFORM:
        WITH: MAIL
        TO:   root
        TEXT: WarningMail
```

「WITH」の後にはMAIL,PAGER のいずれかを記述する。「TO」の後には、電子メールの場合は電子メールアドレス、ページャの場合はページャアドレスを記述する。「TEXT」の後には、報告書定義部のラベル名を指定する。このラベル名は、他のラベルと重複しなければ何でも良い。

再調査

再調査はコマンド名を次のように指定する。

```
RESEARCH:
          CNAME:  Check_DNS
```

この例の場合、「Check_DNS」がコマンド名である。

対策

対策は対策コマンド名を次のように指定する。

```
COUNTER:
          COUNTER-CNAME:  Clear_Core
```

この例の場合、「Clear_Core」が対策コマンド名である。(ただし、これは未実装である)

関数`call()`による呼び出しの戻り値

関数`call()`による呼び出しの戻り値は、ラベル「RETURN」の後の2行に記述する。次のように、1行目に条件部が満たされた場合の値、2行目に満たされない場合の値を記述する。

```
RETURN_0:
            "OK"
            "FAULT"
```

A.5 関数

分析規則部と報告書定義部では関数を用いることができる。関数はフィールドの値の参照や演算以外の分析規則を記述するために用いられる。表A.6がその一覧である。

関数	内容
`change(fieldname)`	変化量を求める
`percentage(fieldname)`	変化率を求める
`history(command, fieldname, arguments)`	指定した管理知識の最新の履歴情報を得る
`history_time(command, arguments)`	指定した管理知識の最新の履歴の時刻を得る
`call(command, arguments)`	指定した管理知識を呼び出す
`shell(command)`	ローカルホストのシェル上でコマンドを実行する

表 A.6 分析条件の記述に用いることができる関数の一覧

A.6　報告書定義部

ラベル名は「行動部」で報告書定義部の指定で用いたものとする。基本的には、テキストデータをそのまま記述すれば良いが、その際に関数およびフィールド名を用いることができる。フィールド名は通常の「$」を先頭に書く方法以外に、「%」を書く方法もある。それぞれ、次のような意味をもっている。

`$name[...]`	要素を空白 1 つで連結した形の文字列で置換する
`%name[...]`	各要素で置換を行ない要素数分行を生成する

また、報告書の生成は次のような流れで行なわれる。

1. 管理コマンドデータベースから報告書の元データを得る
2. 「%」記号によって指定されるフィールド名をその値で置換する
3. 「$」記号によって指定されるフィールド名をその値で置換する
4. 関数を評価し、その値で置換する

ただし、同一行で「%」記号にを用いてフィールド名が記述される場合、置換作業は最初のフィールド名の記述に依存する。

A.7　コメント

定義ファイルでは、行頭に「#」を記述すると、その行のそれ以降をコメントと解釈する。