第 16 部

WIDE/PhoneShell

第 1 章

はじめに

関連する最初の研究 [173] が緒に就いた 1989 年以来、PhoneShell プロジェクトは一貫して広域ネットワーク管理者の作業支援機構のありかたを追求してきた。以下は、過去 3 年間の研究成果の一部で、いずれも “WIDE/PhoneShell” が目指す「ネットワーク管理支援機構」の一翼を担うものである。

1. 音声情報をネットワーク環境で利用する手法 [173, 174]
2. 電話回線から DTMF 信号によって計算機を制御する手法 [175]
3. ページャ[1]を利用したメッセージ伝達システム [176, 177]

ところで、昨年度までは WIDE/PhoneShell を研究テーマに採り入れる研究者は少なかったが、今年度後半になると新規参加者が増え [2]、メーリングリスト上での議論も活発になった。また、メーリングリストでの議論をもとに新たな成果物も得られた。このうち主なものを以下に挙げる。

1. ネットワーク管理者モデルを導入し、WIDE/PhoneShell システムのあり方を明確にした。
2. DTMF 制御系を整理統合した。
3. iNET'92 開催期間中、会場のネットワーク管理に導入しその有効性を示した。
4. WIDE/PCS[3]の運用が複数サイトはじまった。
5. これにともない wncud[4] も複数のサイトで稼働した。
6. Sophia/PhoneShell の実験が開始された。
7. ネットワーク稼働状況を監視する ping スクリプトが開発され, 実際のネットワーク管理に効果を発揮した。

[1] ポケットベルのこと。なおポケットベルは和製英語
[2] メーリングリスト参加者は 1993 年 3 月 31 日現在 18 名。
[3] Pager Control System, ページャ制御システム
[4] NCU 制御デーモン

8. ping スクリプトの改定版である pingd の開発がはじまった。
9. ネットワーク管理用ワーム [178] との連携が検討された。
10. 障害によってネットワークが寸断された際に uucp を利用して障害情報を通知する機構についての検討が始まった。
11. DNS および X.500 を利用した WIDE/PCS 運用情報の管理が提案された。

本報告書には、以上の各項目に関する研究成果を収容した。

本報告は５章から構成され、第１章「はじめに」を大野が、第２章「WIDE/PhoneShell の基礎技術」を大野・矢吹・本間が、第３章「WIDE/PhoneShell によるネットワーク管理支援機構」を大野・新美・徳川が、第４章「WIDE/PhoneShell の今後の展開」を鈴木・清水・野尻・石田・稲田が、第５章「おわりに」を大野がそれぞれ執筆した。

第 2 章

WIDE/PhoneShell の基盤技術

本章では、最初に “WIDE/PhoneShell” の意味について触れ、次に WIDE/ PhoneShell の基盤技術とこの技術を用いて矢吹・本間によって開発された Sophia/PhoneShell について述べる。

歴史的理由により WIDE/PhoneShell は２つの意味を持つ。

著者らは「ネットワークにアクセスできない環境にいるネットワーク管理者が端末装置を使わずに計算機管理を実施するための手法とそれを実現するための機構の総称」の意味で用いており、本報告書でも特に断りなくこの語を用いた場合には、この意味で用いていると考えてよい。PhoneShell ワーキンググループ、WIDE/PhoneShell プロジェクトなどの “WIDE/PhoneShell” もこの意味で用いている。なお、以下に述べるもう一つの意味と明確に区別する必要がある場合には、これを「広義の WIDE/PhoneShell」と呼ぶ。もうひとつの意味は、「計算機へのコマンドやデータ入力を DTMF で行ない、これに計算機が音声や FAX を利用して応答するシェルプログラムの一種」である。「広義の WIDE/PhoneShell」との区別を明確にする場合には、これを「狭義の WIDE/PhoneShell」と呼ぶ。すなわち「広義の WIDE/PhoneShell」は、「狭義の WIDE/PhoneShell」に加えて WIDE/PCS (Pager Control System) [176, 177] や後述する WIDE/pingd、さらにファクシミリ・音声合成・音声認識など、ネットワーク管理に活用可能と思われる手段を積極的に利用しようとする姿勢あるいは方法論である。すなわち以下のような関係がある。

$$(\text{広義の } WIDE/PhoneShell) = (\text{狭義の } WIDE/PhoneShell) + (WIDE/PCS) + ...$$

なお Carl Malamud がその著書の中で “phone shell” [179] として紹介しているシステムは、著者が以前報告したシステムそのものであり [174, 175]、上記の分類にしたがうなら「狭義の WIDE/PhoneShell」 である。

本章では、まず「広義の WIDE/PhoneShell」の背景にあるネットワーク管理者モデルに基づいたネットワーク管理支援機構について述べる。次に「狭義の WIDE/PhoneShell」のひとつとして、上智大学で実装された Sophia/PhoneShell についてその概要を述べる。

2.1 ネットワーク管理者モデル

WIDE/PhoneShell は著者の一人が提案する「ネットワーク管理者モデル」に基づいて設計されている [180, 181]。このモデルによれば、計算機ネットワークの管理を遂行するには以下の３種類のメッセージを送出する機能が必要である (図 2.1)。

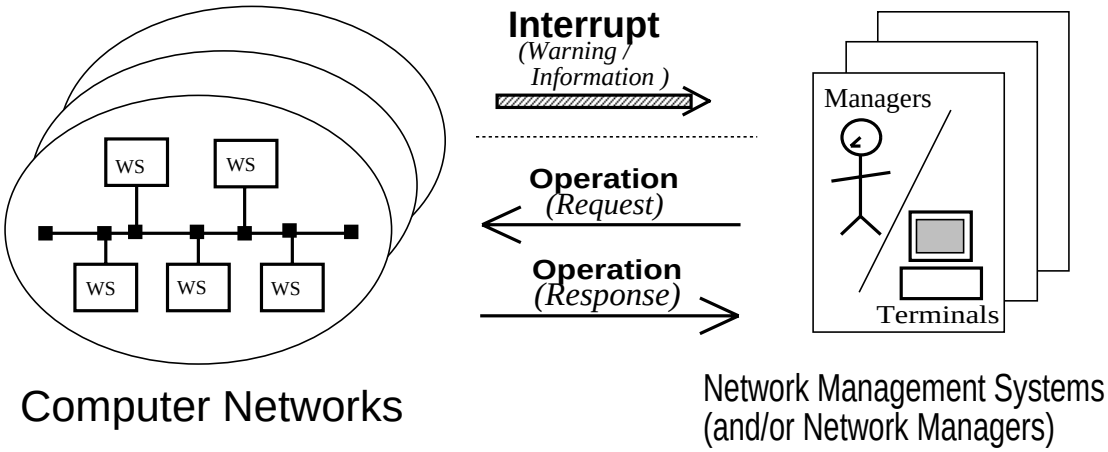


図 2.1: ネットワーク管理者モデル

1. 要求 (request) メッセージ
2. 応答 (response) メッセージ
3. 割り込み (interrupt) メッセージ

このうち要求メッセージは管理者から計算機への指示を、応答メッセージは計算機からの応答すなわち要求メッセージに対する回答を意味する。これらとは別にイベントの発生を計算機が管理者に非同期に通知するメッセージが存在する。これが割り込みメッセージである。文字端末を操作する計算機環境では、要求メッセージはキーボードからの文字打鍵によって、また応答メッセージは画面への文字表示によって実現されており、割り込みメッセージは、メールの到着を告げるベル音などの形で実装されている。割り込みメッセージは、文字端末環境ではあまり重要視されないが、突発する障害を迅速に対処する必要がある管理者にとっては必要不可欠な機能である。

ところで、計算機管理者が端末を直接操作できない環境にいても、これらのメッセージを計算機と交換できればネットワーク管理ができる。逆にこのうちどれか一つでも相手

に送ることができなかったり制限があると、円滑な管理ができなくなる。たとえば、管理者が高速モデムとノート型パソコンと携帯電話を持ち歩いた場合、電波伝搬や電源に問題がなければ要求メッセージと応答メッセージのやりとりは比較的円滑に行なえるが、これらの機材を組み合わせても計算機が管理者に対して割り込みメッセージを発送する機能が提供できないため、管理者は障害発生に気づくことができない。すなわち上記の機材だけでは管理者の作業環境として不十分である。別の例として、管理者がページャを携帯し、そのディスプレイに計算機からメッセージを送ることを考える。この場合、計算機は割り込みメッセージを管理者に送出できるが、その情報量には限りがあり加えて管理者が要求メッセージを送ることができないので、これも作業環境として不十分である。しかし両者を組み合わせると、管理者としての作業を遠隔地からこなすことが可能となる。もちろん、3 つのメッセージを送出できても、回線速度、機材の数、重量、電源、操作性などが問題になるので、3 つのメッセージを送出できることは遠隔管理作業が実施可能であることを示唆するにすぎないが、判断基準として有効である。

2.2 WIDE/PhoneShell が提供する基盤技術

PhoneShell プロジェクトでは、電話回線を介して上記の３メッセージを交換する機構を検討しており、DTMF、音声、FAX などを用いて実装を続けている。具体的なアプローチは以下のとおりである。まず実験系 (図 2.2) 上に以下のサブシステムを構築した。

- DTMF 入出力系
- 音声入出力系
- FAX 入出力系

このうち DTMF 入出力系は 図 2.3 に示すソフトウェア群を介して図 2.2中の NCU を制御することで実現した。DTMF 入出力系は電話回線の制御も受け持つ。なお、DTMF 制御のためのライブラリルーチン (libposh.a) の提供を開始したので、利用者はこのライブラリルーチンを用い通常のファイルアクセス同様の感覚でコーディグできるようになった。図 2.4にライブラリルーチンを使用した DTMF 入出力プログラムの骨格を示す。音声入出力系は、ワークステーションの音声入出力機能と、図 2.2 中の音声処理系および NCU の音声入出力機能を組み合わせて実現した。FAX 入出力系はもっとも単純で、ハードウェアは市販の FAX モデムを接続しただけであり、ソフトウェアもビットイメージを FAX とのあいだでやりとりするという単純なものである。

WIDE/PhoneShell では、これらのサブシステムを以下のように組み合わせて、３種類のメッセージを発生させている。

- 要求 (request) メッセージ
  - DTMF 入力系を利用 (DTMF を直接入力)
  - 音声入力系を利用 (音声認識) [1]
  - FAX 入力系を利用 (手書き文字認識) [2]
- 応答 (response) メッセージ [3]
  - 音声出力系を利用 (音声応答)
  - FAX 出力系を利用 (会話的な処理には向かない)
- 割り込み (interrupt) メッセージ [4]
  - DTMF 出力系を利用 (DTMF を送出しページャにメッセージを伝達)

[1] 計画中
[2] 計画中
[3] 電話回線の制御は DTMF 入出力系の機能を利用
[4] 電話回線の制御は DTMF 入出力系の機能を利用

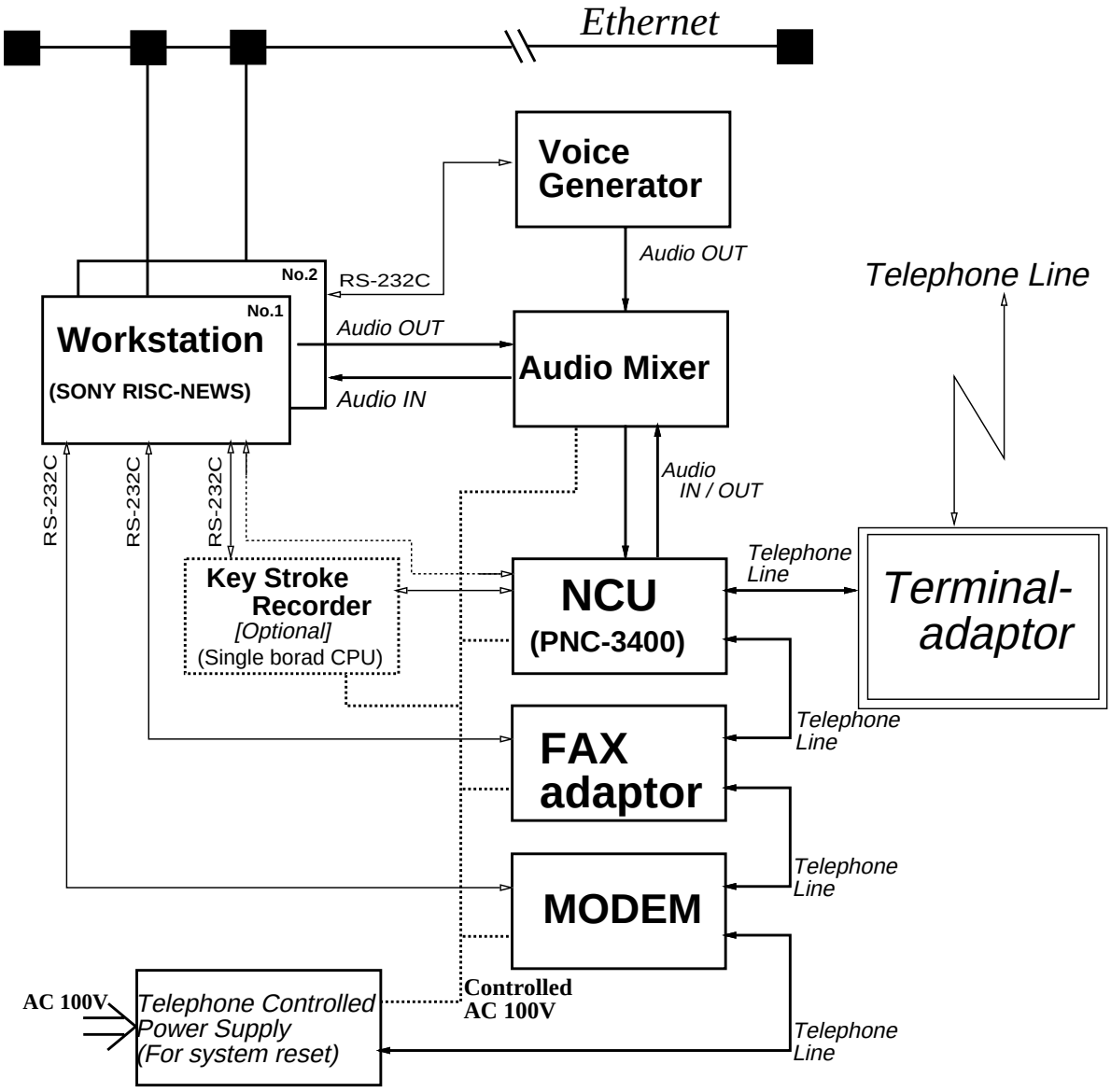


図 2.2: 実験系のハードウェア構成

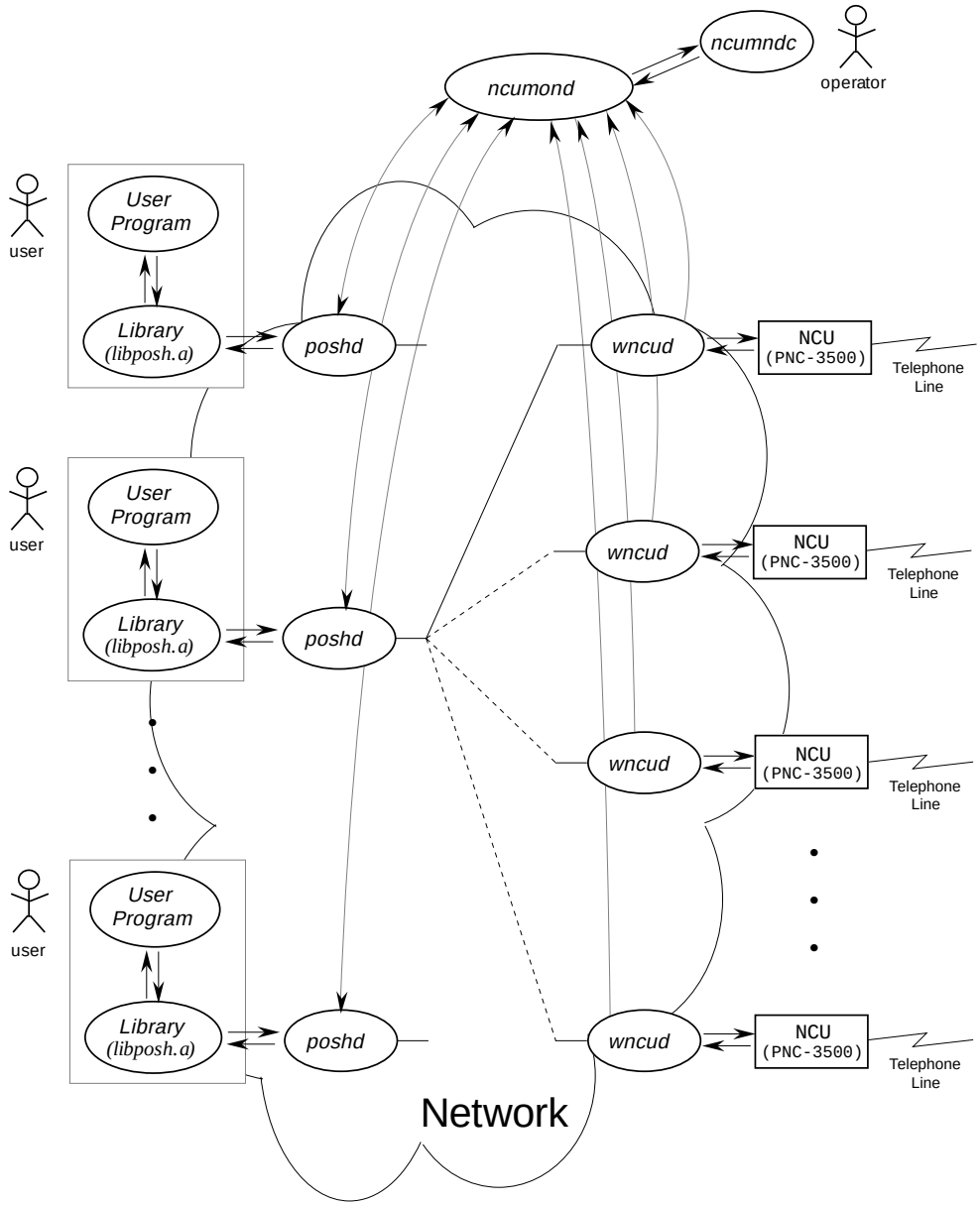


図 2.3: 実験系のソフトウェア構成

```
 1:  /*  *** DTMF 送受用ライブラリルーチン (libposh.a)  使用例。 ***  */
 2:
 3:  if (mode == NCU_CALLMODE) {
 4:
 5:      /* NCU_CALLMODE: NCU を制御してこちらから電話をかける場合。 */
 6:      /* （第３引数として相手の電話番号を指定している）             */
 7:      ncuid = ncu_setup(NCU_CALLMODE,NCU_RWMODE, telno);
 8:
 9:  } else if (mode == NCU_RECVMODE) {
10:
11:      /* NCU_RECVMODE: 着信を待つ場合。着信を監視する対象となる    */
12:      /* wncud 稼働しているホスト名とポート番号のリストを渡す。    */
13:      ncuid = ncu_setup(NCU_RECVMODE, NCU_RWMODE,
14:                        HOST1_ADDR  , HOST1_PORT,
15:                        HOST2_ADDR,   HOST2_PORT,
16:                        HOST3_ADDR,   HOST3_PORT,
17:                        NULL);
18:  }
19:
20:  if (ncuid < 0) { exit(1); }  /* ncuid が負ならここで終了する。   */
21:
22:  /* ncuid が取得できたら、ncuid を引数にして ncu_open() を呼ぶ。   */
23:  /* wncud との接続が完了するまで、ブロックされる。                 */
24:
25:  if ((fd = ncu_open(ncuid, NCU_WAIT_ON_OPEN)) < 0) {
26:      /* fd が負ならここで終了する。*/
27:      exit(1);
28:  }
29:
30:  /* ここに達したらファイルディスクリプタ fd を介し read() write() */
31:  /* NCU との間で DTMF のやりとりが可能。たとえば 数字、#, *, A..D */
32:  /* のいずれかを write() すると対応する DTMF が送出される。また、 */
33:  /* 文字列を read() できたら DTMF が受信されたことを意味する。     */
34:
35:  ret = 0;
36:  while(ret >= 0) { ret = read_and_write(fd); }
37:
38:  /* 終了時には close() ではなく ncu_close() を用いる必要がある。   */
39:
40:  (void)ncu_close(fd);
41:
...
```

図 2.4: DTMF 制御プログラムの概要

– 音声出力系を利用 (管理者の携帯電話に音声で伝達)

WIDE/PhoneShell プロジェクトは、端末装置を使わずに 3 種類のメッセージを送出するシステムの実装をめざしている。この実装が完了すると、ネットワークに不調があってもそのルートを回避してメッセージ交換ができる。これをモデル化すると図 2.5 のようになる。

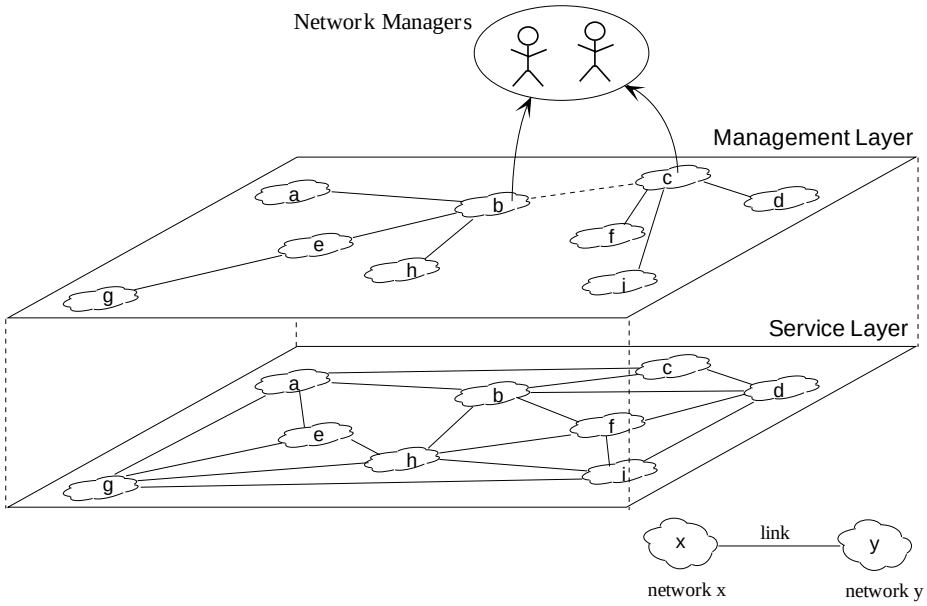


図 2.5: 2 つの階層からなるネットワークのモデル

コンピュータネットワーク上の情報は、通常図の下のレイヤ (サービスレイヤ) を行き来するが、管理情報は必ずしも通常のルーティングだけによらず、それ以外の経路も認めようという考え方がこのモデルの根底にある。この管理情報の通路を管理レイヤと呼ぶことにする。管理レイヤには、システムの異常を通知する情報などが流れる。従来のネットワークでは、既存の物理層が利用できなくなるとネットワークは寸断され、管理情報も流通しなくなる。しかし上記のモデルでは、管理レイヤはサービスレイヤのネットワーク構造と同じである必要はないので、代替経路が確保できればそれを利用して管理情報を流通させればよい。WIDE/PhoneShell が提供するメッセージ送出機能がこのような場合に有効に働く。すなわち WIDE/PhoneShell は、ネットワーク構成を超越した情報交換を提供する機構と位置付けられる。

2.3　Sophia/PhoneShell

前節では、ネットワーク管理業務における「広義の WIDE/PhoneShell」の位置付けをあきらかにし、実装状況についても言及した．本節では、「狭義の WIDE/PhoneShell」の現状報告として，上智大学で実装された Sophia/PhoneShell を取り上げる．

PhoneShell ワーキンググループでは、ネットワークにアクセスできない環境にいる管理者が端末装置を利用せずに管理作業を実施する手段について研究している。そして電話および電話が提供している各種サービスに注目しそれらの効果的な利用を検討している。非常時[5]においては特に電話の利用が効果的であり、これまで通常の電話機のプッシュボタンによる ping、あるいは非常時のページャを使った呼び出しなどが研究されてきている。

ネットワーク利用者は通常の通信手段として電子メールを用いている。しかし現状では手元に端末がない場合には、これを読む手段を持たない。将来的に無線等を用いた小型の移動計算機が利用できるようになれば解決されるが、それまでは非常時に電子メールを読むための何らかの代替手段を持つことが重要である。

現状では個別の特別なハードウェアを必要とせず、また、もっとも広く利用可能な通信手段は電話である。したがって通常の電話機から計算機、あるいは計算機ネットワークにアクセスし、電子メールを読む手段が望まれることになる。すなわち、電話回線からのプッシュボタンの制御によって電子メールを読みあげるシステムである。(本稿では「電子メール読み上げシステム」と呼ぶ。)

これまで、電子メール読みあげシステムを実現するには、実際に音声を発生する装置が高価であることが問題であった。しかし最近多くのワークステーションにマルチメディアのためのペリフェラル及びソフトウェアが備えられるようになり、記憶したディジタル音声データをワークステーションから発生させる機能が標準的に実装されはじめている。また、シリアル回線の制御に基づく漢字まじり文の音声合成装置も 20 万円程度と安価に入手可能である。さらに「かかし」[6] などの日本語テキスト逆変換 (漢字まじり文からかな文へ) のソフトウェアも開発された。

電話回線からの制御を可能とする DTMF 装置とこれら音声合成装置あるいは音声合成可能なワークステーションを利用することにより、電子メール読みあげシステムが現実的に実現可能となってきた。これらを活用する事によって、上智大学では電話回線による電子メール読みあげ実験システムを構築した。

2.3.1　電子メール読みあげシステムと PhoneShell ワーキンググループ

PhoneShell ワーキンググループの研究成果である DTMF 制御技術を利用して、上智大学で電子メール読みあげシステムを実験的に構築した。その結果若干の問題はあるも

[5] 本節で用いる「非常時」とは、災害などの場合ではなく、計算機にアクセスする必要が生じた際に手近にその手段がないような場合を指すこととする。

[6] 漢字仮名変換のフリーソフトウェア

のの、実用可能なシステムとなる感触が得られ、αバージョンの製作に取り掛かっている。αバージョンでは明電舎の協力により入手した明電舎製音声合成装置を利用している。αバージョン完成後は PhoneShell ワーキンググループにソースコードを公開する予定である。

2.3.2 実験システムの検討

実験的に作成した電子メール読みあげシステム (以下実験システム) では、pop および mh に対応したユーザインターフェースを作成した。電話機のプッシュボタン [0-9*#] により個人のメールの制御が可能で、緊急時のメールからの情報の取得には十分利用可能であることが確認された。

作成した実験システムをもとに検討を行なった結果、電子メール読みあげシステムとして以下の問題について解決を行なう必要があることが判明した。

1. ユーザインターフェース
   ユーザインターフェースは入力できるキー [0-9*#] が少ないため、より重要となる。特に通常は利用せず非常時にのみ使用されると考えられ、複雑な操作を記憶していることをユーザに要求することはできない。よく整理された、階層構造をもったコマンド体系を作成する必要がある。同様の理由から、ヘルプ機能も重要となろう。

2. テキストから音声出力へのフィルタリング
   人間がテキストを理解する場合には音声に置き換えて理解することはない。特に記号列などはパターンで認識しており、対応する音声として「聞く」とかえって理解の妨げとなる場合が多い。特に電子メール読みあげシステムでは、メールヘッダが問題となる。例えば、非常時に電子メール読みあげシステムを利用するような場合には、メールの経由の履歴などは必要ではなく、省略した方が分かりやすい。すなわち、テキストから音声出力への変換を行なう際のフィルタリングが重要となる。フィルタリングにおいては以下のものが必要である。

   - 不必要な情報の削除
     メールヘッダなど
   - 記号列の読み替え
     メールアドレスなど

   現状では英語/日本語の双方に対応した音声合成装置がなく、英語のテキストが記号列と認識されてしまうため、特に後者は重要である。

3. セキュリティ
   実験システムではセキュリティについてあまり考慮されていないが、実用システムとしては重要な問題となる。特に入力できる記号が [0-9*#] と限られるため、認証に利用できる組合せも少なくなってしまう。また、電子メイルは個人のプライバシー

に関係するため、その意味からも重要である。 電子メール読みあげシステムが実現されたとしても、非常用であるため、経済性から考えても、各マシンに音声合成装置と電話回線を備えた電子メール読みあげシステムが実装されることは考えられない。すなわち、電話回線からの制御を受けるシステムと実際にメール等のデータを持つシステム (ホームホスト) とは異なることになり、ネットワークを介した通信が行なわれる必要がある。この意味からもセキュリティは十分に考慮されなくてはならない。

2.3.3　電子メール読みあげシステムから PhoneShell へ

作成した実験システムは純粋に「電子メール読みあげシステム」である。すなわち、電子メールを読むことを前提に作成され、さまざまな機構が電子メールと直接結び付いている。しかし、電子メール読みあげシステムとは何かを考えてみると以下のようになる。

- 電話回線からの制御
- DTMF を通じたコマンドの実行 (電子メール読みあげシステムでは show,next など。)
- 実行されたコマンドのフィルタリングと音声合成装置からの出力

すなわち、電子メールに限らず何らかのコマンドを実行し、必要なフィルタをかけてその結果を音声合成装置から電話回線を通じて「聞く」ものである。この意味において、電子メール読みあげシステムは PhoneShell[7]のサブセットとして定義される。

これを踏まえてαバージョンでは、電子メールに依存する部分は切り離し、PhoneShell 上のコマンドとして電子メールに関連するコマンドを実行するものとする。勿論、電子メール読みあげに関する機能はその中核をなすであろう。この方法により、多くの電子メールシステムに対して、対応するフィルタを作成するだけで容易に対応することが可能となる。イメージとしては以下の 3 点が PhoneShell の標準的な実行となる。

- 電話回線からのコマンドの発呼
- 個々のコマンドに対応したフィルタの実行
- フィルタリングされた結果の出力

フィルタはすべて別コマンドとして外部に位置し、自由度を高めるようにする。専用のコマンドとそのフィルタを作成すれば、ping ばかりでなくさまざまなネットワーク監視コマンドを用いて、電話回線からネットワークの状態を把握することも可能となるだろう。

[7]本節で言及している PhoneShell とは、前述の「狭義の PhoneShell」である。

2.3.4 Sophia/PhoneShell の設計

Sophia/PhoneShell は自由度が高くなるように設計を行った。複雑な機能のプロセスを作成せず、機能的に分離されたプロセス群よりシステムを構成している。このため、各々のプロセスを単純化し、それぞれの持つ機能を定義して、各プロセス間のプロトコルを定めた。これらプロセスの通信は、 DTMF 装置が接続されるホスト、音声合成機能を持つ (あるいは音声合成装置が接続される) ホスト、必要とするデータ (例えば電子メール) を持つホストがそれぞれ異なる事を想定して設計を行った。

設計において最も問題となったのは、ユーザの認知と必要とするデータへのアクセス権である。前にも述べた通り、通常は電話回線を制御するホストとデータを持つホストが異なることを想定している。そのため電話回線を制御しているホストのプロセスでは、データを持つホストにネットワークを介して通信を確立しなくてはならない。必要となるデータはメール等個人のプライバシに関連するデータが想定されるため、厳重な認証を必要とする。言い換えれば、システムを利用するユーザの権限でデータを保有するホストにアクセスしなくてはならない。

このためには、ログインアカウントを利用した認証を行なうべきである。 2 つの方法が考えられる。

1. データを持つホストでデーモンプロセスを実行し、通信の接続において認証を行なう。このシステムではキー入力として [0-9*#] のみが可能である ため、数字の識別番号と設定し、実際にログイン名とマップし、数字のパス ワード (/etc/passwd のパスワードとは異なる) により認証を行ない、実際 のログインアカウントのユーザプロセスとなるよう setuid する。

2. データを持つホストにおいて数字だけで識別されるユーザアカウントを作成し、リモートログインにより通信を確立する。この際数字ユーザアカウントの uid および gid を実際のユーザのアカウントと一致させる。ただし、シェルとして専用のプロセスが起動されるように設定する。

方法 1. の利点は次のようなものがあげられる。

- ログインを必要としないため新たにログインアカウントを設定する必要がない。このため専用アカウントを作成が不可能なホストにおいても利用可能である。
- ログインが行なわれないため、ユーザを調べるコマンド (w あるいは finger) などにより容易にシステムの利用を検知されることがない。
- 認証方法が独自に設定できるため、[0-9*#] による認証においてもセキュリティを向上させることが可能である。

方法 1 の利点はすなわち方法 2 の欠点でもある。
一方、2. の利点は次の 2 点である。

- リモートログインによる認証を用いるため認証のメカニズムを必要としない。
- /etc/passwd のエントリにより通常ユーザと同じ管理が可能となる。

Sophia/PhoneShell では、自由度の高い方法 1 を採用する。
Sophia/PhoneShell を構成するプロセスは次のようなものとする。

- ホストデーモンプロセス
  ユーザの電話からの入力を監視し、データを持つホストとの通信の確立および通信、並びに音声合成プロセスへの出力を行なう。
- NCU 制御プロセス
  NCU 装置を監視し、電話からの入力をホストデーモンプロセスに送る。ホストデーモンプロセスと一体化することも可能であるが、システムの自由度を向上させるため別プロセスとする。
- 音声発生装置制御プロセス
  ホストデーモンプロセスと一体化することも可能であるが、音声発生機能を持つホストが異なるホストであうことを想定し、別プロセスとしてプロセス間通信を行なう。
- リモートデーモンプロセス
  ホストデーモンプロセスからの要求に応じて処理を行なう、通常のシェルに相当するプロセス。ユーザの認証および実ユーザとのマップを行ない、実ユーザの権限での処理を行なう。

2.3.5　Sophia/PhoneShell の構成

Sophia/PhoneShell は設計にしたがって次のプロセスから構成されるものとする。全体の構成図を図 2.6に示す。

1. phntserv
   ホストデーモンプロセスである。ユーザからの入力を監視し、データを持つホストとの通信の確立および通信、並びに音声合成プロセスへの出力を行なう。 phntserv はユーザの入力した数字 ID を認識して、必要となるリモートホストのデーモンプロセスとの通信を確立する。このため phntserv は数字ユーザ ID と接続すべきホストとのマップファイルを必要とする。phntserv に利用者はすべてこのマップファイルに登録されていなければならない。また、通常はリモートデーモンプロセスのポート番号は固定であるが、ホストによってデフォルトのポート番号が利用できない場合もあるため、オプションとしてポート番号も指定することができる。マップファイルは 1 行エントリで、次のようになる。

   数字ユーザ ID:ホストアドレス (ドメイン|IP アドレス):ポート番号 (オプション)

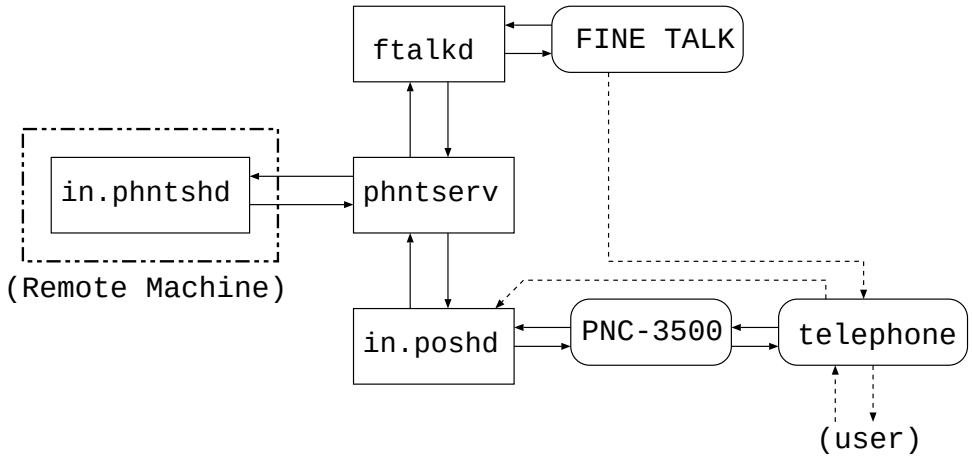


図 2.6: Sophia/PhoneShell 構成図

2. in.poshd
   NCU 装置を監視し、電話からの入力をホストデーモンプロセスに送るプロセスである。PhoneShell プロジェクトで既に開発されたものを流用する。ここでは IO データ社製 PNC-3500 を制御するものである。機能としては、電話回線の自動受信 DTMF の取り込みなどである。取り込まれたデータは phntserv に送られる。

3. ftalkd
   phnserv から受信したデータを音声発生装置用の音声データに変換し、音声発生装置に出力する。現状では明電舎製 finetalk を想定している。将来的には NTT のしゃべりんぼう、漢字逆変換ソフトウェア「かかし」等にも対応する。 さまざまな音声発生装置に対応するために、音声発生装置制御の標準化を行なう必要がある。これについては、東京工業大学での研究に基づいて、printcap あるいは termcap のような記述ファイルを通じた標準化を行なうこととする。

4. in.phntshd
   ホストデーモンプロセスからの要求に応じて処理を行なう、通常のシェルに相当するプロセスである。ユーザの認証および実ユーザとのマップを行ない、実ユーザの権限での処理を行なう。マップファイルの形式は 1 行 1 エントリで次のようになる。

   数字ユーザ ID:暗号化されたパスワード:実ユーザ ID

   通常は inetd から起動されるように設定するが、管理の制限のため inetd の設定が不可能な場合に対応するため、個人デーモンの形での実行も可能とする。

   in.phntshd はユーザ権限での実行開始においてユーザが作成した設定ファイルを読み込み、各種設定を行なう。設定は、

キー : コマンド : フィルタ

の組の指定と、利用のためのヘルプ機能の設定である。電話からの利用では利用できるキーが少ないため、キーの階層構造を取る必要がある。コマンドは通常のコマンドでもよいし、専用のコマンドでもよい。このように実行を外部コマンドとすることにより、ユーザに応じた自由な利用が可能となる。

また、セキュリティを考慮して、接続を受け付けるホストの指定も設定ファイルで行なう。

2.3.6　まとめ

Sophia/PhoneShell は実験システムが完了し、αバージョンの設計が終了したところである。αバージョンの運用を開始し、実際の利用を通じて改良を行なっていく予定である。

第 3 章

WIDE/PhoneShell によるネットワーク管理支援機構

本章では、WIDE/PhoneShell のコンセプトに基づいたネットワーク管理支援機構の現状について報告する。

具体的には WIDE/PCS に注目し、以下の話題を検討する。

- 現在の WIDE/PCS の運用状況について。
- iNET'92(1992 年 6 月に神戸で開催) において、会場に開設されたネットワークの維持管理を支援する目的で導入された WIDE/PCS の運用結果。
- ネットワークの稼働状況を監視するために開発した ping script のしくみと運用状況について。

3.1　WIDE/PCS の運用状況

現在、PhoneShell ワーキンググループでは複数のサイトで WIDE/PCS が動作している。(表 3.1) また、図 3.1に WIDE/PCS 稼働サイトのネットワーク的な位置を示す。

表 3.1: WIDE/PCS 稼働サイト一覧 (稼働開始順)

サイト名	接続 WNOC	WNOC 側機器名
東京工業大学	WNOC-TYO	wnoc-tyo
株式会社ディアイティ	WNOC-TYO	wnoc-tyo
富士ゼロックス株式会社	WNOC-SFC	wnoc-fujisawa-proteon
神奈川大学	WNOC-TYO	wnoc-tokyo-cisco

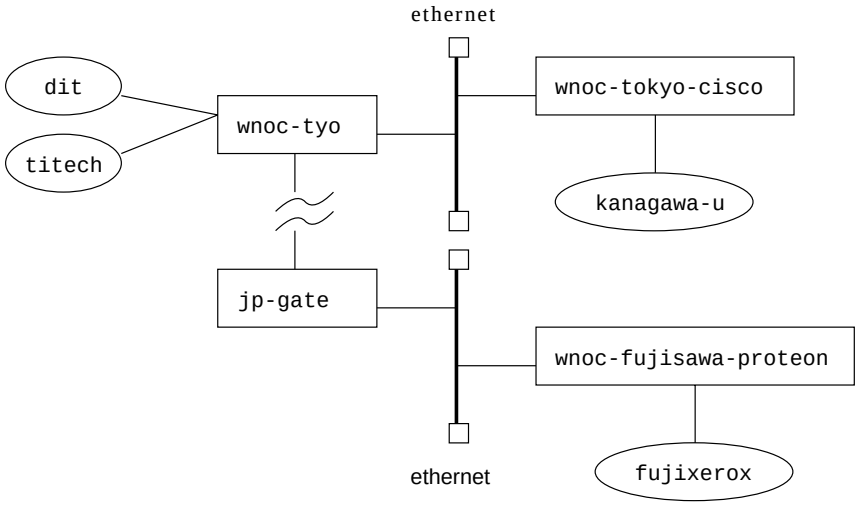


図 3.1: WIDE/PCS 稼働サイトのネットワーク的な位置

また、以下のサイトが稼働準備中である。

- 株式会社日立製作所
- 東京大学
- 株式会社フォア・チューン

また表 3.2に各サイトの歩みをまとめた。

日付	サイト名	記事
91/05/25	東京工業大学	試験運用開始 (OMRON LUNA)
91/06/		自由文型対応
91/12/	東京工業大学	機種変更 (Sony RISC NEWS)
92/06/14-19	INET'92	INET'92 の運営で WIDE/PCS を使用
92/12/25	ディアイティ	AT モデムで WIDE/PCS 開始 (Sun Sparc Station 2)
93/01/07	ディアイティ	社内向け ping スクリプト開始
93/01/11	ディアイティ	WIDE 内 ping スクリプト開始
93/01/23	富士ゼロックス	AT モデムで WIDE/PCS 開始 (Sun-4/110)
93/01/25	神奈川大学	AT モデムで WIDE/PCS 開始 (Sun-4/370+Sun-4/470)
93/02/02	ディアイティ	PNC-3500 を導入

表 3.2: 各サイトの WIDE/PCS の歩み

3.2 サイト間の協力

WIDE/PCS 稼働サイトでは、相互に協力しあって WIDE/PCS を運用をしようという計画がある。この計画には 2 つの意義がある。まず、一部の WIDE/PCS サーバが利用不能な状態になっている場合のバックアップの体制を確立することであり、二番目にページャを呼び出すユーザ側が WIDE/PCS の動いているサイトを意識せずに使えるような環境を提供することである。

3.2.1 wncd と pcsd 間のサイト間協力

WIDE/PCS にはページャ制御サブシステムと呼ばれる 2 つのサーバがある。1 つは wncud と呼ばれ、NCU(網制御装置) を制御し、ページャの駆動に必要となる電話回線の制御や DTMF 信号の発信を行なう。もう 1 つは pcsd と呼ばれ、メールインターフェースなどのフロントエンドサブシステムから送られてくる『リクエスト』と呼ばれるページャ制御要求を受け付けて wncud に転送したり、wncud からの応答をフロントエンドサブシステムに返送する役割がある。

pcsd は『リクエスト』ファイルの内容に応じて、複数存在する wncud から適当な wncud を選択する機能を持つ。選択条件としては以下のようなものがある。

- リクエスト発行者のログイン名
- リクエスト発行者の所属組織
- リクエストの発信時刻
- ページャの電話番号

- 個々の wncud の所在地
- ページャと wncud 間の電話料金
- 個々の wncud が稼働状況
- 個々の wncud へのその時点での到達性

pcsd はこれらの条件を判断し、各サイトで立ち上がっている wncud を選択し、『リクエスト』を最小のコストで、あるいはもっとも優先順位の高い wncud に伝えようとする。

具体的には pcsd は以下のような手順で、wncud を選択する。

1. 選択条件で選ばれた一番優先度の高い wncud へ接続しようとする。
2. その wncud が駄目な場合は、次の優先度の wncud へ接続しようとする。
3. IP 的に全ての wncud に接続できない場合は、UUCP を利用して適切な wncud に『リクエスト』ファイルを転送する。

   詳細は第 4 章で述べる。

3.2.2　分散データベースの利用

仮想的なアドレスにメールを送ると適切な WIDE/PCS 稼働ホストを選択してメッセージを送出するようなシステムを DNS(Domain Name System) の MX フィールドを利用して構築できれば、ユーザはこのアドレスだけを覚えておけばよいことになる。さらに、設定により適切なサーバが選択されるようになるので、システム全体の利用効率も向上することになる。

また、WIDE/PCS 稼働サイトでとり扱う情報などの共有に分散データベースが利用できれば、WIDE/PCS の運用管理が容易になる。分散データベースとしては、DNS と Directory Service(X.500) の利用が考えられる。

これらについても第 4 章で述べる。

3.3 ページャの利用について (神奈川大学の例)

本節では、ページャの利用状況について神奈川大学を例に取り上げて述べる。

神奈川大学では 1992 年 12 月より管理者 1 名がページャを携帯し、ネットワーク管理に利用している。

3.3.1 第 1 期 1992 年 12 月

shell script+AT モデム+TTM NIP 型ページャ

1992 年 12 月の時点で管理者が使用していたページャは東京テレメッセージの NIP 型[1]と呼ばれるもので、漢字仮名混じりの定型文や組み込みのイラストのほか、8 文字以内で英数字、平仮名のメッセージを送ることができるタイプのものである。

ページャの駆動には、awk、shell スクリプト、procmail[2]などを組み合わせた簡単な独自のプログラムを用いており、uucp 用に設置した AT モデムに直接 AT コマンドを送り込むという方法を取っていた。

利用方法としては、主に以下のようなものがあった。

1. メールの到着の通知

   daemon などのあらかじめ指定した特定のユーザからのメールが届くとページャが鳴り、メールの到着を知らせる。また、From 行の最初 8 文字がページャに表示されるので誰から来たメールかを知ることができる。

2. NeWSprint サーバの暴走監視

   プリンタ用 PostScript 互換インタプリタである NeWSprint のサーバプログラム[3]が暴走した疑いがある場合にページャを鳴らす。実際には暴走したかどうかの判断はサーバのプロセスが cpu をあるパーセンテージ以上占有しているかどうかで判断している。この判断方法では NeWSprint サーバが単に複雑な処理を実行しているだけの場合にも誤動作してしまう可能性があるが、他に良い方法がみつからないのでこの方法を採用している。

神奈川大学では、管理する計算機が点在するなどの理由で管理者が部屋に不在がちであったが、本システムによって手近に端末装置がなくてもメールの到着やシステムの障害がわかり有用であった。

しかし、シリアルポートの排他制御などを行なわなかったので、複数のプログラムが同時にページャへメッセージを送ろうとする場合に問題がおこった。これを解消するために次のステップとして、WIDE/PCS を利用することにした。

---

[1] TTM NIP 型と略記する

[2] 到着したメールをヘッダ情報などで分類したり、コマンドを起動したりすることができる

[3] unix 上で動作する

3.3.2　第 2 期 1993 年 1 月 25 日以降

WIDE/PCS+AT モデム+TTM NIP 型ページャ

1993 年 1 月 25 日に神奈川大学平塚校舎内の Sun ワークステーションに WIDE/PCS をインストールし、既存のプログラムのページャ駆動部分を WIDE/PCS を利用するように変更した。その時点では、WIDE/PCS では TTM NIP 型ページャに非定型文を送る部分は完成していなかったため[4]、障害検出プログラム側で DTMF 文字列を作成し、WIDE/PCS を asis モード[5]で利用して、DTMF 信号を送出するようにしていた。

WIDE/PCS を導入したことにより、メールを使ってページャへメッセージを送ることが簡単にできるようになり、障害検出プログラムなどを書く場合は、特定アドレスにメッセージをメールで送るだけで良くなった。また、WIDE/PCS はキューイング機構を持っているため、複数のプログラムが同時にページャへメッセージを送ろうとしても問題なく利用できた。

また、電子メールからページャへメッセージを送れるため、インターネット上の知人にページャ用のメールアドレスを教え、必要な時に呼び出してもらうという利用方法も増えた。最近ではパソコン通信などからもインターネットにメールを送ることができるようになってきたので、メールによるフロントエンドは今後も重要なインターフェースとなるであろう。

なお、TTM NIP 型ページャはバイブレータが内蔵されているので授業中、会議中でも第三者に迷惑がかからずに便利であった[6]。しかし、自由に送れる文字情報が英数字とひらがなで 8 文字と少なく、NNT DoCoMo の自由文型[7]のワイドエリアタイプが入手できたので、NTT DoCoMo 自由文型に切替えた。

3.3.3　第 3 期 1993 年 2 月 26 日以降

WIDE/PCS+AT モデム+NTTI 型ページャ

NTTI 型ページャは定型文、イラストなどの他に JIS 第 1 水準の漢字、平仮名、カタカナ、英数字、記号などが自由に表示でき、一度に最大 36 文字まで受信することができる。

このため、TTM NIP 型に比べ、扱える情報量が格段に増える。第 1 期から動作させている NeWSprint サーバ監視プログラムでは、障害の起きたホスト名の他に「NeWSprint に障害」のようなメッセージやプロセス番号、cpu 占有率なども表示させることができる。また、メールの到着を知らせる場合には、内容の一部まで表示することが可能であるので、端末に向かわずにも内容がわかる場合もある。

この他には、ディアイティで試験的に運用している ping スクリプト[8]に神奈川大学の

[4] 残念ながら現時点ではまだ実装されていない
[5] 渡されたメッセージを加工せずにそのまま送出するモード
[6] NTTI 型にはバイブレータがなく、音を消すと光による合図のみとなる
[7] NTTI 型と略記する
[8] 詳細は後述

主要ホストを登録して、定期的にチェックするようになったのもこの時期である。現在チェックの対象となっているのは以下の通りである。

- kanagw: 神奈川大学の対外ルータ
- jingw: 神奈川大学のプライマリ DNS サーバ, 神奈川大学のニュースサーバ
- alpha: 神奈川大学のセカンダリ DNS サーバ, 平塚キャンパスのメールサーバ
- crimson: 平塚キャンパスのニュースサーバ, 実習用サーバ

NTTI 型ページャの問題点としては、先に述べたバイブレータ機能が無いこと、1 ケ月の利用回数が 150 回を越えると 1 回あたり 70 円の追加料金を徴収される契約となっていることなどがある。現在の料金体系では、障害検知プログラムで些細なことではページャを呼び出さないようにしたり、メールの到着通知でページャを鳴らすユーザを限定するなどしておかないと運用コストが増大することがある。これはページャの有効活用の妨げになると考えられる。今後この料金体系が改善されることを望む。

3.3.4 今後の予定

WIDE/PCS+PNC-3500+NTTI 型ページャ

現在まで、神奈川大学では DTMF 送出装置として専用 NCU(網制御装置) ではなく、AT モデムを利用してきていた。しかし、この場合は話中などでページャにメッセージが送信できなくても、常に問題なく送信できたように振舞うという問題点がある。

これは専用の NCU(網制御装置) を利用することによって回避できる。今後は、東京工業大学や (株) ディアイティなどで運用実績のある NCU である PNC-3500 を導入し、メッセージの到達性を高めていきたい。また、100 台以上ある演習用の Sun ワークステーションの監視を syslog や SNMP を利用して行なうことも計画している。

さらに、ページャとは直接関係無いが、WIDE/PhoneShell と音声合成装置、ファックスモデムなどの組み合わせで、学生向けの情報サービスなどを行ないたいと考えている。

3.4　iNET92 での運用

WIDE/PCS は、iNET92[9]の会場設営期間を含む会期中のスタッフ間連絡およびシステム監視手段として導入された。本節では iNET92 での運用形態、利用状況などについて述べる。

3.4.1　WIDE/PCS の需要

iNET92 は、多数のスタッフが参加し、ネットワークの運用だけでなく会議の運営方法の打ち合せ・会場設営・端末室の準備などを支援することになった。彼らが相互に連絡をとる機会は会期が近付くにつれて急増すると思われたが、会期直前になると端末の前で仕事をする機会が減少し、主たる連絡手段であるメールの利用が困難になると予想された。

また、会期中は端末室が開設されメールを読み書きする環境が提供されることになっていたが、スタッフがメールを読み書きする時間を十分確保できるかどうかが疑問であった。さらに計算機やネットワークに予期せぬ障害が発生した場合には、電子メールサービスが機能せず連絡がつかなくなる可能性もあった。

このような状況から、一部のスタッフは携帯電話を所持することになったが、スタッフ全員に配布するには台数が足りず、さらに会場近辺の建物内部では利用できない可能性があった。そこで WIDE/PCS を連絡機構として採用し、NTT の自由文型ページャを会場に 7 台、藤沢に 1 台配置した。藤沢にも用意したのは、会場のネットワークが藤沢に接続されていたためである。なお会場の神戸ポートアイランドは、ページャのサービスエリアから離れていたので、事前に動作試験を実施し建物内を含めて問題なく利用できることをあらかじめ確認した。

3.4.2　システム構成

iNET92 の会場には、第 2 章で報告した実験系 (図 2.2) のサブセットを設置した。実験系 との相違点は、以下の 3 点である。

1. WIDE/PCS の運用には直接必要ないモデム、ファックスアダプタ、コントロールユニットなどを削除した。
2. 実験系では、音声を全二重で送受するためにワークステーションを 2 台確保しているが、これを 1 台でまかなった。
3. 1 台のワークステーションに NCU を 2 台接続した。

また、東京に設置した実験系も利用した。これにより、最大で同時に 3 つの NCU を駆動して 3 つのページャに同時にメッセージを伝達できるようにした。

[9] 1992 年 6 月 14 日から 19 日まで神戸ポートアイランドで開催された。

3.4.3 利用状況

ページャの準備が整った 6 月 2 日から、会場での運用が終了した 6 月 18 日までの間に、関係者あてのメッセージ転送は 119 回発生した。その内訳を図 3.2に示す。

発信地点	→	相手先	回数
東京地区	→	東京地区	29 回
東京地区	→	阪神地区	42 回
阪神地区	→	東京地区	4 回
阪神地区	→	阪神地区	44 回
合		計	119 回

図 3.2: WIDE/PCS の利用状況

この他に各ページャに 20 種類のメッセージを事前に登録するための操作も WIDE/PCS を介して行なった。一つのメッセージを登録するたびに回線を一旦切らなければならない仕様になっているため、8 台のページャ全てにメッセージを登録するには、延べ 160 回の呼び出しが必要であった。

利用者の感想はおおむね良好であった。以前は、携帯電話が普及すれば WIDE/PCS のようなサービスは不要ではないかという危惧もあったが、ページャはそのサービス圏内であれば、携帯電話の利用が難しい建物の中や地下でも受信でき、また通電したままで 1 月以上利用可能であるといった長所があるため、使い方によっては携帯電話より有効に利用できるということがわかった。

なお、WIDE/PCS の初期のバージョンでは、正しく文字が転送できないとか一度転送に失敗し 2 度目に成功するという誤動作がまれに発生していたが、これらの問題点は iNET92 開催前までにすべて解決できた。このため、上記の 119 回の転送は全て問題なく行なわれた。

3.5　ping スクリプト

本節では、WIDE/PCS を用いたネットワーク監視システム (以下 ping スクリプト) について報告する。

一般に、ネットワークやホストに異常が発生したときに管理者が端末の前にいれば素早く対処できるが、管理者が常に端末の前に座っているとは限らない。このように管理者が持ち場を離れている状況で異常が発生した際に、ページャを利用して管理者に異常を伝えることができれば、トラブルの早期解決に役立つのではないかと考えたのが ping スクリプトを作ったきっかけである。ping スクリプトのプロトタイプは 1993 年 1 月に動き始めた。その後、PhoneShell ワーキンググループのメンバの協力を得て改良を重ね、現在に至っている。

3.5.1　基本仕様

現行の監視システムは cron によって定期的に csh スクリプトを起動することによって監視を行っている。設計当初の ping スクリプトの基本構造を図 3.3に示す。

```
foreach host (host1 host2 host3 host4)
    ping $host >& /dev/null
    if ($status) then
        echo "$host is down" | mail foo-pager
    endif
end
```

図 3.3: ping スクリプトの基本構造

このようにホストが icmp echo パケットに応答しなければ管理者のページャを呼び出すようになっている。このスクリプトは、SunOS の ping コマンドが引数で指定されたホストに対して icmp echo パケットを 1 秒間隔で送信し、20 秒以内に応答がなければ異常終了するように設計されていることを利用して作られているため、他の機種で動かすためには修正が必要と思われる。

3.5.2　運用

まず最初にこのような単純なスクリプトを用いてディアイティ社内のホスト数台と WNOC TOKYO の主要ゲートウェイである wnoc-tyo.wide.ad.jp の監視を始めた。監視は cron で ping スクリプトを 5 分に 1 回起動して行うようにした。運用を始めると、ホストがダウンしている間は 5 分に 1 度ページャが呼ばれてしまうという問題点が明らかになった。また、ホストがアップしたかどうかがわからないという問題も発覚した。これ

らの問題点を解決するために、最後に ping したときの状態をホスト毎に保存しておき、前回の ping の結果と今回の ping の結果が異なる場合だけページャを呼び出すように改良した。この改良により不要なページャの呼び出しを大幅に減少させることができた。

1993 年 1 月 11 日に PhoneShell ワーキンググループに ping スクリプトの運用開始をアナウンスし、自由文型ページャを所有している管理者がいる組織の主要なホストの監視を始めた。監視ホストは自由文型ページャの所有者数とともに増加し、3 月末時点では表 3.3に示すようなホストが監視対象となった。

組織名	監視ホスト数
wide.ad.jp	15 台
kanagawa-u.ac.jp	3 台
titech.ac.jp	4 台
u-tokyo.ac.jp	6 台
jus.or.jp	1 台
foretune.co.jp	3 台
fujixerox.co.jp	2 台
hitachi.co.jp	1 台
dit.co.jp	3 台

表 3.3: 監視ホスト一覧

wide.ad.jp のホストの多くは WNOC TOKYO に設置されているホストだが、国際回線のトラブル監視および WNOC SFC に接続されている富士ゼロックスのホストを監視するために WNOC SFC に設置されているホストも何台か監視している。

ping スクリプトによるネットワークの監視を始めた時点から 3 月末日までのディアイティの WIDE/PCS の利用状況を図 3.4に示す。なお、1 月 3 日の週のデータと 3 月 28 日の週のデータは 1 週間分のデータがないため、グラフの傾向を示す参考にはならない。

図 3.4の misc には ping スクリプトによるページャの呼び出しを除くすべての WIDE/PCS の利用が含まれている。misc の利用頻度は自由文型ページャ所有者の増加に比例して増加する傾向が見られた。

同様の理由で、ページャ所有者の増加に伴い ping スクリプトが監視する組織も増加したため、ローカルトラブルに示されている各組織内のホストのトラブルにも増加の傾向が見られた。1 月 10 日の週にローカルトラブルによるページャの呼び出しが 40 回/週を超えているのは、運用開始当初の ping スクリプトの欠点のため、1 回のトラブルで何度もページャを呼び出してしまったためである。1 月 17 日の週にローカルトラブルによるページャの呼び出しが多いのは、1 月 23 日に発生した wnoc-tyo.wide.ad.jp の gated 関連のトラブルが原因である。ping スクリプトは経路情報に関するチェックを一切行っていないため、本来ならばバックボーン関連のトラブルである wnoc-tyo.wide.ad.jp のトラブ

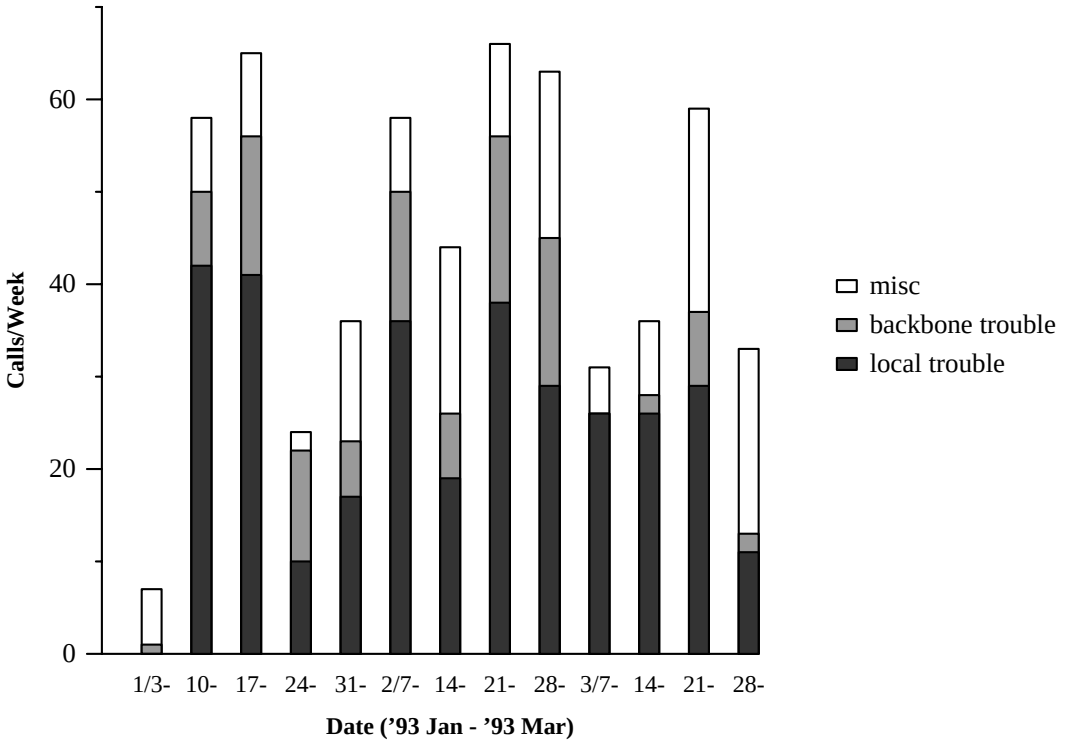


図 3.4: WIDE/PCS の利用状況

ルを各組織のローカルなトラブルであると勘違いして、ping スクリプトによって監視を行っているすべてのページャを順次呼び出してしまった。

グラフを見てもわかるように、WIDE バックボーン関連のトラブルの発生はページャ所有者の増加に比例していない。これは、現在のところバックボーン関連のトラブルが発生した際に呼び出されるページャが 1 台しかないためである。バックボーン関連のトラブルによるページャの呼び出し回数は実際にトラブルが発生した回数とほぼ一致している。実際、グラフ中でバックボーン関連のトラブル発生が多く報告されている 1 月 17 日の週や 2 月 21 日の週には、バックボーンに比較的大きなトラブルが発生している。

3.5.3　評価

結論から先に述べれば、ping スクリプトの運用効果は十分あると言える。ping スクリプトの運用を開始した 1 月から 3 月までの約 3ヵ月間に、WIDE バックボーン関連だけでも gated のクラッシュやルータの熱暴走などによるトラブルに何回か遭遇しているが、ping スクリプトによる監視のおかげでいずれのトラブルも速やかに解決することができた。代表的なトラブルの例を表 3.4に示す。

平日の昼間にトラブルが発生した場合は、ページャの知らせを受けなくてもトラブルに気がつくことが多いだろう。しかし、ここに挙げた例でもわかるように、トラブルの

日時	トラブル内容および対処
01/23/93 00:30	wnoc-tyo.wide.ad.jp の gated が正しい経路情報をアナウンスしなくなり、WNOC TOKYO に接続されている組織がインターネットにアクセスできなくなった。深夜のトラブルのため、トラブル発生直後に WNOC TOKYO に行くことはできなかったが、翌朝 NOC にて対処した。
02/06/93 14:06	wnoc-tokyo-cisco.wide.ad.jpがハングアップした。ページャが鳴ったときは会談中で NOC に急行することはできなかったが、幸いなことに NOC がある岩波書店ビルに WIDE のメンバーがいたので電話で対処を依頼した。トラブルの原因はマシンルームの空調が止まりマシンルームの温度が上昇したことによる熱暴走と思われる。
02/07/93 04:00	2/6 に引続き wnoc-tokyo-cisco.wide.ad.jp がハングアップした。日曜日の早朝のトラブルのため、朝まで待って NOC にて対処した。原因は前日と同様に熱暴走と思われる。
02/23/93 19:36	ネットワークに直接アクセスできない場所でミーティングをしている最中に jus.or.jp からの応答がなくなったという知らせが届いた。ミーティングに居合わせた他の WIDE メンバのページャにもトラブルを告げるメッセージが続々と表示された。WNOC SFC で動いている WIDE Voice Information Service に電話して主要なホストを ping してトラブルの原因は wnoc-tyo.wide.ad.jp の gated だとほぼ断定できたが、その場では対処できないので東京大学へ電話して状況を説明して gated を再起動してもらうことによって解決した。

表 3.4: バックボーントラブル事例

多くは管理者が端末の前にいない可能性が強い土日や早朝・深夜に発生している。ページャの呼び出しを受けた管理者が直ちに端末の前、あるいはトラブルが発生している現場へ急行できれば申し分ないが、管理者が常にトラブルに対処できる状況にいるとは限らない。このような場合でも、他の管理者へトラブルを知らせて対処させたり、あるいは対処ができなくてもトラブルの発生をアナウンスさせることによって、二次災害の発生を未然に防ぐことができる。このような点だけを見ても、ページャによってトラブルの発生を知らせることは意味のあることだと言える。

3.5.4　今後の課題

ping スクリプトによるネットワーク監視システムがトラブルの早期発見/対処に効果があることは間違いないが、現在のシステムはまだ多くの問題点を抱えていることも事実である。ここでは、現状の ping スクリプトが抱える問題点を考察し、今後の課題を検討する。

まず、現状のシステムが抱える大きな問題点として、ホストがアップ/ダウンを繰り返す状況 (いわゆる YoYo モード) に十分に対応できないという点がある。システムの設定を変更したり、ネットワークのトポロジを変更したりする場合には、頻繁にホストの reboot を繰り返す場合が多い。このような場合や、何等かのトラブルによりシステムが定期的に reboot している状況では、その都度ホストのアップ/ダウンを報告するのではなく、ホストの状態が不安定であることを管理者に伝えるのが望ましいが、現状の ping スクリプトの実装ではこのような状況に対応できない。現状の ping スクリプトはホストが頻繁にアップ/ダウンを繰り返した場合に逐次報告しないようにするために、一度ページャを呼び出したら 120 分間は同じ報告はしないようになっているが、動作が不安定な状況を判断して管理者に伝えられるようにするべきだろう。

ping スクリプトはその名前からもわかるように、ホストがアップしているかどうかを知るために icmp echo パケットに対する応答を調べている。この方法は同一セグメント上のホストに対してはかなりの確度で信用することができるが、広域インターネットのように経路上に複数のゲートウェイが介在するような状況では、目的のホストがダウンしているのか途中の経路上のゲートウェイがダウンしているのかを判断するのが難しい。このため、現在の ping スクリプトはアップしているかどうかを調べたいホストまでの経路を明示的に指定し、もし経路上のゲートウェイがダウンしている場合には目的のホストに icmp echo パケットを送らないようになっている。この方法だと経路の変更に対応することができない。つまり、もし目的のホストに到達する経路が 2 通り以上ある場合、現在の ping スクリプトは 1 通りの経路しか指定できないため、目的のホストに到達できないと解釈してしまう。経路の変更に対応できる柔軟な設定ができるように改良する必要がある。

以上の 2 点が、現在の ping スクリプトが抱える大きな問題点だが、この他にもいくつか改良の案が出されている。たとえば、現在の ping スクリプトはトラブルが発生すると昼でも夜でもページャを呼び出してしまうが、深夜にトラブルが発生した場合には状況に応じてページャの呼び出しをやめたり、あるいは朝まで待って呼び出すなどの工夫が必要と思われる。また、停電や定期メンテナンスのように予めダウンすることがわかっている場合には、ping スクリプトにダウンスケジュールを教えることによってページャの呼び出しを抑止するような仕組みも必要だろう。

現在 ping スクリプトはディアイティでしか動いていない。複数の組織で ping スクリプトが動くことを想定した場合、ping スクリプト間で情報交換をすることによって、効率のよい監視ができるようになるかもしれない。

このように ping スクリプトには改良の余地が多分に残されているが、シェルスクリプ

トで記述されているため機能拡張に柔軟に対応するのが難しい。拡張性に富み、様々な状況に対応できる柔軟なシステムにするためにも、C 言語や C++言語によるシステムの全面的な作り直しを検討するべきだろう。

第 4 章

WIDE/PhoneShell の今後の展開

本章では、WIDE/PhoneShell とりわけ WIDE/PCS の発展を意図して導入が検討されている項目のうち、すでに具体的な検討に入っているものをいくつか紹介する。すなわちここで取り上げる話題は、いずれも検討中か試験運用中で実際のネットワーク管理にはまだ利用していないが、来年度の早い時期に現場に投入可能となるものばかりである。

具体的には以下の話題を取り上げる。

- ping スクリプトを拡張した pingd の設計
- ネットワーク管理用ワームの利用
- uucp や DNS を利用した WIDE/PCS 専用バックアップリンクの確保
- BIND による WIDE/PCS 情報の管理
- X.500 による WIDE/PCS 情報の管理

4.1 WIDE/pingd

前章で述べたように、ping スクリプトは非常に単純ではあるが、WIDE/PCS との組合せにより、かなり効果的な使い方ができることが分かった。しかし、C シェルのスクリプトによる記述には限界がある。

ping スクリプトは、ping を用いた検査という点では実用レベルに達している。しかし、情報不足や結果の系統立てたとりまとめが出来ないために、正確な判断ができないことがある。

たとえば、ping スクリプトでできるのは、「トラブルかどうか分からないが、とりあえずページャを呼んでみる」もしくは「トラブルがおきているようだが、複数回呼んでしまうのはまずいので、とりあえず呼ばない」といった処理であるため、トラブルが通知されないよりは通知された方がよいという判断にもとづき、現時点では本当にページャを呼び出さなければならない状況よりも多くの呼びだしを行なっている。このような問題を解決するため、pingd を開発している。

pingd には以下のような特徴がある[1]。

4.1.1 ホスト情報・管理者情報

pingd では、対象とするネットワーク上のオブジェクトそれぞれに対して、付随する情報を定義できる。組織、管理者、ホストはすべてオブジェクトの例である。

そのなかで核となるのが、管理者名とコンタクト方法の記述である。たとえば、管理者がだれであるのか (もしくは、root 権限を持つのがだれであるのか)、その管理者のページャアドレス、メールアドレス、そして、呼出方法に対する条件づけ (呼出プレファレンス – 後述) などである。pingd はこれらの情報を元に、適切な方法で管理者にコンタクトを試みる。

加えて、管理を助けるであろう付随情報も記述することができ、さらに、記述を容易にするために、省略名の指定をすることができる。以下に例を示す。

```
# ドメイン vaporware についての情報
info domain vaporware.co.jp "vaporware"
        admin list adminstaff@vaporware.co.jp
        office phone "03-xxxx-xxxx"
        office fax "03-xxxx-xxxx"
        primary gateway vaporgate
end info

# 管理者 taro@vaporware.co.jp についての個人情報と
# 呼び出しプレファレンス
```

[1] 現時点で開発中のため、細かい記述については変更される可能性がある。

```
info person taro@vaporware.co.jp "taro" at vaporware
        JPNIC-handle TT999JP
        pager taro-p@vaporware.co.jp
        mail  taro@vaporware.co.jp
        emergency phone "030-xx-xxxxx"
        call preference {pager, mail, office phone, emergency phone}
end info

# ホスト vics の管理者についての情報
# ホスト情報で定義されていない場合は、ドメイン情報が参照される
info host vics.varporware.co.jp "vics" at vaporware
        emergency contact       taro
end info
```

4.1.2　ホストグループとネットワーク

ping スクリプトでは、それぞれのホストについて icmp echo を送り、reply を得るという作業を繰り返すことによって、ネットワークやホストが正常に動作しているかどうかを判断していた。

pingd では一歩進めて、一つのホストへの到達性を確保するのに必要なホスト群をひとまとまりとして取り扱えるようになっている。例えば、vics.vaportware.co.jp への wnoc-tyo からの到達方法は以下の方法で記述できる。

```
# ホスト vics へは、ゲートウェイ wnoc-tyo, vaporgate を経由する。
# target は現在対象となっているホストを指す。
info of reachability for host vics
        route wnoc-tyo vaporgate  target;
end info
```

4.1.3　フィルタリングと管理者呼出プレファレンス

条件を組み合わせることにより、呼出条件を絞ることも可能であるが、それらの条件のうちかなりの部分はきまりきったパターンをとっている。

例えば、海外リンクが落ちた場合、リーフノードの管理者はその情報を知りたいが、就寝中にページャに呼び出されるのは避けたいはずである。しかし、海外ゲートウェイの管理者は状況によっては対応すべきかもしれない。このような場合、リーフノードの管理者にはメールで通知が行き、ゲートウェイの管理者にはページャを用いて通知されるのが妥当であろう。

また、複数の管理者がいる場合には、障害の種類障害発生時刻に応じて告知する相手を切替えると言う「フィルタリング機能」が必要であろう。たとえば、マネージャクラス

の管理者であれば、一般には障害が起こったという事実を知るだけで充分で、対処要員（もしくは、当直要員）は迅速な告知を求めるだろう。緊急な場合には、マネージャも告知されることを望むかも知れない。

このように、呼出プレファレンスは、ある条件を満たす障害が起きた場合に、どのような方法で、どの管理者に告知するのかを指定できるメカニズムである。

これは先に示した記述のうち、“preference” 指定で記述している部分がこれにあたる。

4.1.4 容易な設定変更

ping スクリプトは対象となる複数のホストを実際に ping するスクリプトを書くことで設定変更を行なう。このため、設定変更をするには、実際に ping するスクリプトを記述する必要がある。

pingd は、設定ファイルを書くことによって設定を行なうので、ping スクリプトより簡単に見通し良く設定することが可能である。

典型的なパターンのチェックについては、より簡単な記法を用意する。また、典型的なパターンのレポート方針はデフォルトルールとして組み込みでもち、適切な解釈が行なわれない場合はこのデフォルトルールが用いられるので、ホストごとに細かい設定をする必要はない。

4.1.5 複雑な条件の記述

ping スクリプトで複合条件を記述するには、shell の if-then-else 構文を使って適時処理を記述してゆく方法しかなかったが、pingd の設定ファイルでは、if-then-else だけでなく、簡単な条件式を記述することができるので、ある程度複雑な条件における処理も比較的簡単に記述可能である。

例をあげると、

```
# vics が到達できない場合、regular_contact 手続きを呼ぶ
on unreachable vics
        do regular_contact
end on

# 第一ゲートウェイがダウンしているか、対象ホストがダウンしている
# 場合、メンテナンス要員をページャで呼びだし、かつ、報告メール
# をマネージャに送る
on regular_contact
        if target down or primary gateway down
                page to maintainer
                report to manager
    end if
```

```
end on
```

4.1.6　並行処理

ping スクリプトは、C シェルスクリプトにより書かれているため、以下のような問題点があった。

第一に、対象となるホストを順番に ping し、かつ、複数回呼出を避けるために 1 分単位の時間をおいてから、再試行を行なっているため、一連のテストが 10 分程度では終らないことになり、そのため、単位時間あたりに ping スクリプトを実行する回数に制限が生じる。たとえば、スクリプトを実行するのに 10 分かかるのであれば、10 分に一度しかテストすることができない。

第二に、ある瞬間におけるネットワークの状態を知るのが難しいという問題がある。たとえば、二つのホストに対する ping の間にネットワークの状態が変わってしまうと、トラブルが発生したのは分かるが、どこで起こったのかという情報を得にくい場合がある。

pingd は、複数のホストグループに対して並行動作をする。これにより単位時間当たりの実行回数に制限がなくなり、ネットワークの状態を把握しやすくなり、迅速かつ的確なレポートが可能になる。

4.1.7　拡張性

ping スクリプトは、ping プログラムを実行し、その実行結果 (exit code) によって対象ホストが正常に到達可能かどうかを判断するという単純な方法を採用している。しかし、この方法では、traceroute のような複雑な出力をするプログラムの実行結果を利用することができない。

pingd は C で書かれているため、ping スクリプトと比して、多くの機能を組み込むことができる。現時点では、以下のような管理情報を入手可能にする方針である。

- routing 情報の取得 (netstat -r に相当) どのようなルーティングが来ているか。

```
if no route to default
  ...
end if
```

- interface 情報の取得 (ifconfig, netstat -i に相当) および、incoming/outgoing packet の推移。

```
if interface vics-fddi.vaporware.co.jp down
  ...
end if
```

- ripquery – ルーティングと、それぞれのメトリック値。

```
if metric of route to u-tokyo >= 3
  ...
end if
```

- telnet(または sendmail) ポートへの TCP アクセス

  TCP のコネクションを該当するホストに対して試行し、そのホストが生きているかまたは、inetd が正常に動作しているのかを知る。

```
if trytelnet vics fail
  ...
end if
```

- syslog の内容チェックまたはそれに相当する機能

  当該ホストの状態チェックに有効。また、loghost であった場合は、他のホストの情報も見ることができる。

- gated.log の内容チェックまたはそれに相当する機能

  routing の up/down, interface の up/down を gated がどう見ているのかを判断できる。

- 適当なホストに対する SNMP Query

  SNMP を実装しているホストまたはゲートウェイからのいろいろな情報のとりだしと設定変更。

さらに、これらの情報を取り出した後に WIDE/PCS による呼出を行なうことが出来るだけでなく、他の告知機能を利用できる枠組も用意する。現時点では以下のようなものがあげられるだろう。

- ページャ呼出

```
page maintainer
```

- メール告知

```
mail maintainer "FILENAME"
```

- write/msend によるターミナルを介しての告知

```
notify maintainer "default route vanished!"
```

- WIDE/PhoneShell による告知

```
phoneshell maintainer "dialogfile""
```

4.1.8　デーモン・プロセスとして動作

ping スクリプトは、cron によって定期的に実行される形をとっていたが、pingd は常時デーモン・プロセスとして動作する形をとり、複数のホストグループに対するチェックを常時並行して行なうので、ホストグループ毎にチェックする頻度を明示的に指定できる。また、それぞれのホストの状態を常に保持しておくことが可能なので、ホストグループの状態変化にかなりのレベルで追従できる筈である。

たとえば、安定したホストグループであれば比較的長い間隔でチェックすれば充分であろうし、不安定なホストがある場合は逆にチェックを強化することも可能であろう。

4.1.9　今後の予定

ping スクリプトは単純な方法であったが、ネットワークの管理を効率良く、ネットワークダウンしている時間を減らすことができた。pingd はこの経験を踏まえて述べて来たような機能を盛り込む予定であり、よりよい管理環境を提供するであろう。

現在、細かい要求仕様のとりまとめを行ないながら一部コーディング作業を始めており、数カ月以内には本格的な運用を開始する予定である。

4.2 ネットワーク管理ワーム

前節で述べた pingd は、ある固定されたホストから他のホストやネットワークを調べるもので、いわば定点観測のツールである。これに対して、管理エージェントがホスト間を移動しながらネットワークの状態を調べる方式が考えられる。本節ではその一例として、ネットワークワーム (以下ワーム) をとりあげる。

4.2.1 ネットワークワーム

本節で述べるワームとは、プログラムの一種であり、ネットワークを通して複数の計算機間をプログラム本体が移動するものを指している。コンピュータウィルスと同種のものとされることもあるが、ここではウィルスのような悪い意味を含まない。

ネットワーク管理に WIDE/PhoneShell とワームを併用すると、以下のような効果が生じる。

- ネットワーク上を移動しながらの調査、分析、制御が可能となる

  ワーム自身が調査、分析、制御しながらネットワーク上を移動できる性質を利用する (図 4.1)。例えば、WIDE/PhoneShell のみではホスト間の設定の関係を調べて、矛盾を発見するのは困難な作業であるが、ワームを用いれば容易に設定を照合して矛盾を発見することができる。

- 管理作業に必要な通信量を削減できる

  WIDE/PhoneShell は入力を DTMF、出力を音声合成またはページャで行なうため、送ることのできる情報量が限られているという欠点がある。ワームを利用して本当に必要な情報のみを収集すれば、通信量を効果的に減らすことができる。

- 障害が起きた際の通知が容易になる

  ワームは、作業の終了や障害の検知などを管理者に WIDE/PCS を用いて伝えることができる。したがって、管理者はワームの作業が終るまで端末の前で待っている必要がない。この応用として、常にネットワークを監視しているワームを実行しておき、障害が起きたら WIDE/PCS によって管理者に知らせるシステムが考えられる。

  またワームが障害を検知して管理者に知らせようとした場合、ネットワークに発生した障害のために管理者のいるホストに到達できないことがある。このような場合、もよりの WIDE/PCS ホストと通信し、ページャによって異常を知らせることができる (図 4.2)。

以上のように、ワームを WIDE/PhoneShell と連係させたネットワーク管理は有効であると予想される。ここではワームによるネットワーク管理システムの一つとして、現在実装中の NMW-System (Network Management Worm System) [178] を取り上げる。

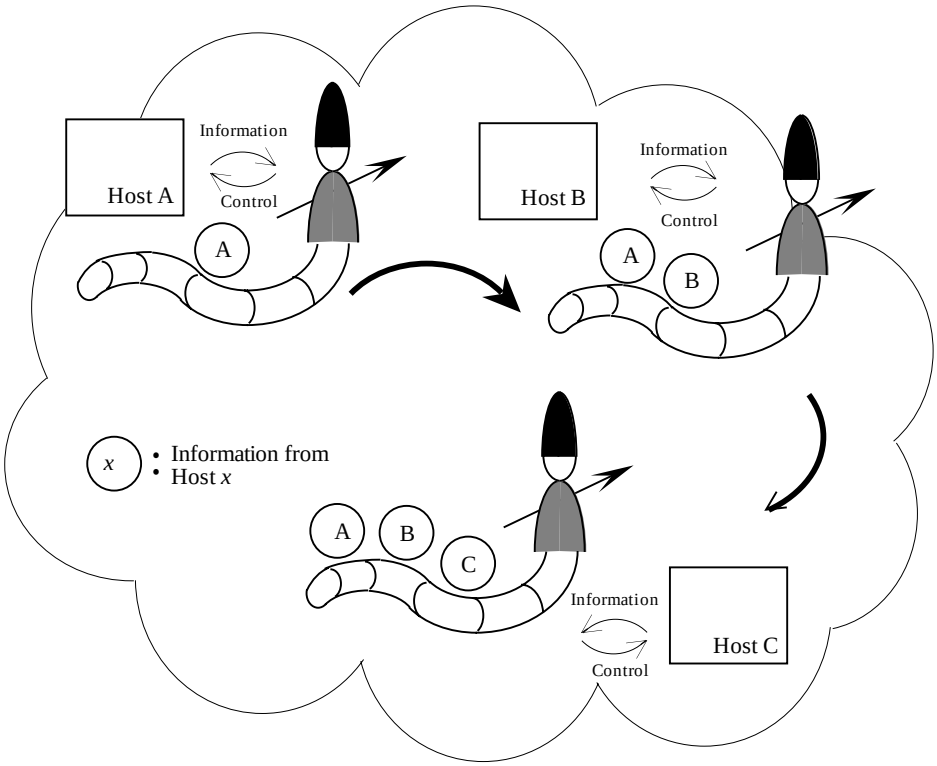


図 4.1: 計算機間での移動に伴う調査、分析、制御

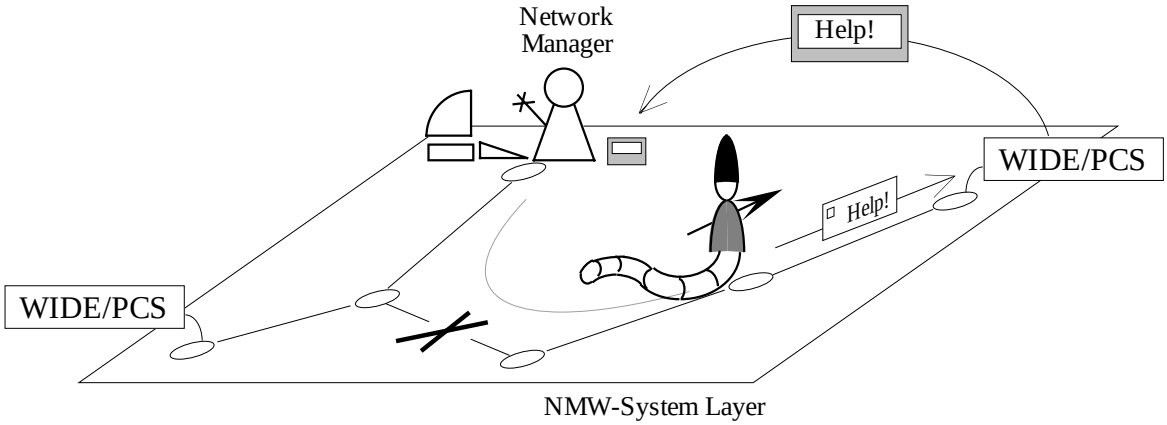


図 4.2: WIDE/PCS による異常の通報

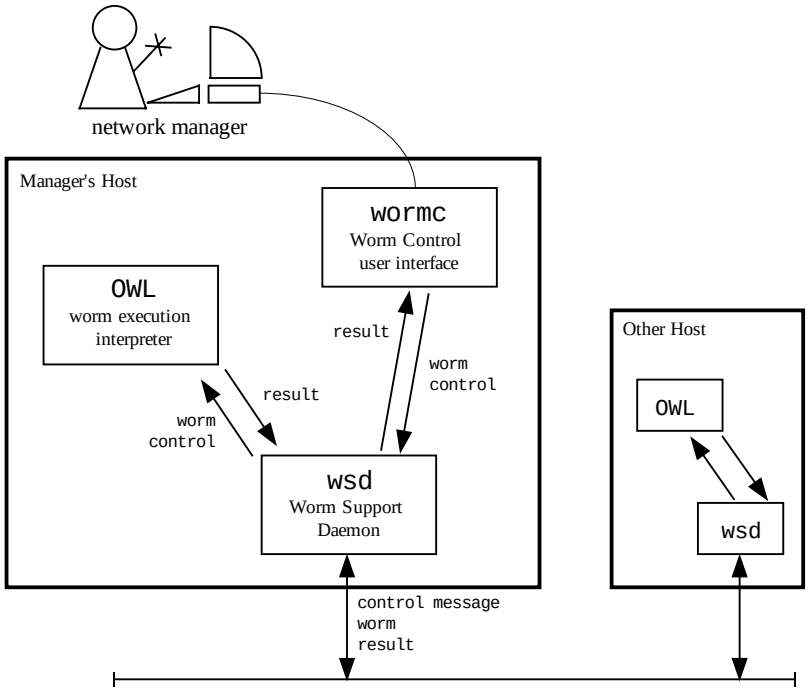


図 4.3: ネットワーク管理ワーム支援系の構造

4.2.2 NMW-System の構造

NMW-System は、ワームとワーム支援系から構成される。ワーム支援系には以下のような 3 つのサブシステムがある。

owl：ワーム実行インタプリタ
ワームを実行するインタプリタ (OWL-interpreter)

wsd：ワーム支援デーモン
ワームや制御メッセージの配送、結果の回収などのホスト計算機間の通信を行ない、owl を制御するサーバ

wormc：ワーム制御インタフェース
システムを制御するためのユーザインタフェース

これらのサブシステムの関係を図 4.3 に示す。

4.2.3 NMW-System の特徴

NMW-System の特徴としては、4.2.1節で述べたものの他に以下のようなものが考えられる。

- 対話的な操作が不要
  ワーム自身がネットワークの状態を調べて作業内容を判断できるので、作業中の操作が不要である。

- 応用範囲が広い
  本システムのワームはプログラムできるので、プログラムを変更することでさまざまな応用が考えられる。

- 新しい機能を加える際に、すべての計算機のプログラムを変更する必要がない
  サーバクライアント方式のサービスでは、新しい機能を加えるためにはサーバのプログラムを変更しなければならない。したがってサーバが多数ある場合は、すべてのサーバに新機能を追加する作業は多くの時間を必要とする。ワームを用いれば、ワームプログラムの変更のみでシステムに新しい機能を加えることができる。

- ネットワークや OS のアーキテクチャから独立
  ワームの転送にはネットワークが提供するサービスをそのまま利用するため、ネットワークや OS のアーキテクチャによらずに本システムをインストールできる。

- ネットワークの経路制御情報が失われていても転送可能
  (4.2.4節参照)

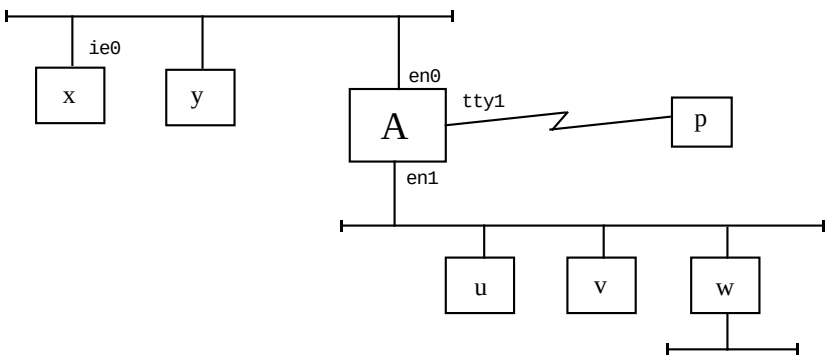
4.2.4　ネットワークの経路制御情報が失われている場合のワームの転送

NMW-System は、ネットワークが正常に動作していない場合でも可能な限り動作することを目指している。そのような場合の一例として、ネットワークの経路制御情報が失われた場合を考える。

4.2.4.0.1　隣接ホストグループ　あるホストから見て、ネットワークのハードウェアが直接つながっているようなホストを隣接ホストと呼ぶ。この隣接ホストのうち、同じインターフェースにつながっている隣接ホストの集合を隣接ホストグループと呼ぶ。図 4.4 に例を示す。

IP ネットワークにおける経路情報とは、同じネットワークに継っていないホストに送るデータグラムを、同じネットワークのどのルータに送れば良いかを示すことが主な目的であり、同一ネットワークにつながっている隣接ホストと通信する際には経路情報は必要ない。したがって、 ネットワークの経路情報が失われた場合でも、隣接ホスト間をたどってワームが移動できる可能性は高い。この方法でワームが目的のホストへ到達する方法には、以下のようなものがある。

1. ワームにあらかじめ経路を教えておく
   到達すべきホストが固定されており、到達するための経路もほぼ定まっている場合には、ワームにあらかじめ経路を教えておけば良い。



ホスト A の隣接ホストグループ： en0: { x y }
en1: { u v w }
tty1: { p }

ホスト x の隣接ホストグループ： ie0: { x y A }

図 4.4: 隣接ホストグループの例

この方法の応用例として、次のようなものが考えられる。経路情報が失われやすいネットワークを pingd で監視している場合、そのネットワークから反応がなくなった時に、単に経路情報が失われただけなのか、監視対象のネットワークへの経路が切れたのか、監視対象のネットワーク自身に障害が起こったのか不明なことがある。このような場合には、あらかじめ経路を教えておいたワームによって、途中の経路と相手先のネットワークを調べることができる。

2. ワーム自身が探索する
目的ホストへの経路を知らない時の方法の１つとして、ワーム自身が探索する方法がある。あるホストに到着したワームは、そのホストの隣接ホストグループを調べて行き先を決定する、または隣接ホストすべてにワーム自身をコピーすることができる。

3. NMW-System 独自の経路制御を行なう

もう一つの方法として、NMW-System 独自の経路制御を行なうことが考えられる0。

目的ホストへの経路がわからない時、そのホストを宛先とする制御メッセージ RRM[2] を NMW-System 全体に流す。該当のホスト、または宛先のホストへの経路が確立可能なホストは RRM を受けとると、経路制御の制御メッセージ RIM[3] を出す。ワームは RIM が来る方向に向かって隣接ホスト間を移動して目的ホストに到達すること

[2][178]Route information Request Message
[3][178]Route Information Message

ができる。

4.2.5　今後の予定

NMW-System はまだ実装中であるため、まだ PhoneShell や WIDE/PCS との連係実験を行なっていない。しかし、4.2.1で述べたように WIDE/PhoneShell にとっても大変有効であるので、NMW-System の実装が済み次第、連係させた実験を行なう予定である。

4.3 障害発生時の WIDE/PCS の利用

4.3.1 uucp によるバックアップ・リンクの利用

現在 WIDE/PCS は、ネットワーク並びにホストの障害発生を管理者に伝達する手段としての利用度が高い。そのため、障害発生時でもそれを回避してページャにメッセージを送る手法の検討は極めて重要である。

本節では、専用線等を用いた通常の IP の経路に障害が発生した場合に、uucp 等を利用したバックアップ経路による WIDE/PCS の効果的な運用について説明する。

4.3.2 WIDE/PCS ホストの系列化

障害の発生時には、ページャを所有している管理者 (以下では単に受信者と呼ぶ) に対して、一般利用者 (以下では送信者と呼ぶ) や監視プログラムからのメッセージが送信される。ページャの利用が必要とされるのは緊急度が非常に高い場合であるので、ネットワーク・トポロジや発生している障害によらずに、送信者から受信者にメッセージが透過的に配送される必要がある。

障害を回避しつつページャを透過的に利用するためには、以下の二種類の動作が障害発生時にも円滑に行われる必要がある。

- WIDE/PCS が稼働しているホストの情報の取得
- WIDE/PCS が稼働しているホストへのメッセージの転送

このうち、前者は DNS 等の分散データベースを利用し、セカンダリ・サーバを適当に分散し、情報を冗長に保持することにより実現可能である。例えばセカンダリ・サーバを WNOC ごとに用意すれば、最低限 WNOC までの到達性が保証されれば、必要な情報が獲得できる。

一方、後者においては、WIDE/PCS をできる限り多くの組織で稼働させることが望ましい。しかし、現実には設備や機器などの問題により実際には制限がある。そこで、ネットワークに重要な障害が発生した場合でも、uucp 等による緊急的な経路を用意することで、WIDE/PCS が稼働しているサーバまでの到達性を確保することを実現する。

このために、WIDE/PCS サービスに参加しているサーバ群をプライマリとセカンダリに分ける。

プライマリ WIDE/PCS サーバ: WIDE/PCS をインストールしたサーバで uucp リンクを持つ。

セカンダリ WIDE/PCS サーバ: WIDE/PCS をインストールしていないがプライマリ WIDE/PCS サーバと uucp リンクを持つ。

これは、WIDE/PCS が稼働していないホストから uucp を利用することにより、WIDE/PCS と接続ができるような緊急用の経路を用意することに相当する。セカンダリ

WIDE/PCS サーバに対して、例えば DNS を利用する場合には、低いプリファレンスを持つ MX レコードを記述する。送信者から送り出されたメッセージは、試行を繰り返すうちに、プライマリあるいはセカンダリのいずれかの WIDE/PCS サーバに到達し、最終的に受信者に配送される。

ただし、このためには WIDE/PCS システム全体が pager.wide.ad.jp というようなドメイン形式で仮想化されている必要がある。この仮想化については次節で説明する。

4.3.3　実際の障害回避

これまで述べた設定を前提として、障害を回避する手順を以下に示す。

- 発信ホスト (送信者のホスト) は pager.wide.ad.jp ドメインの MX レコードに従いプライマリ WIDE/PCS サーバ、もしくはセカンダリ WIDE/PCS サーバにメールを送ることを試みる。
- メッセージを受けとったセカンダリ WIDE/PCS サーバは、IP により直接的に pager.wide.ad.jp ドメインのプライマリ WIDE/PCS ホストにメールを送ることを試みる。
- プライマリ WIDE/PCS ホストへ接続できなかった場合は、そのメッセージはセカンダリ WIDE/PCS サーバの sendmail queue に保持される。その後 uucp によりプライマリ WIDE/PCS サーバへの uucp による配送が試みられる。

セカンダリ WIDE/PCS サーバでは、uucp によるプライマリ WIDE/PCS サーバへのメッセージ配送のために、以下の設定を行う。

1. プライマリ WIDE/PCS サーバとの間に uucp リンクを設定する。
2. xxxx@pager.wide.ad.jp 宛のメールをプライマリ WIDE/PCS サーバに uucp 経由で配送するルールを sendmail.cf.uucp に記述する。
3. sendmail.cf.uucp をコンフィギュレーション・ファイルとして sendmail queue に保持されているメールのうち、pager.wide.ad.jp を宛先アドレスに含むメールのみを対象として sendmail を定期的に起動する cron の設定を行う。
4. プライマリ WIDE/PCS サーバに対して、uucico を用いて接続するスクリプトを設定する。さらに、定期的に、あるいは、uucp キューに転送すべきジョブがあるときに接続スクリプトが実行されるように cron を設定する。

cron により起動する sendmail のパラメタ形式と sendmail.cf.uucp の例を以下に示す。

```
/usr/lib/sendmail -Rpager.wide.ad.jp -C/usr/local/wpcs/sendmail.cf.uucp -q
```

```
# sendmail.cf.uucp
# sendmail.cf for WIDE/pcs mail transfer through uucp-link

# uucp peer host
DRwpcs-host

DVwpcs-uucp
DnMAILER-DAEMON
DlFrom $g  $d
Do.:%@!^=/[]
Dq$?x $x$.<$g>
De$j Sendmail $v/$V ready at $b

OA/etc/aliases
Odbackground
OD
OF0600
OL9
Oo
OQ/usr/spool/mqueue
Or30m
OS/etc/sendmail.st
Os
OT1h

# Message precedences
Pfirst-class=0
Pspecial-delivery=100
Pbulk=-60
Pjunk=-100

# Trusted users
Troot daemon uucp

# Format of headers
HReceived: by $j ($v/$V)
        id $i; $b
H?D?Date: $a
H?F?From: $q
```

```
H?x?Full-Name: $x
HSubject:
H?P?Return-Path: <$g>
H?M?Message-Id: <$t.$i@$j>

# Mailer specification
Mlocal, P=/bin/mail, F=rlsDFMmn, S=10, R=20, A=mail -d $u
Mprog,  P=/bin/sh,   F=lsDFMe,  S=10, R=20, A=sh -c $u
Muucp,  P=/usr/bin/uux, F=msDFMhuP, S=10, R=20, M=100000,
        A=uux - -r $h!rmail ($u)
# Rule set
S3
R$*<$*>$*               $2
S4
R$*<$*>$*               $1$2$3                  defocus
S0
R$*@pager.wide.ad.jp    $#uucp $@$R $:$1@pager.wide.ad.jp
R$+                     $#local$:$1             to local
S10
S20
```

ここで、uucp により最初の uucico 接続の対象となるプライマリ WIDE/PCS サーバに接続できなかった場合、他のプライマリ WIDE/PCS ホストに対して、正常にメッセージを受け渡せるまで順次 uucp によるメールの配送を試みる。このとき、第 2 番目以降のプライマリ WIDE/PCS サーバへの配送には、新たなサーバに対する uucp のコントロール・ファイルの生成が必要である。これは、最初の uucp のコントロール・ファイル中の宛先、メッセージ本体をもとに、sendmail.cf.uucp の転送先ホスト名を書き替えたコンフィギュレーション・ファイルを用いた sendmail の起動により実現される。

4.3.4　まとめ

本節では、送信者から WIDE/PCS サーバまでのメッセージの伝達経路に冗長性をもたせることにより、ネットワークの障害発生時においても使用に耐えうるシステムとして WIDE/PCS を運用する手段を説明してきた。

一方で、WIDE/PCS サーバからページャへのメッセージ送信の信頼性も高める必要がある。これに関しては、モデム障害等検出の容易なもののほか、DTMF によるページャへのメッセージ送信時の障害にみられるように、検出自体が困難なものとがある。後者に関しては、ページャ自体の改善項目として別節で述べた内容の提案を NTT に対して行なった。

4.4 BIND サーバによる WIDE/PCS 運用情報の管理

BIND(Berkeley Internet Name Domain) サーバはドメインによる名前空間を管理するための、Global Internet 上で普及している分散データベースサービス DNS の実装の一つである。以下 BIND サーバを単に BIND と呼ぶ。本節では、この BIND を利用して WIDE/PCS が必要とする情報を管理する方法について考察し、その実現形態について提案する。

4.4.1 BIND サーバ

一般に BIND を用いた情報管理では、以下のような特徴を有する。ここで+は利点、−は欠点を表している。

+ BIND は分散データベースの一種として最も広く利用されており、分散データベースのプロトタイプに用いるには有効である。

+ BIND は IP を利用した Internet 上でドメインに基づく名前空間に関する情報を管理するものであり、電子メールや TCP/IP を利用したアプリケーションとの連携が容易である。

+ BIND は全体として冗長に情報を管理しているため、ネットワークの障害に対して比較的頑丈である。そのため、ネットワーク管理のために必要な情報も管理できる。

- BIND は比較的静的なデータ管理に向いており、データの動的な更新や頻繁な更新には向かない。

- 現行の BIND にはアクセス制御の機能がないため、全てのデータが公開情報となる。

- BIND では “.” をルートとした木構造によりデータを管理しており、ユーザごとに異なった視点に基づいた情報提供を行うことが不可能である。

このような特徴を踏まえた上で、管理すべき情報の選択と、その管理形態についての注意深い設計が必要になる。

4.4.2 WIDE/PCS と情報管理

これまでの議論をまとめると WIDE/PCS が必要とする情報には以下のものがある。

(1) メッセージの配送経路に関する情報

(2) メッセージを受信するメディアそのものに関する情報

(3) メッセージの受信者そのものの状態に関する情報

さらに具体的に述べると、BIND により管理することが有効であると予想される情報は、以下の 3 点である。

(1) WIDE/PCS の所在地: 電子メールの形式でメッセージを受け取る WIDE/PCS が存在するマシンの情報

(2) ページャの情報: 受信者が所有するページャの形式と電話番号

(3) 受信者の状態: ページャを所有している受信者の状態

である。

次に、こららの情報を BIND を用いてどのように実現するのかについて述べる。

4.4.3 BIND による WIDE/PCS 情報の管理

(1) WIDE/PCS の所在地

純粋なユーザの視点に立つと、WIDE/PCS の所在地はなるべく隠蔽されていることが望ましい。すなわち、WIDE/PCS が稼働しているのがどのドメインにあるどのマシンかということを、潜在的なユーザには知らせず、あるエントリを公開する。これには BIND の中のある名前空間の一つ (たとえば wide.ad.jp) の中に仮想的なドメインに相当する名前 (たとえば pager) を確保し、そのドメイン内の情報として BIND のレコードにより WIDE/PCS に関する情報を登録する。ここで障害に対する信頼性を高めるためには、ドメインに対するセカンダリを複数の位置に設置する。

仮想的な名前空間の中で、実際の WIDE/PCS の所在地に対応するためには、いくつかの方法が考えられるが、ここでは以下の二種類の方法とする。

(a) 電子メールの転送機構を利用して、ページャの所有者ごとに適切な転送先を指定する。たとえば東京周辺の所有者に対しては東京近郊の WIDE/PCS によりメッセージを送出するように転送先を指定する。

具体的には、従来からの慣用に従い user-pager@pager.wide.ad.jp の表記法を利用し、各 user-pager ごとに適切な転送先を指定する。東京大学に所属する yoshiki が所有するページャ yoshiki-pager に対し、東京大学の starbow.nc.u-tokyo.ac.jp で稼働している WIDE/PCS を利用する場合には、以下のような alias のエントリを pager.wide.ad.jp を展開するマシン上に用意する。

```
yoshiki-pager:          yoshiki-pager@starbow.nc.u-tokyo.ac.jp
```

この方法は従来の方法の延長上にあり従来の慣用的な利用が可能であるが、転送先が固定的に指定されるために、障害に対する頑健性や冗長性に劣っている。

(b) BIND の MX レコードによる電子メールの配送を利用して、IP の経路制御および MX の優先度に基づく適切な WIDE/PCS 稼働マシンの選択を行う。

実際には、pager.wide.ad.jp の下に各ユーザごとのエントリを用意する。各ユーザは、ページャを呼び出すための負荷を分散し、あるいはネットワークの障害発生時に IP の経路あるいは uucp などの代替経路を利用して何らかの方法により WIDE/PCS に到達することができるように、複数の MX レコードを用意する。先に述べたのと同じく yoshiki-pager を起動するための yoshiki ドメインに対しての MX レコードは、次のように指定する。

```
; primary WIDE/PCS server for yoshiki
yoshiki      IN MX       10 starbow.nc.u-tokyo.ac.jp.
; secondary server aroud WNOC-TYO
             IN MX       20 aner.dit.co.jp.
             IN MX       20 nsh.is.titech.ac.jp.
; ternary server around WNDC-SFC
             IN MX       30 fxwide.fujixerox.co.jp.
```

MX レコードではプリファレンスが異なると最も小さな値を持つ MX レコードが選択され、プリファレンスが等しいと MX レコードの選択はは乱数により決定される。これはネットワークが正常な状態ではある定まったマシンを利用し、ネットワークに障害が発生すると、MX のエントリを順に試すことで WIDE/PCS までの到達性を確保することができる。

実際には、この二種類の方法をあわせて利用する。

(2) ページャの情報

ページャの情報はページャの形式と電話番号である。現在の WIDE/PCS では wide-pcs.pl が参照するファイル pagerdb により、それぞれのマシンで静的に管理している。この方法の問題点は、全ての WIDE/PCS が稼働しているマシン上で同一の情報を個々に管理することである。今後予想される WIDE/PCS 稼働マシンの増加およびページャの所有者の増加に対して、常時すべてを整合をたもちつつ管理することは難しくなってくる。

ページャの形式と番号に関する情報を BIND により管理するためには、HINFO レコードを利用することが最も容易であろう。たとえばページャの所有者を “yoshiki” とすると、BIND に登録するデータ形式は次のようにすればよい。

```
yoshiki      IN HINFO    ntti 03-3141-5926
```

この方法の問題点は、BIND ではアクセス制御の機能がないため、ページャの番号が公開情報となり、不正利用されるおそれがあることである。このため、何らかの手法により暗号化して登録しなければならない。この暗号の復号化の方法は、HINFO

レコードを利用する pcsd のみが知り得ればよい。具体的にどのように暗号化するかは、当面の実験のためには比較的単純な方法で十分である。しかし、WIDE/PCS を広く公開する際には、セキュリティを含め十分な考慮が必要になるであろう。

(3) 受信者の状態

受信者の状態とは、ページャの着信を拒否するであるとかメッセージの受信が不可能な場所にページャがあるといった情報である。このような状態に関する情報を管理するためには、登録されているデータを頻繁に更新する必要があるが、これは BIND に想定されている利用形態からは外れる。

そのため、受信者の状態情報を BIND で動的に管理するのではなく、BIND の情報により配送されたメッセージが WIDE/PCS を起動する部分に手を加えて、受信者の状態に応じてページャの呼出しのための振舞を変更する。

この方法を円滑に運用するために、慣用的に用いられている電子メールからページャへの転送サービスの宛先となっている user-pager を用いずに、メッセージの優先度が一見して分かるようなキーワードに置き換えることを考える。

(1)-(b) のユーザごとのドメイン形式を利用すると、pager.wide.ad.jp のサブドメインとしてページャの所有者は一意に決定され、@の左辺のユーザ名の部分の利用は冗長になる。転送されるメッセージそのものの優先度のキーワードを定め、ユーザ名の部分にキーワードそのものを利用する。たとえばユーザ名の代わりに emergent、urgent、normal をそれぞれ非常事態、緊急、および通常の優先度を持つメッセージのキーワードとして、非常事態のメッセージを、emergent@yoshiki.pager.wide.ad.jp と表記することにする。

WIDE/PCS の起動部分に、このキーワードに従ってメッセージの転送の方法を変更できるような機構を用意し、各ユーザがそれらの起動機構が参照するファイルを各自の状態にあわせて書き換える。たとえば、あるファイルに緊急のメッセージ以外は転送を禁止する、あるいはページャが利用できない場所に出張中の時には非常事態メッセージに対してはその旨を送信者に返送する、といったことをを記述する。これにより、受信者の状態に応じてメッセージ転送の挙動を変更するという当初の目的を実現できる。

4.4.4　BIND の設定

これまでに述べてきた BIND による情報の管理をまとめると、実際の *pager.wide.ad.jp* のドメインに関する内容はたとえば以下のようになる。

```
; Nameserver Data File for 'pager.wide.ad.jp' domain
;
; pager.wide.ad.jp の SOA の記述
;
```

```
@           IN SOA      pager.wide.ad.jp. root.pager.wide.ad.jp.
            ( 100010 3600 300 3600000 86400 )
;
; pager.wide.ad.jp の NS の設定
; 障害を回避するためには複数用意すべきである
; 最終的に全ての NOC ごとにセカンダリを設置することが望ましい
;
            IN NS       pager.wide.ad.jp.
            IN NS       nsh.is.titech.ac.jp.
            IN NS       starbow.nc.u-tokyo.ac.jp.
;
; pager.wide.ad.jp そのものの MX の宛先
; これについても全ての NOC に WIDE/PCS を設置することが望ましい
;
@           IN MX       10 pager.wide.ad.jp.
            IN MX       20 nsh.is.titech.ac.jp.
            IN MX       20 starbow.nc.u-tokyo.ac.jp.
;
; pager.wide.ad.jp の下位のドメインの一般的な MX の宛先
;
*           IN MX       10 pager.wide.ad.jp.
            IN MX       20 nsh.is.titech.ac.jp.
            IN MX       20 starbow.nc.u-tokyo.ac.jp.
;
; 先に述べた yoshiki.pager.wide.ad.jp に関する記述
;
yoshiki     IN MX       10 starbow.nc.u-tokyo.ac.jp.
            IN MX       20 aner.dit.co.jp.
            IN MX       20 nsh.is.titech.ac.jp.
            IN MX       30 fxwide.fujixerox.co.jp.
            IN HINFO    ntti 03-3141-5926
```

次に、pager.wide.ad.jp の電子メールのエイリアス・ファイル (aliases) には以下のようなエントリを追加する。

```
#
# yoshiki-pager@pager.wide.ad.jp というこれまでと同じ形式
#
yoshiki-pager:  yoshiki-pager@yoshiki.pager.wide.ad.jp
#
```

```
# yoshiki@pager.wide.ad.jp という形式でも正しく転送される
#
yoshiki:        yoshiki-pager@yoshiki.pager.wide.ad.jp
```

さらに yoshiki.pager.wide.ad.jp のエイリアス・ファイルには次のようなエントリを用意する。

```
#
# wide-pcs を呼出しメッセージを転送する
#
yoshiki-pager:  "|/usr/lib/wide-pcs -user=yoshiki@nc.u-tokyo.ac.jp"
#
# メッセージの緊急度にあわせて対応する処理を変更する
#
emergent:       "|/usr/lib/emergent -user=yoshiki@nc.u-tokyo.ac.jp"
urgent:         "|/usr/lib/urgent -user=yoshiki@nc.u-tokyo.ac.jp"
normal:         "|/usr/lib/wide-pcs -user=yoshiki@nc.u-tokyo.ac.jp"
#
# 他のユーザが WIDE/PCS を利用するためのエイリアスも設定する
#
user-pager:     "|/usr/lib/wide-pcs -user=user@nc.u-tokyo.ac.jp"
```

このような設定を実際に用意して、複数の WIDE/PCS のサーバと有機的に連携することにより、有効なネットワーク管理のツールとして利用することが可能になる。

4.5 X.500 Directory Service による WIDE/PCS 運用情報の管理

WIDE/PCS はサーバ上にあるファイルよりユーザ名からページャの電話番号などのデータを取り出し、モデムや NCU を用いてページャにメッセージを転送する。

WIDE/PCS サーバにファイルを置くという形式は簡便ではあるが、以下のような不都合がある。

1. 変更が生じた場合、すべてのサーバのデータを変更しなければならない
2. 利用者が知らないページャに関する情報がとれない
3. セキュリテイに関する保護が Unix のファイルシステムのパーミッションのみである

そこで、何らかの分散型のデータベースが要求されている。現在候補としては以下のようなものがあげられている。

1. DNS のフィールドを拡張してデータを入れる
2. X.500[183][184] を利用する
3. 独自のデータベースを作成する

本節ではこのうち 2 番めのアプローチである X.500 Directory を用いた場合の有用性などを考察する。

4.5.1 X.500 の優位性

X.500 Directory は、強力な検索機構と分散型データベースを持つ。検索に関しては、検索範囲を以下のように指定できる。

1. 特定の部分木の下全て
2. 特定の部分木の下一階層のみ
3. 特定の部分木の下のリーフのみ

また、検索探索条件として以下のように指定でき、また複数の属性について検索を組み合わせて指定できる。

1. 正確な一致 (Exact matching)
2. 大体の一致 (Approximate macching)

分散型データベースとしては、DNS 同様に Master/Slave モデルを採用しデータのレプリケーション (Replication) を行なっている。そのため、図 4.5のようにネットワークが分断されているような場合でも、レプリカが分断されたネットワーク内にあれば問題なくデータのアクセスが可能であり、WIDE/PCS が効力を示す緊急時に際してもレプリカの配置をうまく行なえば[4]データの入手が可能である

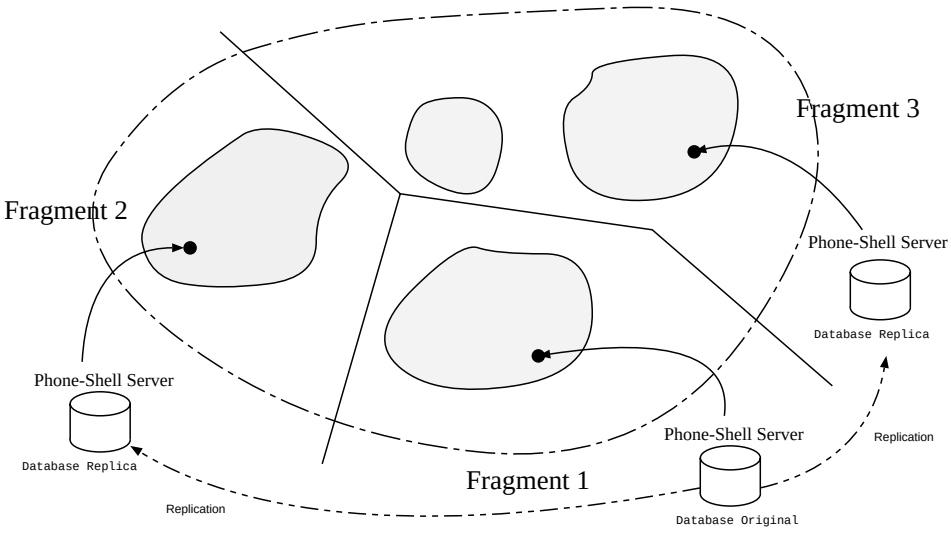


図 4.5: 分断されたネットワーク

セキュリテイに関しては X.500 は認証 (Authentication) と承認 (Authorization) を用意している。

認証 (Authentication)[5]においてはユーザは DAP(Directory Access Protocol) を用い DSA(Directory System Agent)[6]にアクセスする際に認証され、データの入手を行なうユーザの特定ができる。認証には「なし (None)」/「簡易 (Simple)」/「厳密 (Strong)」が用意されており、場合により使い分けができる。通常は簡易もしくは「なし」が使われる。

1. 「なし」は、認証を全く行なわないモードである。
2. 「簡易」は、簡易なユーザ認証を行なうもので通常は一方向性ハッシュ関数を用いたパスワードのによる認証が行なわれる。
3. 「厳密」は、公開鍵暗号系を用いたデジタル署名による認証が行なわれる。

承認 (Authorization) に関しては、ACL(Access Control List) により、任意のユーザの特定のエントリィ(Entry)/エントリィ内の属性 (Attribute) に対して以下の権限の指定ができる。

[4]例えば必要なデータのレプリカをすべての WIDE/PCS サーバ上におく等
[5]認証に関しての詳細は X.509[185] を参照のこと。
[6]DAP/DSA に関しては [186] を参照のこと

1. READ – 読み出し許可
2. WRITE – 書き込み許可
3. COMPARE – 比較許可
4. DETECT – 検出許可

DETECT が指定できることにより、ユーザによってはエントリィ(もしくはエントリィ内の属性) の存在すらも検知できないような設定ができる。また属性の継承 (Inheritant Attribute) を使うことにより、下位のエントリィに ACL の内容を継承することができる。

```
acl= group # c=JP@o=Fuji Xerox Co., Ltd.@cn=Manager # write # entry &\
        group # c=JP@o=Fuji Xerox Co., Ltd.@cn=Manager # write # default &\
        self # write # attributes # userPassword &\
        others # compare # attributes # userPassword &\
        self # write # attributes # userPassword &\
        others # compare # attributes # userPassword &\
        self # write # attributes # pagerTelephoneNumber &\
        others # detect # attributes # pagerTelephoneNumber
```

図 4.6: ACL(Access Control List) の例

図 4.6の例では以下のような指定を行なったことになる。

1. c=JP@o=Fuji Xerox Co., Ltd.@cn=Manager はエントリィの追加とデフォルト値の設定
2. 自分自身が userPassword と pagerTelephoneNumber の設定
3. それ以外のものは userPassword の比較と pagerTelephoneNumber が設定されているかどうかを検出

4.5.2 WIDE/PCS への適用

WIDE/PCS では、ページャのユーザ/ページャの電話番号/ポケットベルのタイプ/パスワードを図 4.7の形式でローカルのファイルとして保持している。また、WIDE/PCS 自体の動作を決める情報を図 4.8の形式で保持している。

pagerdb は各行のデータはユーザ名/ページャの電話番号/ページャのタイプ/パスワード (オプション) の形式になっている。

wncud.conf は、使っているダイヤルタイプ、エリアコード、コネクションを許す/許さないホストのリストなどが記入されている。

これらのデータを X.500 のデータとして保持することを考える。

```
#
# Pager # database for Phone-Shell
#
# User name     Pager TEL. #    Pager Type      Passwod(Optional)
shigeya@*.foretune.co.jp        03-5100-0000    ntti    0123
nojiri@*.hitachi.co.jp          03-5018-1111    ntti    4567
shin@*.iij.ad.jp                03-5018-2222    ntti    8901
hohno@*.titech.ac.jp            03-5018-3333    ntti    2345
toku@*.dit.co.jp                03-5018-4444    ntti
ryu@*.fujixerox.co.jp           03-5018-5555    ntti
bignum@*.kanagawa-u.ac.jp       03-5018-6666    ntti    6789
yabuki@*.sophia.ac.jp           03-5018-7777    ntti
yoshiki@*.u-tokyo.ac.jp         03-5018-8888    ntti
```

図 4.7: pagerdb の例

```
% wncud.conf
%
dialer = tone
areacode = 044
outsidelineseq = "0,"
countrycode = 81
accept = 129.249.88.*
accept = 131.112.40.3
refuse = 131.221.*.*
```

図 4.8: wncud.conf の例

X.500 には人を表す Object として Person という ObjectClass が存在する。この ObjectClass は X.521[187] に定義されている。広く Inetrnet で使われている DSA である quipu[188] では、quipuPerson[148] という ObjectClass を追加しておりこの中で pagerTelephoneNumber という属性を追加している。同様に WIDE/PCS に必要なデータを属性として登録し新たな ObjectClass として phoneShellPerson を定義する。

図 4.1に WIDE/PCS で必要な属性と思われる属性の性質をまとめた。これらの属性は今後の拡張も考え、現在の WIDE/PCS のスーパセットとなっている。

phoneShellPerson/phoneShellEntity は、図 4.9のような ObjectClass となる。phoneShellPerson/phoneShellEntity で使用する AttributeType は、図 4.10のようになる。追加される AttributeType は、表 4.2のようになる。

CaseIgnoreString は大文字小文字を区別しない文字列であり、NumricString は数字 (0-9) を表す AttributeType である。

必要な属性	データとして存在しうる値
ページャの電話番号	数字および ()-
ページャの型	任意の文字
ページャのパスワード	数字および*#
ページャの呼び出し可能時間	任意の文字
WIDE/PCS サーバの設置場所の市外局番	数字
WIDE/PCS サーバに接続できるホストのアドレス	英数字 [0-9A-Za-z]
WIDE/PCS サーバに接続できないホストのアドレス	英数字 [0-9A-Za-z]

表 4.1: WIDE/PCS で必要な属性

```
# WIDE Phone-Shell defined object class
phoneShellPerson:       widePhoneShellObjectClass.1 : \
        newPilotPerson : : \
        pagerType, pagerPassword, pagerActivity
phoneShellEntity:       widePhoneShellObjectClass.2 : TOP : \
        phoneShellAreaCode : \
        phoneShellAcceptHosts, phoneShellRefuseHosts
```

図 4.9: phoneShellPerson/phoneShellEntity の ObjectClass

このような ObjectClass/AttributeType を定義し、DIB にデータを入力し WIDE/PCS が必要なデータを検索できる環境を X.500 上で構築すればよいことになる。

プロトタイプとして、富士ゼロックス内の DIB に pagerTelephoneNumber を入れたユーザを作成し、検索できる環境を整えた。富士ゼロックスないのディレクトリィの環境は ISODE 8.0 の Quipu DSA を持ちいて構築されており、DUA(Directory User Agent)[186] は同じく ISODE 8.0 の DUA のう dish のコマンドラインを用いた。

富士ゼロックスの DIT のおよその構造を、図 4.11に示す。

“c=JP@o=Fuji Xerox Co., Ltd.” の直下に organizationalUnit である “ou=Distributed Services” があり、その中の何人かのユーザのうち”cn =Ryu Inada” のみが pagerTelepho-

AttributeType 名	属性	Description
pagerType	CaseIgnoreString	ページャの形式 (NTTI など)
pagerPassword	NumericString	ページャのパスワード
pagerActivity	CaseIgnoreString	ページャを呼び出して良い時間帯を記入
phoneShellAreaCode	NumericString	WIDE/PCS サーバの設置場所の市外局番
phoneShellAccept	CaseIgnoreString	WIDE/PCS サーバが接続を許すホスト
phoneShellRefuse	CaseIgnoreString	WIDE/PCS サーバが接続を拒否するホスト

表 4.2: 拡張する AttributeType

```
# WIDE Phone-Shell defined object attribute
pagerType:      widePhoneShellAttributeType.1 \
                        :CaseIgnoreString
pagerPassword:  widePhoneShellAttributeType.2 \
                        :NumericString
pagerActivity:  widePhoneShellAttributeType.3 \
                        :CaseIgnoreString
phoneShellAreaCode:     widePhoneShellAttributeType.4 \
                        :NumericString
phoneShellAcceptHosts:  widePhoneShellAttributeType.5 \
                        :CaseIgnoreString
phoneShellRefuseHosts:  widePhoneShellAttributeType.6 \
                        :CaseIgnoreString
```

図 4.10: phoneShellPerson で追加された AttributeType

neNumber 属性を持っている。

図 4.12に csh のスクリプトで記述した pagerTelehoneNumber を持つユーザを探すスクリプト pagerSearch をあげる。

このスクリプトは、パラメータとして「Directory 上でのユーザ名」と「検索を行なう部分木 (Subtree) の名前をとる。行なっている処理は、簡単で以下の通りである。

1. ディレクトリィサービスに接続
2. 検索条件を設定し検索
3. ディレクトリィサービスを切断

わずか数行の csh のスクリプトで WIDE/PCS にとって必要なデータを取り出す枠組が完成した。このスクリプトの実行例を図 4.13にあげる。

この例では “@c=JP@o=Fuji Xerox Co., Ltd.@ou=Distributed Services@cn=Ryu Inada” が検索条件にマッチし、ページャの電話番号 (+81 3-5018-5555) が出力されている。

4.5.3　今後の展開

今回は、すでに属性として pilotPerson に登録済みの pagerTelephoneNumber を使ってデータの検索ができることを実証した。

今後はその他の属性値を設定し、phoneShellPerson/phoneShellEntity を用いた DIT における WIDE ／ PCS が検索しやすい形での DIT の構成などを実験する予定である。

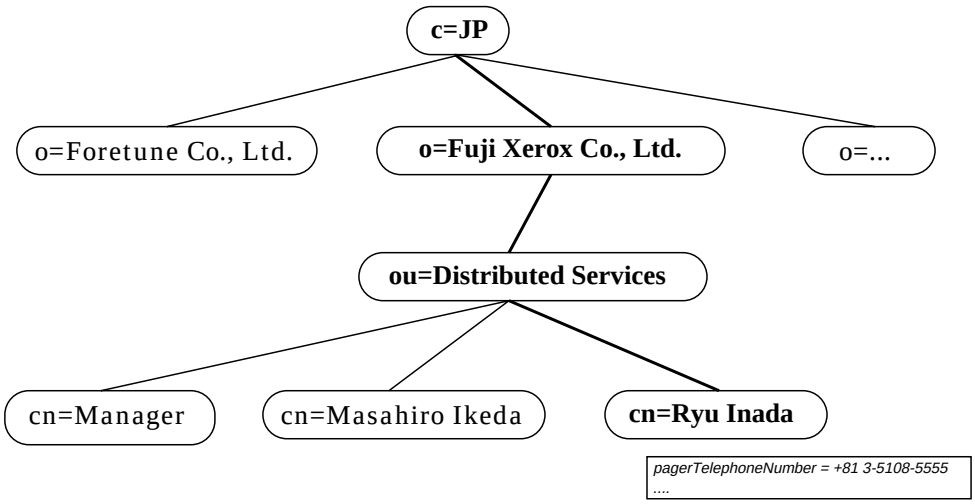


図 4.11: c=JP@o=Fuji Xerox Co., Ltd. の DIT 概観

```
#! /bin/csh -f
#       Search pagerTelephoneNumber attribute & show its value.
# Usage:
#       pagerTelSearch ‘‘username’’ ‘‘serach subtree’’
#
setenv  DISHPROC        "127.0.0.1 ‘expr $$ + 32768‘"   # Define Port No.
# Connect to DSA by using DAP.
bind -user $1
if ($status != 0) then
        exit 1
endif
# Search operation.
search -object $2 -filter "pagerTelephoneNumber=*" -subtree -show \
    -type pagerTelephoneNumber -norelative
if ($status != 0) then
        unbind
        exit 1
endif
# Disconnect.
unbind
exit 0
```

図 4.12: ISODE の環境下での pagerTelephoneNumber を Search するスクリプト

```
ryu@valkyrie[12:13pm][Phone-Shell:199] pagerTelSearch '"@c=JP@o=Fuji
Xerox Co.,Ltd.@ou=Distributed Services@cn=Ryu Inada"' '"@c=JP@o=Fuji
Xerox Co., Ltd."'
Enter password for "@c=JP@o=Fuji Xerox Co., Ltd.@ou=Distributed
Services@cn=Ryu Inada":
1   c=JP@o=Fuji Xerox Co., Ltd.@ou=Distributed Services@cn=Ryu Inada
pagerTelephoneNumber  - +81 3-5018-5555
```

図 4.13: Search 実行例

第 5 章

おわりに

WIDE プロジェクトの正式なワーキンググループとなった PhoneShell ワーキンググループは、今年度後半から新たな参加者得て WIDE/PhoneShell に関連する諸問題をとりあげ検討した。その成果は以下の 3 つ集約される。

1. WIDE/PhoneShell の基礎技術が強化された (第２章)。
2. WIDE/PCS を利用した応用技術が開発されネットワーク管理に寄与した (第３章)。
3. 今後の発展に向けて、あらたな試みが始まった (第４章)。

今後は第４章で述べた (上記 3 に属する) 各項目の発展にまず力を注ぎ、来年度の早い時期に WIDE インターネットの管理の現場にその成果を投入したい。第４章で述べた項目が実装されれば、現在の WIDE/PCS はいっそう利用しやすく障害に強いシステムに発展する。

また、今年度はあまり活発でなかった音声合成装置、音声認識装置やファクシミリなどと連携する研究を発展させ、さらに今後普及が予想される無線を使った小型携帯端末装置の利用も積極的に検討する予定である。

謝辞

Sophia/PhoneShell の開発に際して、ご協力を頂いた株式会社明電舎に感謝する。