JAIST Repository
https://dspace.jaist.ac.jp/

Title	深い知識に基づくネットワーク故障診断ESのモデル化と実装 : プロトタイプ第1版
Author(s)	長田, ともみ; 篠田, 陽一; 山口, 高平; 落水, 浩一郎
Citation	Research report (School of Information Science, Japan Advanced Institute of Science and Technology), IS-RR-93-0016I: 1-37
Issue Date	1993-11-27
Type	Technical Report
Text version	publisher
URL	http://hdl.handle.net/10119/8420
Rights	
Description	リサーチレポート（北陸先端科学技術大学院大学情報科学研究科）

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

深い知識に基づくネットワーク故障診断ＥＳの モデル化と実装（プロトタイプ第１版）

長田ともみ　篠田陽一　山口高平　落水浩一郎

1993 年 11 月 27 日
IS-RR-93-0016I

北陸先端科学技術大学院大学
情報科学研究科
〒 923-12 石川県能美郡辰口町旭台 15
tomomi@jaist.ac.jp,shinoda@jaist.ac.jp
yamaguti@cs.shizuoka.ac.jp,ochimizu@jaist.ac.jp

©Tomomi Nagata, 1993

ISSN 0918-7553

概要

本論文では、ヘテロなネットワークシステムを対象とした故障診断エキスパートシステムの開発に関する中間成果を報告する。

われわれは、ネットワーク故障診断エキスパートの診断事例の分析結果に基づき、エキスパートシステムの開発が困難であるソフトウェア分野における、エキスパートの有する推論方式、利用する知識の構造を明らかにしつつある。

本論文は、この結果に基づき、プロトコルスタックに関するドメインモデルを入力して、故障診断木を出力するような、知識コンパイルの手法と中間成果をまとめたものである。

本論文では、以上の方式を詳細に説明しつつ、故障診断木を評価することにより中間成果をまとめるとともに、今後の研究方向を明らかにする。

もくじ

1　はじめに

ヘテロなネットワークとは、異なったベンダーから供給されるネットワーク機器上において、さまざまなプロトコルモジュール群の組合せによって実現されているシステムのことをいう。ネットワークの必要性が増大する現代、特に、このヘテロなネットワークの構築が進められている。

ヘテロなネットワークは、そのシステムの構造上の特質から、故障が発生した場合に故障箇所や原因を特定することが非常に難しい。そのため、実際にネットワークの構築に携わったごく少数の専門家でなければ、ネットワークの維持・管理を行ない、故障に対応することができないという問題が生じている。

そこで、適当なエキスパートシステム（以降“ＥＳ”とする）を用いて故障診断を自動化し、ネットワークシステムの円滑な運用を行なうために、専門家の知識を他者が利用可能な形に形式化することが必要である。

本研究では、まず、研究対象を故障診断に限定し、以下の２点について具体的な事例の分析を行なった。

1. 故障診断を行なう過程で使用している推論方式

2. 推論過程の中で利用されている様々な種類の知識とその相互関係

次に、この事例分析の結果をもとに、深い知識に基づく故障診断としてのネットワーク故障診断ＥＳ（プロトタイプ第１版）の実装を行なった。本論文では、事例分析と実装を通して得られたネットワーク故障診断ＥＳに関する考察と課題について述べる。

深い知識に基づく故障診断は、ＲＢＤ（ルールベース）やＣＢＤ（事例ベース）と比較して、特にこれまでＥＳのボトルネックといわれた知識獲得の困難さを軽減するという点で、ネットワークの故障診断に適している。また、対象であるネットワークは論理的世界であり、かつその構造が複雑であるために、同じくモデルを利用したＭＢＤ（知識のコンパイルプロセスを重視）よりも深い知識に基づく故障診断（深い知識の記述を重視）の方が適していると考える。

プロトタイプの実装に際しては、既存の、深い知識に基づく故障診断システム“ＫＣ２”を推論エンジンとして利用し、そのシステムに与えるネットワークモデル、知識の記述方法に関して検討し、システムの有効性を評価した。

以降、第２節では、２つの代表的な故障診断事例を取り上げて、専門家が使用している推論方式と推論に利用している知識構造に関する特徴を洗い出し、ネットワーク診断の基本的考察と概念を示す。第３節では、診断事例の分析結果から導かれる特質を持った（あるいは、そのような要件を満たすべき）ネットワーク故障診断という対象領域には、深い知識に基づく故障診断が有効であることを示す。そして、プロトタイプの作成として、実装する領域（問題・障害）を限定し、ＫＣ２に与える深い知識の定義と実際の記述例を示す。第４節では、実際に出力された故障木について考察を行ない、そこからネットワークモデルの概観を示す。最後に、事例分析とプロトタイプの実装を通して明確になってきた、ネットワークＥＳに関する今後の課題について述べる。

2　ネットワーク故障診断事例の分析とＥＳの概念

本節では、実際の故障診断事例を分析し、その結果から導かれるネットワーク故障診断ＥＳの概念を示す。

今回分析の対象とした２つの事例は、障害状況等を見ると、全く性質の異なる障害であるといえる。しかし、その２つの障害対応のために専門家がたどった過程は、非常に良く似たパターンを示していることがわかる。

2.1　具体的な故障診断事例の分析と、その解釈

ここでは、分析の対象とした２つの代表的な故障診断事例を紹介し、専門家が利用している知識構造モデルや推論方式を分析する。

2.1.1　事例１：ＮＦＳレスポンスが悪い

事例１として、新しくインストールされたネットワークシステムにおけるＮＦＳレスポンスの悪さの原因追求過程において、初期の徴候から原因の特定までの間に、エキスパートが行なった活動の経過を示す。さらに、事例１の活動の過程の各項目毎に、活動の目的を整理する。

1. 徴候 “ＮＦＳレスポンスが悪い” から原因を推論する過程

   1. まず、何がこの現象に関係しているのかを想定する。（故障対象の想起）

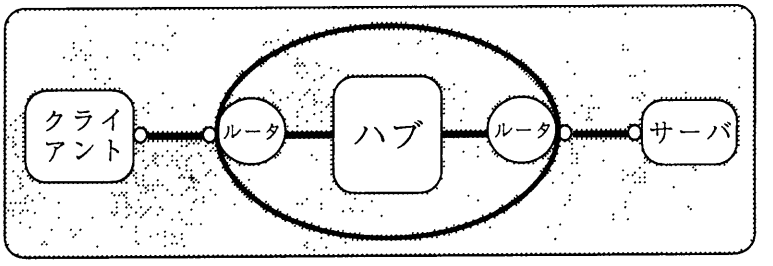


図１ 故障対象の想起

同時に、過去の経験事例の中に良く似たものが存在することを思い出す。
（過去の事例と運用上の実績データより仮説を生成）

故障対象	仮説	判断材料
クライアント	異常	新ＯＳ、Binary
サーバ	正常	安定稼働
ネットワーク	正常	工場テストＯＫ

表１ 故障仮説の生成

2. 両仮説に基づいて、有罪と思われるクライアントの異常を確認するテストを行なう。

3. クライアントもサーバも応答する、異常がない。(実は異常が存在したが気づかなかった)

4. クライアントのネットワークレスポンスをテストする。遅くない時もある。

5. ネットワークを通さずに、直接クライアントとサーバを繋いでレスポンスをテストする。するとレスポンスが良い。

行き詰まり

ここまでのテスト・観測を通して、通るファイルと通らないファイルが存在することがわかった。またファイルサイズが大きくても通ることもあり(Xsun:1M byte)、小さくても通らないことがある(.twmrc:1k byte)という事実も確認している。

6. サイズに無関係であることを確認するため、パケットを観察する。

7. **新徴候発見**:サーバからのフラグメント化された応答の最終パケットが欠落している。→ 仮説の翻意

故障対象	新仮説	判断材料
クライアント	正常	送信要求ＯＫ
サーバ	異常	パケット欠落
ネットワーク	異常	パケット欠落

表2 故障仮説の翻意(修正)

8. サーバに異常があるか否かを確認する。欠落パケットがサーバからは送信されている。

9. その結果、ネットワークに原因があるという仮説を生成する。同時に、ＳＵＮのパケットモニタを利用した観察の結果、欠落はＵＤＰプロトコルで発生していることが確認されている。そこで、異常はＮＦＳではなく、その基盤であるＵＤＰ全般に発生するのではないかという仮説を生成した。

10. ＵＤＰのサービスを使い、データのサイズとパケット欠落の関係を調査した。

11. **障害原因の特定**　ＵＤＰのフラグメンテーションが起きた場合に、最終パケットのサイズが、60〜182 bite の範囲にある時に、その最終パケットのみが欠落する。

2. 事例１における、各故障診断過程の活動の目的

1. まず、障害に関する何らかの異常兆候が得られ、故障対象を想起し、さらに、故障対象と障害の関係について仮説をたてる。
2. 仮説を裏付けるためのテストを行なう。
3. 観察を行なった結果、新しい兆候群を得る。
4.5.6. 新しい兆候群を元に、仮説の検証を進める。
7. 仮説の翻意が起こる。
8. 新しい仮説を裏付けるためにテストを行なう。
9. 仮説が裏付けられる。
10. 犯人候補の内部構造を分解し、どの部分に原因があるのかを調査しつつ、原因を特定する。

上記過程において利用されている推論と知識の相互関係をまとめると、図２のようになる。

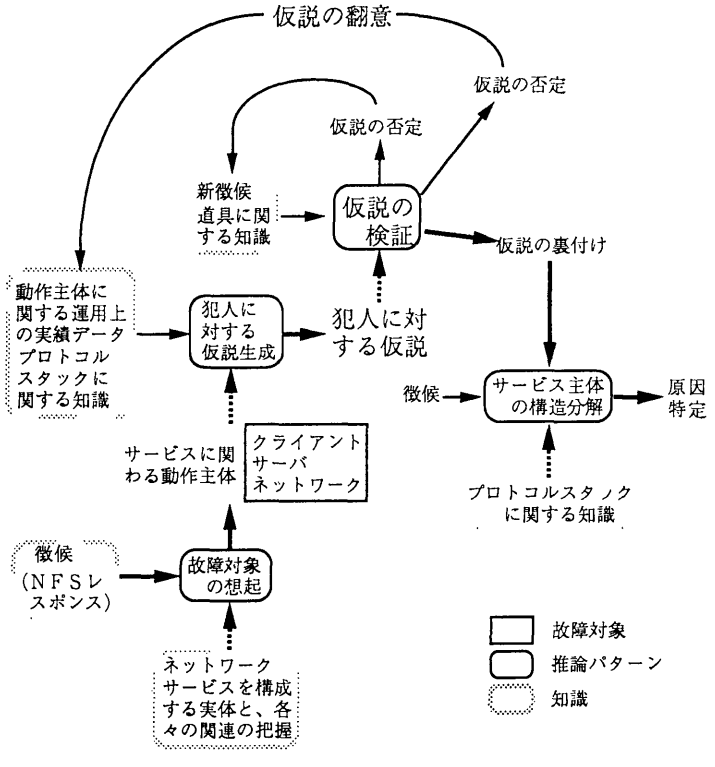


図2 ＮＦＳレスポンスが悪い

2.1.2 事例２：送信待ちメールがたまる

事例２（付録２）では、電子メールサーバにおいて送信待ちのメールが徐々に蓄積していくという現象の解消を取り挙げる。

1. 徴候 “送信待ちメールがたまる” から原因を推論する過程

1. まず、何がこの現象に関係しているのかを想定する。（故障対象の想起）

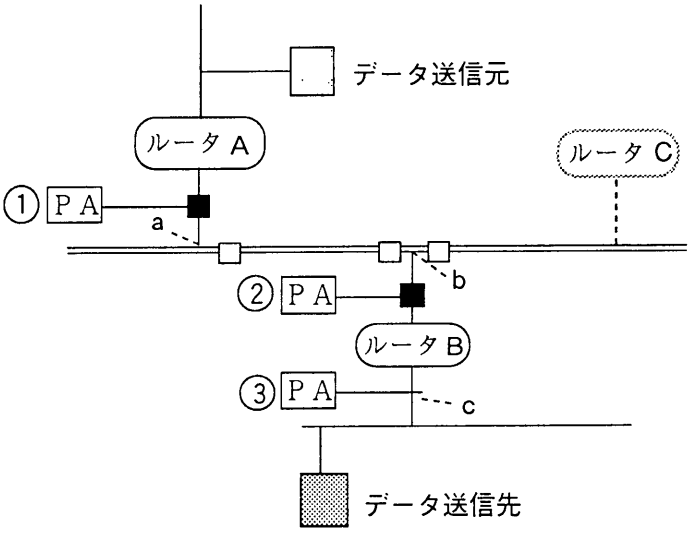


図３ 故障対象の想起

{送信元ホスト、ルータＡ、ネットワーク、ルータＢ、送信先ホスト}
送信先ホストがダウンしているにしてはメールのたまり方が少ないと判断する。

2. 送信先ホストがダウンしていないことを確認。

3. telnet が可能な時もある。ネットワークが不安定。→ＴＣＰ層以下の層で異常が発生していることが多い。

4. 経路制御がおかしくないか確認。ルータＡにルータＢへの経路が存在する（正常であると判断する）。

5. 経路制御より下がおかしい？。ルータＡからのＩＰレベルでの出力は正しい（a 点での観測による）。

6. ルータＢへもＩＰパケットは流れている（実は常に届いているわけではない、b 点での観測による）。

7. ルータＢから送信先へのパケットがでるか?→出ていない（c 点での観測による）。

8. ＡＲＰに異常があるのではないか。

9. ルータＡから送信されるパケットの ethernet ヘッダを見直す。全てのＩＰパケットが同じ宛先の ethernet アドレスを持っている（そのような機器は存在していないはず）。

10. つまり、ルータＢ以外のところにパケットが吸い込まれている。

11. ルータＡのＡＲＰエントリを調べると全てのＩＰアドレスが同じ ethernet アドレスにバインドされている。

12. **障害原因の特定**　障害発生当日に新設されたルータＣが、バックボーン上の全てのＡＲＰに答えている。

2. 事例２における、各故障診断過程の活動の目的

1. まず、障害に関する何らかの異常兆候が得られ、故障対象を想起し、さらに、故障対象と障害の関係について仮説をたてる。

2. 仮説を裏付けるためのテストを行なう。

3. テストを通して得られた新兆候から、仮説を修正する。

4.5.6.7. プロトコルスタックを上位から下へ向かってたどりながら、仮説を裏付けるテストを行なう。そして、その結果得られた新兆候から仮説を詳細化していく。

8. 得られた新兆候から仮説を修正する。

9. 仮説を裏付けるために観測を行なう。

10. 仮説が裏付けられる。

11. この障害の直接的な原因が特定される。そこからさらに、その原因を生じた源を調査し、特定する。

上記過程において利用されている推論と知識の相互関係をまとめると、図４のようになる。

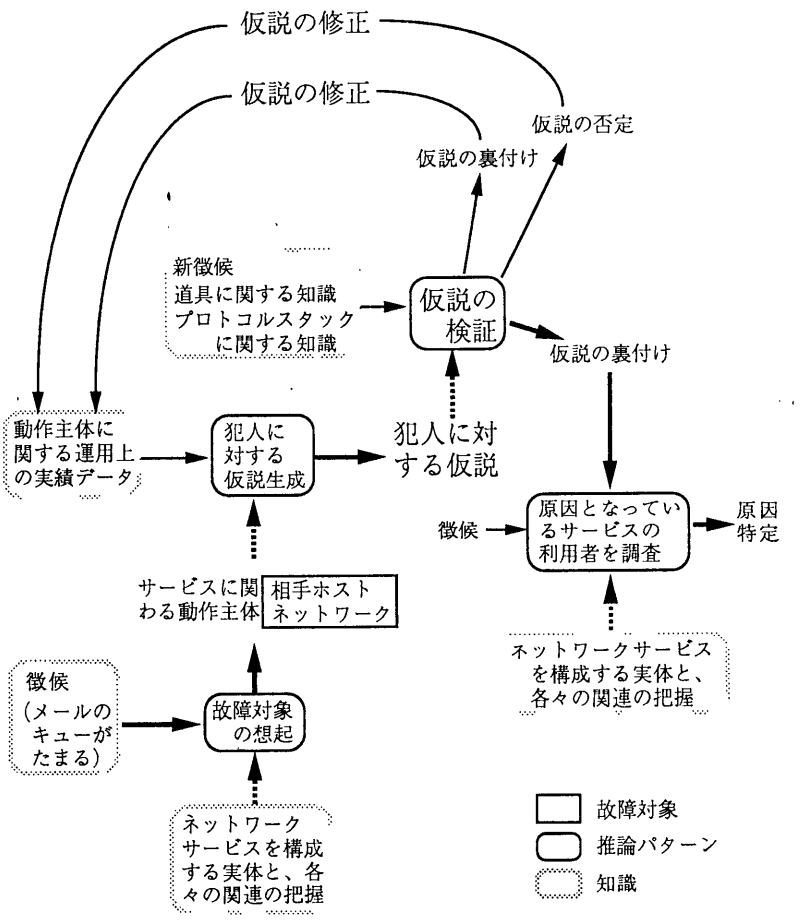


図4 送信待ちメールがたまる

2.1.3 専門家による故障診断の特徴（推論パターンと知識世界

以上みてきたように、今回収集した２つの事例は、故障状況としてはかなり性質の異なる事例である。しかし、診断過程を分析し、概念化した図（図２、及び図３）を見ると、徴候を得てからその故障原因を特定するまでの過程には、共通した手順（図の中の、実線の枠で囲われた推論パターンや検証）と、そのために利用される知識（図の中の破線で囲まれたネットワークに関する知識）が存在することが明らかになった。

以下ではその故障診断の手順を追いながら、同時にその際にどのような知識を利用しているかを示す。

1. 異常徴候が発生した場合、まず最初に「故障対象を想起」する。
    - ネットワークサービスを構成する実体と、各々の関連に関する知識
2. 想起した故障対象に関して、犯人に対する「仮説を生成」する。
    - サービスの動作主体に関する知識
    - 運用上の実績データ
3. その仮説を裏付けるために「仮説の検証」を行なう。
    - テスト手段に関する知識
4. 検証結果によって「仮説の修正」を行ない、再度仮説をたて直す。
    - プロトコルスタックに関する知識
    - ネットワークサービスを構成する実体と、各々の関連に関する知識
    - 運用上の実績データ
5. 直接故障箇所を示す仮説が裏付けられると、その障害が発生する根本的な原因を特定しようとする。
    - ネットワークサービスを構成する実体と、各々の関連に関する知識
    - プロトコルスタックに関する知識

このように、２つの事例に見られる推論パターンは５つに大別できる。さらに、その推論に利用される知識をまとめると、以下の３つの知識に整理される。

推論の際に利用している知識

1. 物理ネットワーク（ハードウェア）の構成についての知識：Structure の世界
2. 仮想ネットワーク（プロトコルスタック）の構成についての知識：Function の世界
3. 故障原因と観測される徴候の結び付けについての知識：Behavior の世界

2.2 ネットワーク診断ＥＳの基本的考察と概念

現在実用化されている故障診断ＥＳの大半はＲＢＤとして実装されている。このＲＢＤの推論方式は、異常徴候（条件部）と故障仮説（結論部）を表層的に関係づけたルールベースを用いて故障診断を試みる。そのために、ルールベース に 記述されていない（未知の）状況に対しては対処できないという「脆弱性」問題を持つ。

一方、前節の２つの事例分析からわかるように、専門家は、場当り的に収集された知識（ルール）を限定された観測事象に適用して、ネットワーク診断を試みているのではなく、すべての観測事象に対して対処できるように、ある原理原則（モデル／深い知識）に則って故障診断を試みている。事例分析から判断する限りにおいて、プロトコルスタックがそのモデルであると認識できる。

そこで、知識工学におけるモデルに基づく診断（Model Based Diagnosis、以下ＭＢＤ）の枠組みにより、前節の事例問題を捉えることができる。

ＭＢＤは、観察事象によりを仮説を生成した後に、仮説をテストする（その後、仮説を分類することもある）ための枠組みとして、従来 General Diagnostic Engine（ＧＤＥ）[1] や Reiter の診断理論 [2] など、形式的記述空間における探索問題の観点を重視して考察されてきた。その中で題材となってきたのは、論理回路診断等小規模なものが多く、診断システムが実用化時に遭遇する問題点等は、ほとんど議論されていないのが現状である。しかし、前節の事例分析は、その問題点を顕在化させている。

以下、その観点から、ネットワーク診断におけるＭＢＤシステムの構成法について考察する。

2.2.1 モデルの記述

従来のＭＢＤでは、論理回路等、物理世界（ハードウエア）がモデル化の対象となり、モデリング（モデル記述構成要素）問題については触れない。しかしながら、プロトコルスタックは、図５のように、最下位層でハードウエアとリンクした論理世界（ソフトウェア）であり、モデリングの良否が故障診断ＥＳの性能に大きく影響する。

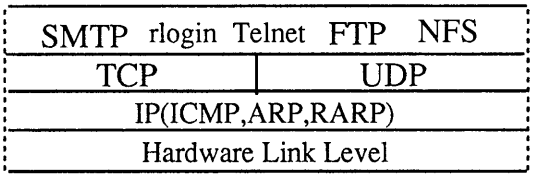


図５プロトコルスタックの概要

プロトコル名	プロトコル識別子
上位プロトコル名	プロトコルスタック探索時に使用
下位プロトコル名	プロトコルスタック探索時に使用
機能	プリミティブを定義する必要あり →故障シミュレーションに使用
信頼性	プロトコルスタック探索時に使用
振舞い	観測される異常兆候との照合
機能テスト手段	新兆候が獲得される
テスト容易性	プロトコルスタック探索時に使用
物理世界構成要素	故障箇所をプロトコルから物理世界に移行する時に使用

表３ プロトコルスタックの記述要素

現在、プロトコルスタックを表３のように考えており、例えば、Prolog風にSMTPを記述すると以下のようになる。

記述例

```
protocol(smtp1, AP, TCP,simple-mail-transfer, 2, [mail-queue,...],
        [ping, telnet,...], 2, host1).
```

物理世界構成要素間の接続関係の記述は、物理世界における故障箇所の探索に利用するためのものである。

2.2.2 モデル探索知識と汎用戦略

従来のＭＢＤは、診断プロセスを木構造におけるラベル計算に還元する[2]等、形式的に扱いすぎた感がある。前節の事例分析からわかるように、ネットワーク診断はプロトコルスタック探索問題に帰着できる。そして、その診断プロセスは、効率的探索計算手法の考察だけではなく、以下に示すモデル探索サブタスクに関わる知識の同定／整理を行なうことによって、さらに効率的なネットワーク診断を可能にするといえる。

Ａ．探索開始点の決定

事例２の「ＳＭＴＰが故障」という仮説について考えてみる。この仮説は、「送信待ちメールが溜る」という異常兆候とプロトコルスタック（振舞い）の記述と照合することにより容易に生成できる。

一方、事例１の場合には、「ＮＦＳのレスポンスが遅い」という異常兆候とプロトコルスタックを照合すれば、「ＮＦＳ（あるいはその下位プロトコル）の故障」という仮説が生成され、ネットワーク診断は非効率となる。従って、探索開始点の決定タスクにおいては、

経験則

```
if   ＯＳ ＝   新しい   and   バイナリ
          then   マシン故障確率が高い
```

というような経験則（事例）の整理が必要である.

Ｂ．探索移行点の決定

ネットワークの故障診断はプロトコルスタックの探索と考えることが可能であることを示した。そのプロトコルスタックの探索は、必ず上位から下位へ、一階層づつ移行しているということではない。

例えば、事例２においては、ＳＭＴＰ故障仮説から、ＴＣＰを飛ばして、もう一層下のＩＰ（ルーティング）の故障という仮説を立てている。

このプロトコルスタックの移行点の決定方法について、事例では、壊れる可能性の低いもの（信頼性の高いもの）は診断の対象から外したり、また、仮説検証のためのテストを実施しやすいものを優先し、診断を進めている。つまり、探索移行点を決定するひとつの目安は、主観的ではあるが、ある指標のようなものを利用しているといえる。

プロトコルスタックの探索下降点の決定タスクは、プロトコルスタックの信頼性値を利用すれば比較的容易に実現できるが、探索上昇点の決定タスクでは、複数候補が存在することが多いため、Ａと同様に経験則（事例）の整理が必要となる可能性が高い。

C．新規テスト手段の生成

このタスクを実現できる事が、専門家としての一つの特徴である。

具体的には、事例１でＵＤＰ故障という仮説を立てた後、次に、ＭＴＵ値の設定ミス（ＩＰレベルでの障害）という仮説を立てた。その仮説を検証するために、データサイズを順次小さくしながら ping するテストプログラムをＵＤＰ上位プロトコルとして位置づけて実行した。その結果、データサイズが 60-182 bite の時、最終パケットが落ちる事を発見し、ＭＴＵ値の設定ミスという故障原因を同定している。

ここで、専門家は、ＭＴＵ設定ミスの現象を経験していたために、ＩＰレベルの仮説を検証するためにＵＤＰレベルのテスト手段を利用するようなテストを実施することができた。もし、類似経験がなければ、通常のプロトコルテスト手段を使用して、上記のような発見はできなかったであろう。従って、やはり本タスクにおいても、関連する経験則（事例）の整理が必要となる。

D．プロトコルスタック探索戦略

Ｂ．でも述べたように、実際の故障診断はプロトコルスタックを単純に上位から下位に下降しているわけではない。事例２では、ＳＭＴＰからプロトコルスタックをハードウエアレベルまで下降し、その後、ＡＲＰまで上昇し、また、ハードウエアまで下降して、故障箇所を発見した。事例１では、ハードウエアレベルからプロトコルスタックを上昇し、故障箇所を発見した。

つまり、ある場所での仮説検証を終えた時、次にどこに移行するのがもっとも効果的な故障診断であるかの判断をとりいれようとするならば、それを行なえるのはモデルでも深い知識でもなく、人の経験則であるので、その整理が必要であるといえる。

以上のように、ネットワークにおける故障診断は、ハードウエアレベルでリンクされたプロトコルスタックの上昇下降による探索を基本戦略として、さらに、A～Dに関する経験則を付随実行させることにより、効率的な探索が可能になるといえる。以上の結果より、図6のようなネットワーク診断エキスパートシステムの概念図を得ることができる。

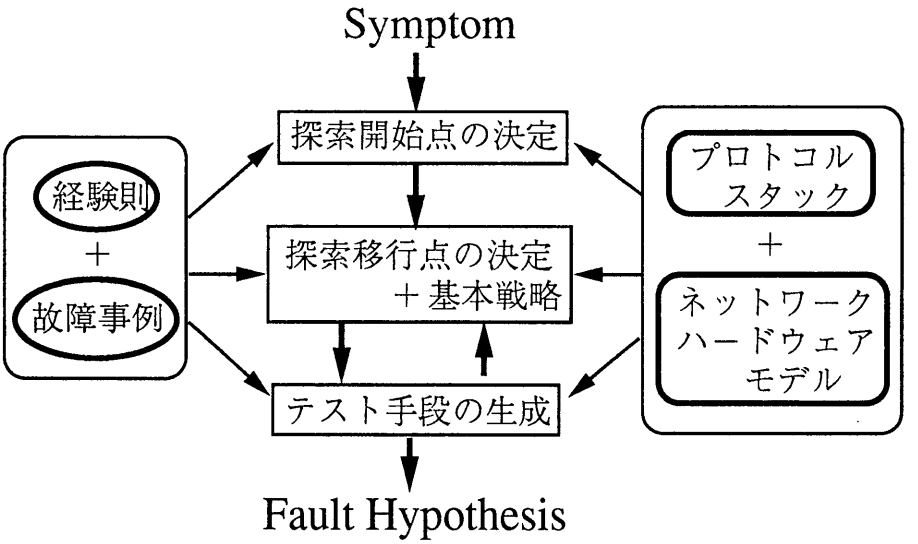


図6 ネットワーク診断エキスパートシステムの概念図

ネットワーク故障診断を行なうＥＳは、まず、ネットワークに関する基礎的な知識（プロトコルスタックを顧慮したソフトウェア、ハードウェアのモデル）を客観的に記述した知識ベースを用意する。そして、異常徴候が得られた時、知識ベースを基にして、どこから探索するか、次はどこを探索するかを決定しながら、故障仮説を導き出し、提示する。さらに、前項で示したように、整理された経験則や事例を利用することにより、効率的な故障仮説の提示が可能となると考える。

2.3 深い知識に基づく故障診断ＥＳの枠組

ここでは、事例分析の結果を踏まえ、これまでに示してきたネットワーク故障診断に有効なＥＳを実装するためには、どのような枠組で捉えることが必要であるかを検討し、実装についての方向を定める。

このＥＳの最も基本的な機能は、異常徴候を入力すると、知識ベース（モデル）を利用して探索を行ない、故障仮説（障害原因の候補）を出力することであるといえる。

そこで、まずこのような基本的な故障診断を行なうために、モデルをどのように記述するか、そして、どのように探索させるのかといった基本的な戦略に関する枠組について検討する必要がある。

2.3.1 モデルを利用した故障診断ＥＳ

従来のＥＳ研究を見た場合、ＥＳで必ず顧慮しなければならない問題として挙げられるのが知識獲得問題である。すなわちエキスパートの経験をそのままの形（ルール）で獲得することは、客観性、汎用性という点に欠け、また、知識の抜け落ちが発生する危険性をはらんでいるという問題である。

それに対してモデル・深い知識といった、実際に存在するものの接続関係や、その世界を拘束している原理原則（物理法則等）を記述する場合には、対象世界から比較的容易に構築することが可能であり、客観性、汎用性が高い。

しかし、この性質は必ずしもすべての対象領域にあてはまるとはいえない。深い知識やモデルが存在し、それが明確に理解できる領域もあれば、人の身体と病気の関係のように、原理原則といったものが解明されていない領域もある。前者の場合にはモデルや深い知識を用いるのが有効であるし、後者の場合には、経験則を整理し、ルールベースのシステムを構築するのが適している。

今回のネットワークの場合にはモデルが存在することから、ルールベースや事例ベースではなく、専門家が故障診断の際に利用する原理原則（モデルや深い知識）をうまく採り入れる必要があるといえる。

このような原理原則（モデルや深い知識）を利用した故障診断として研究されてきたシステムには、ＭＢＤ（Model Based Diagnosis：モデルに基づく故障診断）と深い知識に基づく故障診断の２つのシステムが存在する。

現在ではこの２つのシステムはほぼ同義として捉えることが可能であろう。しかし、それらの違いは、各々の研究の発展の始点（何に着目してきたか）を考える時に顕著に現れる。

以下にその両者の特徴と概要を示す。

1. ＭＢＤ

- 比較的小規模な物理世界（電器回路等）を研究対象としてきた。そのために、対象とする世界のモデルは比較的単純であり、記述が容易な世界である。
- そのような対象領域の性質から、対象世界のモデルや深い知識は既に与えられたものと位置付ける。
- ＭＢＤが重点を置くのは、既存のモデルや深い知識から “浅い知識をコンパイルするプロセス” の研究である。
- コンパイルされた浅い知識をどのように利用するかは考えない。

2. 深い知識に基づくＥＳ

- ＭＢＤと同様に、モデルに基づいた推論を行なう。
- 故障診断の対象世界のモデルや物理法則等の記述そのものを研究対象とする。つまり、深い知識をどう表現すると効果的にコンパイルすることができるのかを研究対象とする。
- また、コンパイルされた知識（浅い知識）を推論にどう利用するか、という浅い知識に関しての考察を含む。

これら２つのシステムの特徴と、前節で分析したネットワーク故障診断の特徴や要件から、ネットワーク故障診断ＥＳシステムの枠組としては、対象世界の知識の記述、及び生成された浅い知識の利用方法の考察を含む、深い知識に基づくＥＳとして位置付けるのが有効であると結論付けられる。

2.3.2 深い知識に基づくＥＳ（ＫＣ２）の概要

深い知識に基づくＥＳで現在実装されているシステムとしては、空調器の故障診断を題材とした“ＫＣ”と“ＫＣ２”が存在する。

これらのシステムでは、空調器の主要な部品をとりだし、それら部品間の接続関係と物理パラメタのやりとりを詳細に記述している。このシステムのモデルは、研究対象となっているハードウェアの接続関係であり、Device World (DW) として記述している。そのハードや、それらの間でやりとりされる物理パラメタの動き、変化を拘束する原理原則を Physical World (PW) として記述し、制御や解釈のための知識を CW、IW として記述する。この４つの知識によって対象としている空調器の世界を表現しているのである。

探索戦略としては、与えられた異常徴候から、接続関係と原理原則に基づいて物理パラメタの値を伝播させていき、故障原因を探索していく。

また“ＫＣ２”の特徴のひとつとして挙げられる点は、深い知識から直接的な故障仮説を導くだけでなく、遠因を探索する機能を備えているというところである。

探索の結果システムによって提示される故障原因の候補（ＫＣ２では“故障木”と呼ぶ）は、モデルから考えられるすべての可能性を含んでいる。そして、ある原因の候補から与えられた異常徴候が発現する理由を、モデルを利用して客観的に説明することが可能であるという特徴をもつ。

図７に、深い知識に基づくＥＳ“ＫＣ２”の概念図を示す。

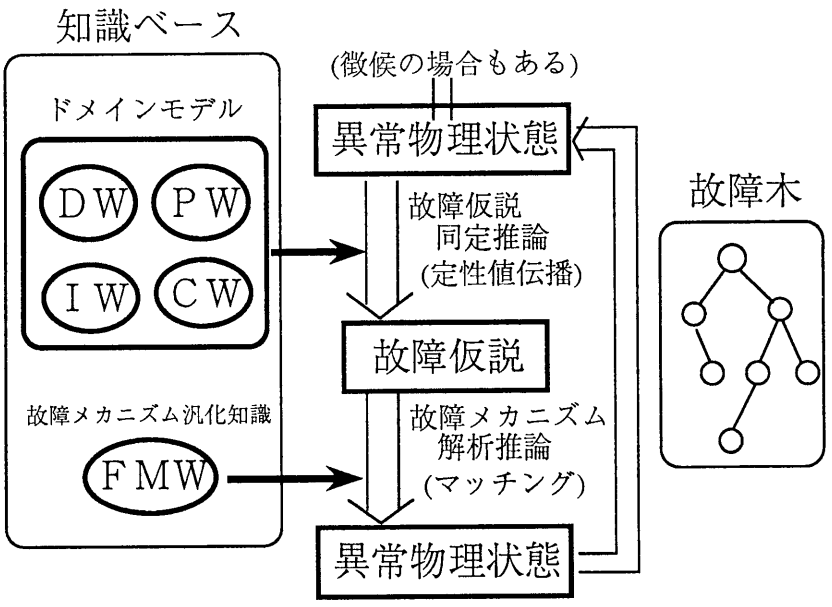


図7 深い知識に基づくＥＳの概念図

今回の研究対象であるネットワークは、ハードウェアとソフトウェアの２つの世界が存在する。そして特に、専門家の診断にはソフトウェアの世界である論理的なプロトコルスタックが有効に活用されていることがわかっている。

このプロトコルスタックを、仮想リンクを含んだプロトコルの接続関係とそれらの間でやりとりされるデータやメッセージとして記述することは、表現方法としては妥当な方法の一つであるといえる。なぜなら、実際にプロトコル間ではデータやメッセージをやりとりし、それに従ってプロトコルが機能しているからである。また、そのデータやメッセージの流れを追う探索戦略も非常に現実的であり、論理的なネットワークを対象とした場合にも有効であるといえる。

以上のように、図６におけるプロトコルスタックとハードウェアモデルを、図７におけるドメインモデルとして表現して、異常徴候を入力し、知識コンパイルすることにより、図６の Fault Hypothesis の集合を図７における故障木として得るような実装は、図６の要求を満たしつつ、同時に知識獲得ボトルネックを解消する手段として有効である。

そこで、今回のプロトタイプの作成にあたっては、このＫＣ２を推論エンジンとして利用し、そこにネットワークに関する深い知識を与えて、故障診断（故障木の出力）を行ない、以下の点について検討することを目的とした。

1. 深い知識に基づくＥＳ（ＫＣ２）の有効性

2. 効果的な故障診断のためのネットワークの記述・表現方法

3　深い知識に基づくネットワーク故障診断ＥＳの実装

3.1　プロトタイプの適用領域の限定とネットワークモデル概要

プロトタイプを作成するにあたって、種々の検証を確実に行なうことを可能とするために、故障診断の対象とするネットワークの構成要素やプロトコル、そして障害状況を以下のように限定した。

3.1.1　プロトタイプの適用領域

1. **ネットワーク構成**（ゲートウェイを介したデータ送信を想定）
   送信ホスト（送信元）、ゲートウェイ、受信ホスト（送信先）

2. **障害状況**
   送信ホストから受信ホストに送信したデータが届かない

3. **プロトコル**
   アプリケーションレベル（ＡＰＬ）、ＴＣＰレベル（ＴＣＰ）、ＩＰレベル（ＩＰ）、物理レベル（ＡＲＰ、ethernet）

4. **その他**
   すべてのプロトコル・ハードの故障可能性は同じであるとする
   （故障しやすさ等の判断基準を現段階では導入しない）

図８に想定したモデルの概念図を示す。

Figure of Full_Logical_Layer of Protocol

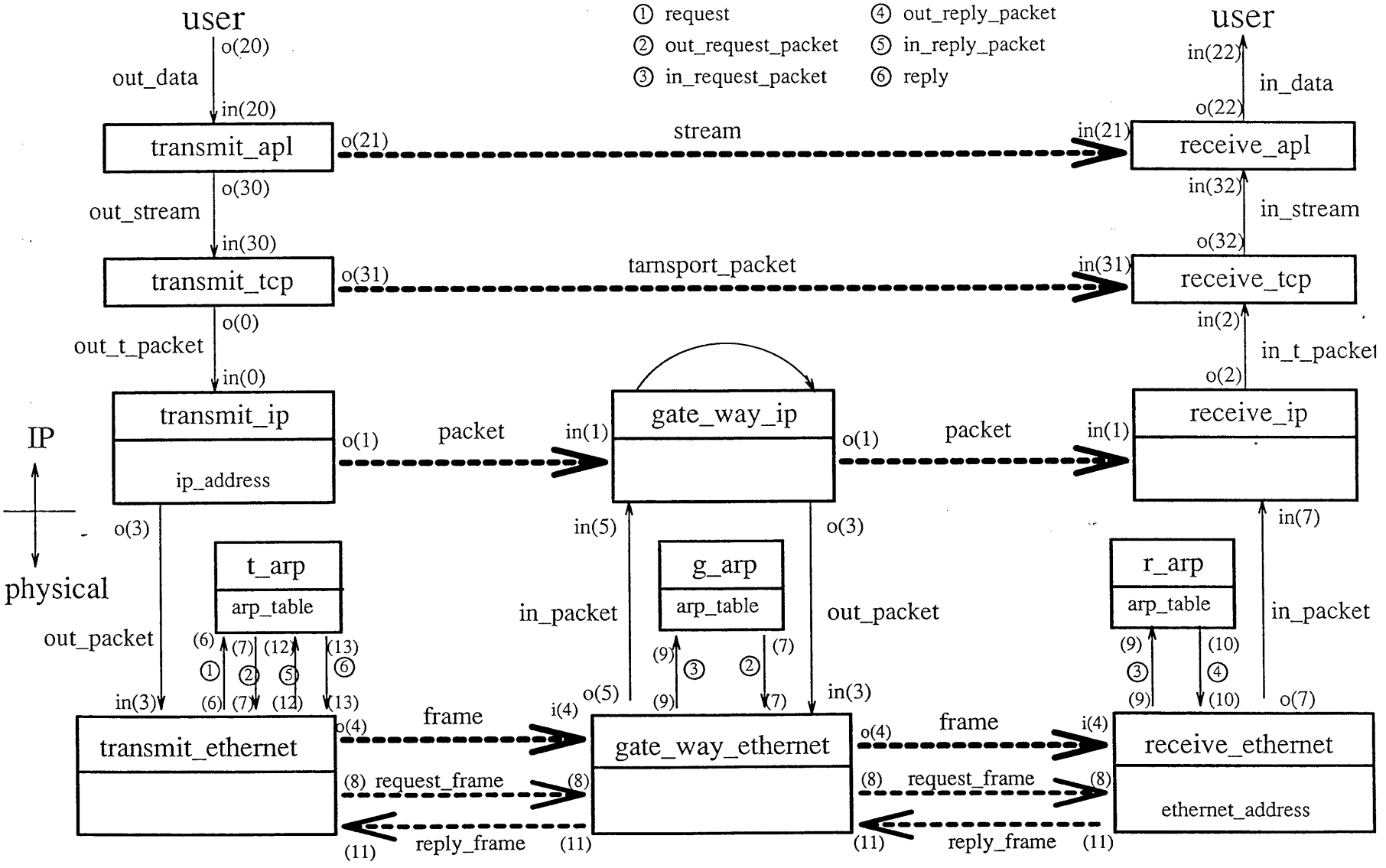


図８ ネットワークのモデル：プロトコル間のパラメタの接続関係

3.1.2 ネットワークモデルの概要

ここでは、表3を基に設計し図8に示した、プロトコルスタックの接続モデルの概要を示す。

表3で示されたプロトコルスタックの記述要素の中で今回実装の対象としたのは、プロトコルを一意に定めるためのプロトコル名、上位、下位プロトコルとの接続関係、またそのプロトコルの機能や信頼性などの、プロトコルスタックの基本的な探索のために不可欠な要素である。

特に接続関係は、上位、下位プロトコルとの実際の接続関係に加え、レイア間で提供しているサービスを仮想リンクとして実装し、さらにやりとりされている情報についても考慮している。また、機能としては主として矢印で示す情報の送信と、情報の保持を実装している。

1. この図は、上に述べた条件において想定される、プロトコル間でのデータのやりとり、つまりプロトコルの接続関係を示している。

2. 実線の枠で囲まれているのが、プロトコルであり部品（Device）として捉えている。プロトコル名は一意に定まる。

3. 横のプロトコル間に走る、太い横の破線矢印は、仮想リンク（レイアが提供するサービス）を示す。

4. 縦のプロトコル間に走る、上下の実線の矢印は、上位プロトコルから下位プロトコルへ、またはその逆の、データの流れを表す。

5. それらの矢印に併記されている値（名前：例えば、stream、out-t-packet）は、その矢印で示される接続でやりとりされるデータにつけられた名前である。これを物理パラメタと呼ぶ。

6. プロトコルと矢印が接する箇所に記されている数字は、接続関係のポートを示す番号である。o(30) はプロトコルから出力（out）されるポート番号３０番の接続であることを示し、in(7) はプロトコルに入力されるポート番号7番の接続であることを示す。
   ただし、これはシステム実装のための値であり、モデルとしては直接関係はない。

3.2　5種類の深い知識の定義と記述：ネットワークにおける位置付け

3.2.1　DW（Device World）

ＤＷは、診断対象に関する情報が記述されている知識である。

今回のネットワークの実装ではハードウェア構成とプロトコル構成に対応する知識（世界）と考え、一つのホスト上に存在するプロトコルそれぞれを、ひとつの“部品”として記述した。

以下に、ＤＷに記述されるべき３つの主要な内容を示し、各々について、ネットワークでは何に対応すると考えるか、またその記述が出力される故障木にどう影響するかを、ＩＰレベルの記述を例にして示す。

1. **部品の機能** 部品が主体的に働きかける処理と考える

   - 送信ＩＰ：パケットをゲートウェイＩＰに送信する。
   - 送信ＩＰ：パケットを送信 ethernet に渡す。
   - 受信ＩＰ：データを受信ＴＣＰに渡す。

   これをＤＷに記述することによって、このような異常が発生した場合には、システムは“ＩＰそのものの機能に異常がある”と結論付ける。それ以外は物理状態としてＩＷに処理をまかせる。

2. **部品間の接続情報**

   ・送信ＩＰはゲートウェイＩＰとポート１で接続しており、そこでやりとりされるのは パケットである。

3. **物理パラメタの集合**

   送信ＩＰに関係するパラメタをすべて列挙する。物理パラメタを伝播させるときにも、ここに記述されているパラメタに関するものだけが対象となる。

   例えば、物理パラメタである“パケット”と“入力パケット”の２つに関するある異常が考えられる時、送信ＩＰの物理パラメタの集合の中にパケットはあるが、入力パケットは記述されていないので、この異常に関しては、送信ＩＰは故障対象としてはみなされないとする。

実際の記述例

```
1  %/*---------------------- DWの書式 -----------------------
2  %dw(部品名, 部品の分類, 部品の耐久度(CW2),
3  %        部品の役割(IW1)のリスト,
4  %        ポートの接続関係のリスト,
5  %        外部構造のリスト, 内部構造のリスト,
6  %        関連パラメータのリスト,
7  %        部品依存型パラメータの変動容易性のリスト).

8  %・部品の分類:          流体(stuff), 導管(conduit), 部品(component)
9  %・部品の耐久度:        0なら壊れやすい、1なら壊れにくい。
10 %・部品の役割のリスト: [[目標部品, パラメータ名, 与える/奪う],...]
11 %・ポートの接続関係:   [[in(_) or _, パラメータ名,out(_) or _,
12 %                        generate or consume or communicate],...]
13 %・外部構造:   [[この部品のポート, 目標部品, 目標部品のポート, パラメータ],.\
..]
14 %・内部構造:   [[in ポートとパラメータのリスト,
15 %                        out ポートとパラメータのリスト],...]
16 %・部品依存型パラメータ:[[パラメータ名, 変n動容易性(CWを参照)],...]

17 %-----------------------------------------------------------*/
18 %/*================ 受信 IP ================*/
19 dw(receive_ip,component,0,
20       [      [receive_tcp,          in_t_packet,    +]
21       ],
22       [      [in(1), packet,        _,      consume],
23              [in(5), in_packet,     _,      consume],
24              [_,     in_t_packet,   out(2), generate]
25       ],
26       [      [in(1), gate_way_ip,           out(1), packet],
27              [in(5), receive_ethernet,      out(5), in_packet],
28              [out(2),receive_tcp,           in(2),  in_t_packet]
29       ],
30       [],
31       [in_t_packet, packet, in_packet ,host_up, lower_level],
       []).
```

プログラム概要説明

例示したプログラムは、実際に受信ＩＰを記述したものである。以下に、プログラムがどういう意味を持つのかを各々の先頭行について例示し、それに対応する行番号を示す。

20：機能

受信ＴＣＰに in-t-packet を渡すのが受信ＩＰの機能である。

22-24：接続関係

in(1) に packet が渡され（届けられ）それは内部で消費する。

26-28：外部構造、内部構造

in(1) は gate-way-ip の out(1) と接続し packet をやりとりしている。

31：物理パラメタリスト

受信ＩＰに関係するパラメタを、すべて列挙する。

3.2.2 ＰＷ（Physical World）

タスクの種類に依存しない、ドメインに固有の原理的知識であり、ＤＷを拘束する原理原則として位置付ける。

物理法則が例として挙げられることが多いが、ネットワークの場合には、ネットワークの情報の変化や、前提条件を表す知識が対応すると考える。そのため、ドメイン全体で共有できるような原則はほとんど存在せず、レイア毎、プロトコル体系毎にまとまる可能性が大きい。

・パケット：（上位層からの）データとＩＰアドレスから構成される

・ＡＲＰテーブル：ＩＰアドレスと物理アドレスから構成される

記述例

```
%/*---------------------- ＰＷの書式 -----------------------

1  %pw(名称, 物理式の一般表現,
2  %        [全パラメータのリスト],
3  %        [左辺のパラメータのリスト],
4  %        [右辺のパラメータのリスト],
5  %        [パラメータ間の関係を表すリスト*]).

6  %註）パラメータ間の関係は以下のように表す。
7  %        ａが増加でｂが増加の時：[ａ，＋，ｂ]
8  %        ａが増加でｂが減少の時：[ａ，－，ｂ] と表す。

 %---------------- ＩＰ 機能の記述 -------------------------

9 %*パケット ＝ アウト・Ｔパケット ＋ ＩＰ・アドレス
```

```
10  pw(pw101,'packet=out_t_packet+ip_address',
11          [packet,out_t_packet,ip_address],
12          [packet],
13          [out_t_packet,ip_address],
14          [[packet,+,out_t_packet],[packet,+,ip_address]]).
```

% *パケット ＝ ホストが正常稼働 ＋ 下位レベルのサービスが保証されている

```
15  pw(pw102,'packet=host_up+lower_level',
16          [packet,host_up,lower_level],
17          [packet],
18          [lower_level,host_up],
19          [[packet,+,host_up],[packet,+,lower_level]]).
```

プログラム概要説明

プログラム例からもわかるように、ＰＷではある物理パラメタに関する記述を複数個並べて記述することが可能である。

上に提示したものは、10-14 行がパケットの生成に関する原則であり、15-19 行がパケットが正常に送信されるための前提条件である。

このＰＷを適用するかしないかは、このパラメタリスト（11、16 行）がＤＷのパラメタリストに含まれているときである。実際、受信ＩＰでパケットに関する探索が行なわれた場合には、pw101 のパラメタがＤＷに記述されていないために pw102 のみが適用され、ノードを展開することになる。

14、19 行に記述されているのが、パラメタの定性値を伝播するための増減関係を記述したものである。

3.2.3 CW（Control World）

診断過程を制御する知識であり、部品の耐久性、パラメタの変動容易性を示す指標と考える。

経験的知識に近いように思われるが、本来は、物理的な制約や、性質等から客観的に導かれるものであり、浅い知識（徴候と故障仮説の関係を記述している）とは異なる。

この知識を１：変動しにくい、と設定すると、故障診断の際に、そのパラメタ以前には故障可能性がないと判断しそこで診断を中断する。故障原因の刈り込みを行なうために利用する。

・ＵＳＥＲからＡＰＬへ渡されるデータは変動しにくい

記述例

```
%/*---------------------- CWの書式 ----------------------

%CW1：パラメータの変動容易性

1  %        cw1(パラメータ名, 変動容易性).

2  %        註）変動容易性＝（０：変動し易い、１：変動しにくい
%                          dependent：この場合パラメータの変動容易性は
%                                     部品情報に依存するので、
%                                     DWに記述されている。）

3  %CW2：部品の耐久性

%        註）ＣＷ２は部品情報としてＤＷに記述されています。

%---------------------------  ＡＰＬ  --------------------------
4  cw1(out_data,          1).          % 送信データ
5  cw1(in_data,           0).          % 受信データ
6  cw1(stream,            0).          % ストリーム
```

今回の実装では、すべてのプロトコル、パラメタの故障可能性を同じと想定している。そのためＣＷの記述はすべて０（変動し易い）として設定している。

ひとつだけ異常が発生しにくいと設定したのが、４行めの ＵＳＥＲから渡されるデータである。

3.2.4 ＩＷ（Interpretation World）

定性値やパラメタの値で表された内部表現を解釈し、人間が持つ概念と対応づけるための知識。浅い知識とはことなる。値の解釈であって、値の原因を表現しているわけではない。

・ＡＲＰテーブルが減少：ＡＲＰテーブルが破壊

記述例

```
%/*---------------------- ＩＷの書式 ----------------------
1 %ＩＷ１：部品の役割による故障仮説の生成
% 註）これは部品依存型の知識なのでＤＷに記述されています。
2 %ＩＷ２：部品の分類による故障仮説の生成
3 % iw2(分類名,パラメータ名,定性値（増減）,故障仮説名).
%----------------------ＡＲＰ__ＴＡＢＬＥ
4 iw2(component, arp_table, -,injury).
                                        % 部品,arp_table, 減少->arp_table 破
損
```

このプログラムが表す意味は、“component に関係する arp-table が - となった時はＡＲＰテーブルが破損している” ということを示している。

3.2.5 ＦＭＷ（Failure Mechanism World）

他の４つの知識を用いて、直接的な故障箇所を同定し、そのさらに原因をたどるための故障メカニズム汎化知識。

実装されたシステムでは、レイア間の移動（レイアの下降）を行なう際に利用する。

・下位レベルに異常がある ＝＞ 物理レイアからＩＰレベルへデータが出ない

記述例

```
%/*
1 %書式）fmw1([KC2 の状態],[KC1 のノードの内容のリスト]).
%*/

2 %/* 下位レベルの異常 <= 受信物理レベルからイン・パケットが出力されない */
3 fmw1([break,generate,OnTheComponent],
4         [[OnTheComponent,in_packet,-]]).
```

これは、ＩＷによって生成された故障仮説を、再度ＫＣ２の物理状態としてマッピングさせるために記述されている。

ここでは受信ＩＰで適用される知識を示している。その意味は “ break という異常状態がこの部品に発生した場合は、この部品に in-packet が渡されないという異常が発生している” ことを表している。

4　プロトタイプと、その出力結果（故障木）に関する考察と評価

前節の各知識の定義に従ってネットワークを記述することにより、ネットワークの故障可能性を示す（説明する）故障木を得た。（出力された故障木の全体は添付資料参照のこと）

本節では、まず出力結果である故障木の書式を説明する。その後、今回作成したプロトタイプのネットワークモデルと故障診断の有効性について検討する。

4.1　故障木の書式

実装によってシステムから出力された “故障木” は、直接故障診断に利用される “浅い知識” そのものである。ここで得られた故障仮説のすべてが与えられた異常徴候を引き起こす原因となり得ることを示している。

これをルールベースで表現していたとすると、原因どうしの関係を掴むことは不可能であるし、原因と徴候の関係を説明することもできない。

深い知識を用いた故障診断の利点のひとつとして、故障原因の説明が原理原則に基づいて正確に行なえるという点が挙げられていた。その特徴を生かすために、故障の原因を的確に表現する必要がある。それと同時に、故障木の表す状態を正確に読みとり、検討していく必要があるといえる。

図９に、出力された故障木（１ノード）の書式を示す。

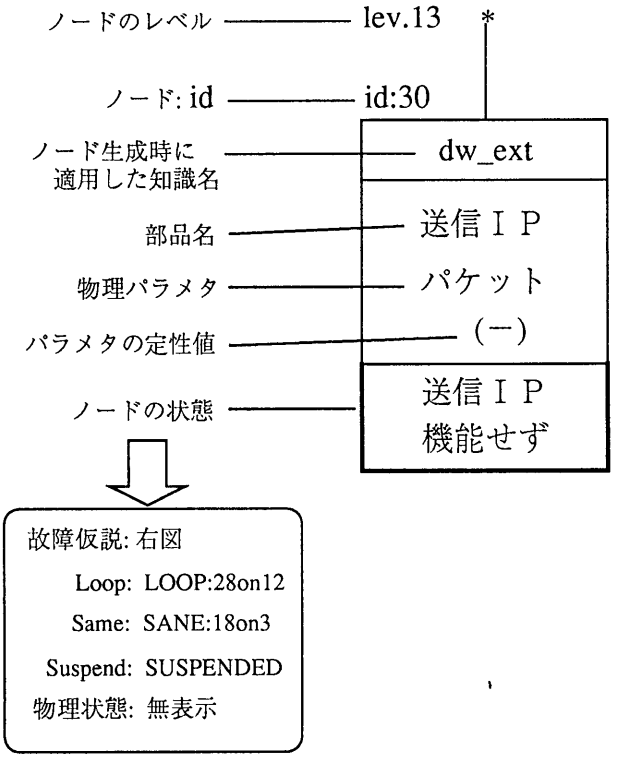


図９ 故障木の書式

4.2 故障木の妥当性（有効性）

出力された故障木の評価は、２つの視点から行なわなければならないと考える。

一つは、現段階での評価であり、それは今後の課題でもある。もう一つはＥＳの可能性として有効か否かの判断である。

まず、現段階で出力されている故障木には以下のような傾向がみられる。その各々について出力された故障木の該当部を示しながら詳細を説明する。

出力された診断結果（故障木）の検証

1. 異常徴候を引き起こしている故障原因（の候補）を提示している。

   出力された故障木は、徴候から導かれる異常状態の集合（74 ノード）から構成されている。

   この故障木によって提示された故障仮説は、すべてが異常徴候の原因となり得る。つまり、徴候と仮説の関係を浅い知識として見た場合には、非常に妥当な故障仮説を生成しているといえる。

   下の故障木の中では、id:15、id:17、id:18、id:19 で示される故障が、id:0(symptom)の徴候を引き起こしている可能性があることを示唆している。

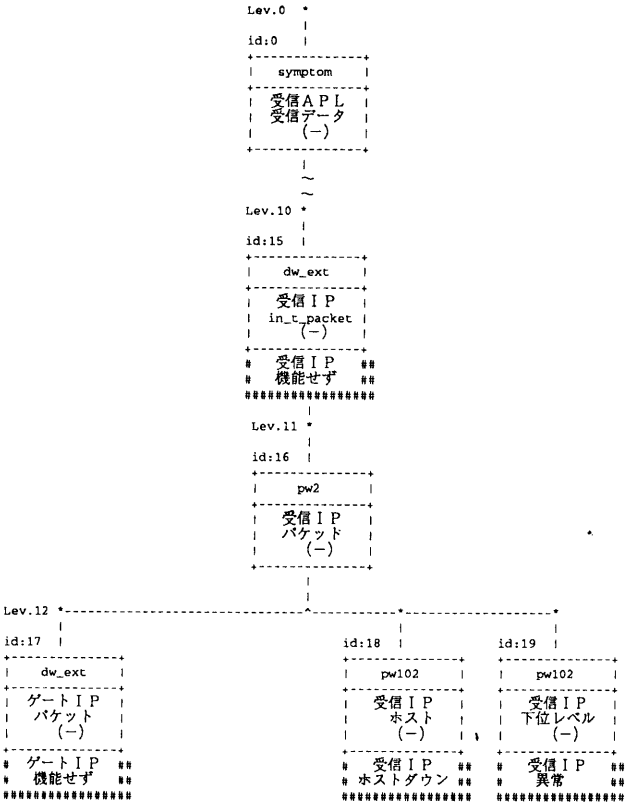


図１０ 故障木 （出力例：故障原因）

2. 非常に詳細な故障可能性を提示している。

故障木全体を眺めると、ネットワークの知識として与えたドメインモデルから、当然出力されるべき故障仮説がすべて網羅されている。プロトコルの接続関係や、データの生成方法といったネットワークに関する詳細な知識から得られるすべての可能性が提示されるのである。

3. 故障木の展開は、ドメインモデルで論理的に説明可能である（故障木の一部を除く）。

故障木の展開（構成）は主としてＤＷとＰＷによって行なわれている。

モデルを用いたＥＳの利点として、原因から障害を論理的に説明することが可能であるという点が挙げられるが、これが可能となるのはＤＷやＰＷといった客観的で論理的な知識を適切に用いて故障木を展開するからである。

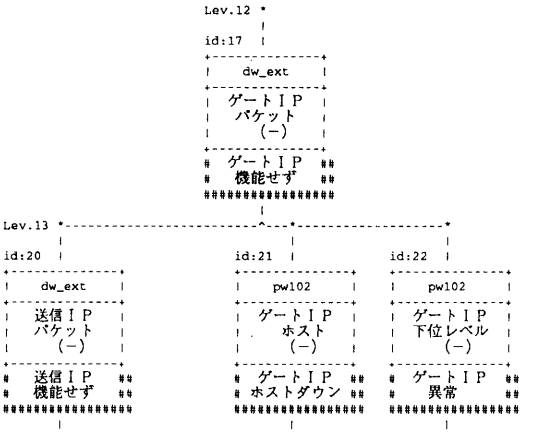


図１１ 故障木（出力例：故障木の展開）

例えば、id:17 から展開されている id:20 はＤＷに記述された接続関係から導かれており、“ゲートＩＰにパケットが届かないということは、送信ＩＰからパケットが送信されていない” 可能性を示す。また、id:21 、id:22 はＰＷから導かれており、パケットが届くための前提条件を示している。

このように、故障仮説から徴候へ至る道筋は、客観的なＤＷやＰＷにより順序立てて説明することが可能となる。

本プロトタイプでも、図８に示したプロトコルの接続関係を、ほぼ正確にトレースする故障木が得られている。

4. パラメタの書式の制約により、パラメタを詳細化しており、理解しずらい。

ＫＣ２では、物理パラメタはそのパラメタが持つ意味や役割で、一意に識別できるように名前づけを行なわなければならない。

故障対象を空調器という物理的な機械とした場合には、物理パラメタ、(例えば冷却水やトルク等）は、それ以上の意味を持つことが少ないために、実際のものの名前から物理パラメタの名前が一意に定まる。

しかし、ネットワークは論理的な世界でもあり、実際のネットワークでは区別する必要のないもの（物理パラメタや接続関係）でも、役割に応じてより細分化した名前をつけることが求められる場合が生じる。

例えば、物理レイアが提供するサービスはフレームを配送することであるが、このフレームの内容にはいくつかの種類がある。ある場合にはブロードキャストのためのフレームであり、ある場合には普通のデータを送信するときのフレームであるというように。

このフレームの種類によって扱い方が異なるため、すべてをフレームという名前で記述することは不可能であった。

そのために、図８では、物理レイアに３つの仮想リンクが張られているかのような表現となり、その結果出力される故障木でも、ミクロ的な出力となり、モデル上での“フレーム”と実際のネットワークでの“フレーム”に違いが出ることとなる。これは、ネットワークの理解を阻害する要因となる可能性がある。

5. パラメタの書式の制約により、つながり方が正常ではない部分がある。

現在出力されている故障木では、２箇所で不自然な展開がみられる。

id:54 は、モデルから見た場合には、id:51 の下ではなく id:55 の下、つまり id:57 として展開されるべきである。

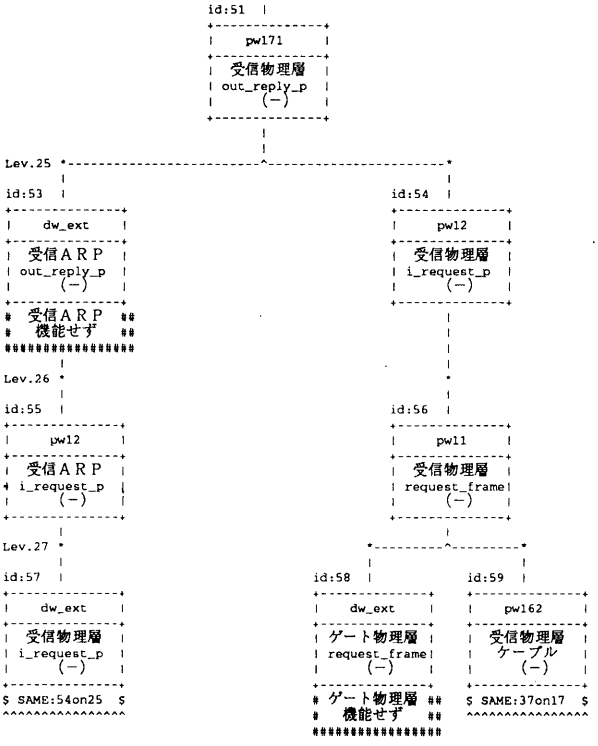


図１２ 故障木（出力例：不自然な展開）

この不自然な展開は、id51 で out-reply-p を展開した時、id:53 を探索すると同時に、ＰＷにも適用できる知識があったために、機械的に id:54 が展開されてしまったために発生している。

これは、ＰＷの適用をＤＷのパラメタリストでは制御しきれない場合があるという問題を示している。

6. ハードウェアへの対応付けが必然的ではない（ＰＷの定義が不十分）。

下の故障木では、“ゲートＩＰにパケットが届かない（id:17)” の原因として３つの故障仮説を提示している。

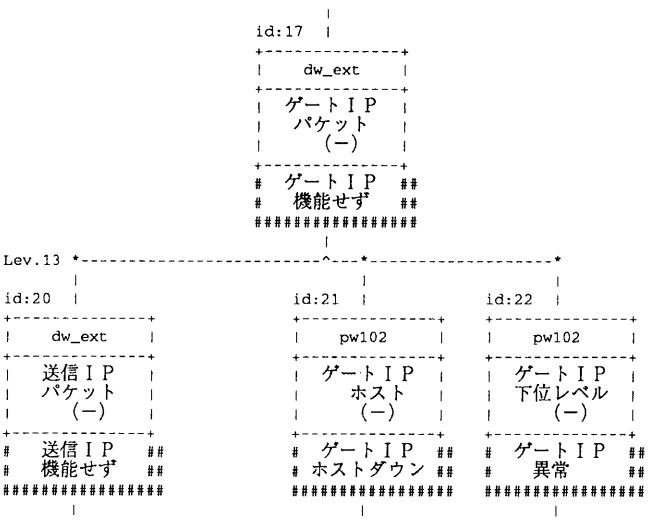


図１３ 故障木（出力例：ハードウェアの同定）

その中の “ゲートウェイホストがダウンしているから（id:21)” は一見妥当な故障仮説であるように思われる。

確かに “ホストがダウンしていればパケットが届かない” は正しい。しかし、ホストがダウンしていればフレームも届かないというように必ずしもパケットが届かないことの客観的に導かれた前提条件ではないのである。

ここにこのノードが出力されたのは、ＰＷにそのように書いたからであって、この知識は原理原則というよりも、むしろ経験則を用いてハードウェアへの対応づけを表現しようとした知識になっている。しかし、プロトコルスタック探索中に突然ハードウェアを同定している、このノードには客観的な裏付けを与えることができない。

以上の結果から、現段階での故障木は、故障の可能性を詳細に提示はしているが、パラメタのやりとりが多くなる部分ほど故障木の展開が理解しにくくなる場合があり、故障の原因を正しく説明できるものではないと結論づけられる。

しかし、浅い知識としてみた場合には、妥当な可能性を提示しており。システム化は比較的有効である。

正確にネットワークを表現した深い知識から導かれた故障可能性（故障木）は、すべての可能性が網羅されており、専門家の故障診断を助けることが可能であると考える。

4.3　ネットワークモデルの改良ポイント

ネットワークをどうモデル化するかがＥＳの質を左右することは前にも述べた。

今回のモデル化は、専門家の故障診断がプロトコルスタックに基づいて行なわれることに着目し、実際のパラメタのやりとりだけでなく、送・受信プロトコル間の仮想リンクをモデルに採り入れている。

これにより、プロトコルレイア単位での故障診断が可能となり、専門家の故障診断過程と同じであり、理解しやすいものとなっている。

その一方で、レイアを採り入れたモデルが実際のネットワークのプロトコルの動きや位置付けを正確に表現してはいないという問題が存在する。

レイアが提供するサービス等の大きな枠組は差異はないが、それをサポートするプロトコルでは食い違いが見られる。

例えば、ＡＲＰはＩＰアドレスに対応する物理アドレスを提供する機能をもっているため、サービスから考えると、物理レイアに属する。しかし、実際には物理アドレスを得るために、ＡＲＰパケットを作りブロードキャストを行なう等ＩＰレベルの機能を持っているのである。

今回のモデリングでは、下位のトラブルの原因を上位に求めることは行なわないため、この現実を正確に表現することは不可能であった。

また、今回のモデリングの対象となったのはプロトコルレイアにとどまり、ハードウェアへの対応付けが行なわれていない。しかし、実際のネットワークは物理的な環境の上にプロトコルスタックが構築され実現されているものであって、相互に影響を与えていることは明らかである。

この考え方をモデルとして取り入れるならば、さきに故障木を示したように、プロトコルスタックの中に突然ハードウェアが単独で出現するような方向性では、経験則に基づくことになり、客観性を求めたシステムの概念に反することになる。そこで、基本的にはプロトコルスタックの世界と物理的な世界を切り離して、それぞれの接続関係やそれを拘束する原理原則に基づいて構築し、関連のある部分では相互にやりとりを行なうことが適切であると考える。すなわち、

「プロトコルスタックとハードウェア構成の各世界に対してＤＷとＰＷをそれぞれ定義し、両方の世界を双方向的に接続するようなモデル化が今後必要である。」

今回のプトロタイプ作成を通して、構築すべきネットワークモデルの概観（詳細ではない）は図１４のようになると考える。

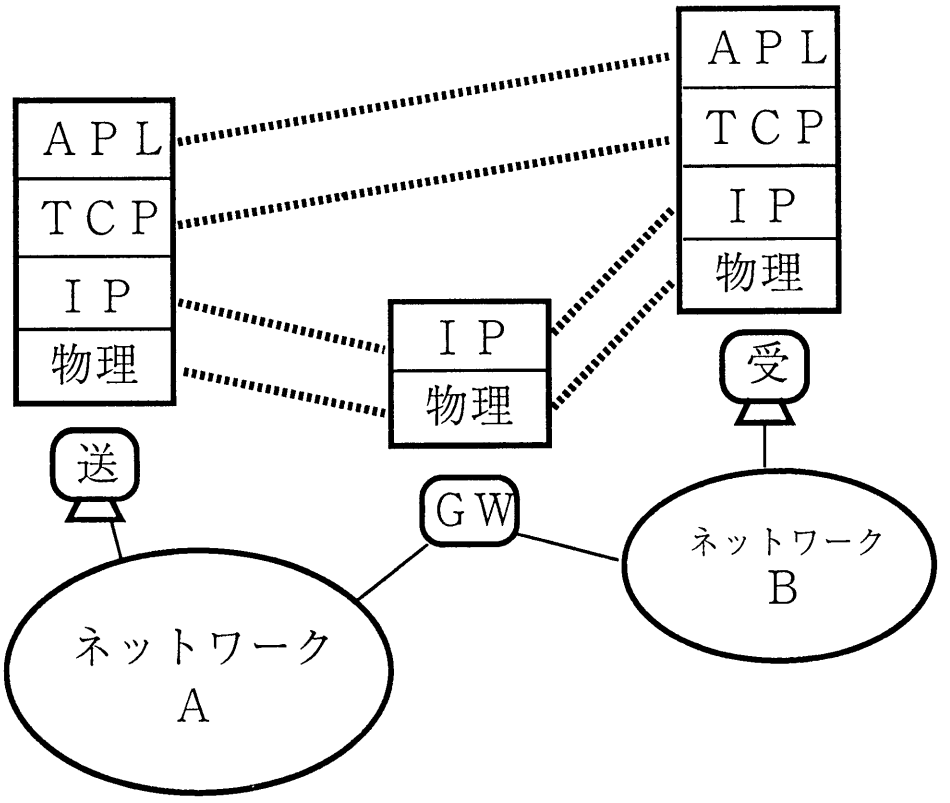


図１４ 構築すべきネットワークモデルの概観

5 今後の課題（展望）

5.1 今後の作業

現在のＫＣ２の仕様では、異なる意味を持つパラメタはすべて別の名前をつけ、接続するポートや障害状況、制御情報等を個別に管理しなければならない。この制約は、実際の物（空調器の部品等）を扱うシステムであれば問題にならない。

しかし、本研究の対象であるネットワークの性格から、この制約に従うことは非常に困難である。なぜなら同じパラメタであっても、その内容によってプロトコルの対処が異なる場合が存在するからである。

例えば、ゲートウェイを介してパケットを送信する場合を考える。送信パケットには送信先のＩＰアドレスが入っている。ゲートウェイがパケットを受けとると、送信先ＩＰアドレスを確認して自分宛のパケットであれば受信し、そうでなければ、適当なネットワークに送り出すのである。

またもしその内容毎にパラメタを別のものとして管理したとすると、１つのパラメタがアドレスの数だけ存在するという事態が生じてくる。

このような問題に対応するために、パラメタの内容を考慮し、伝播することが可能となるように、プログラムの改修を行なう。

```
stream -> stream(),stream(arp_request) ...
```

5.2 プロトコルの世界とハードの世界の切りわけ

今後研究を進める上で必要な作業としては、ハードウェアモデル（ネットワーク機器構成）と論理的なとプロトコルスタックを切り離すことが可能かどうか、客観的（理論的）な判断基準を提示することである。

深い知識に基づくＥＳという枠組でネットワークを考える時、まず、各々の知識世界に対応するものとして、ＤＷにはハードウェアを、ＰＷにはソフトウェアを対応付けることを考えた。

しかし、モデル化と実装（特にプロトコル世界を中心とした実装）を行なう中で、ハードウェアとプロトコルスタックは、相互に依存しているが、１対１に対応しているわけではなく、また、それぞれを拘束している世界は別のものではないかと考えるに至った。つまり、ハードウェアにも接続関係とその背後にあるプロトコルに依存しない原理原則があり、またソフトウェアにもプロトコルの接続関係とハードウェアに依存しない原理原則が存在する。

そこで、その各々の世界を独立した領域と見なし、知識を記述するのが良いのではないかと考える。

しかし、この両世界に共通して、あるいは束縛しあっている部分も明らかに存在しており、またプロトコルの探索から実際のハードウェアの同定を行なうことを考えると、知識の記述方法をどちらかの世界に固定的なものとしないように顧慮する必要がある。

6 おわりに

ネットワークの専門家から事例を収集し、モデル化、そしてプロトタイプの実装と行なってきた。ここまできてようやく、ネットワーク故障診断ＥＳとしての良否や可能性を検討できるようになった。

ヘテロなネットワークはプロトコルやハードの組み合わせによって構築されているシステムであるために、その規模や性質は千差万別であるといえる。そのために、実装を伴わないモデル化、あるいはモデルや概念を欠いた実装であっては、このヘテロなネットワークを対象としたＥＳを構築することは不可能であった。

現段階のモデルやプロトタイプは、まだネットワークのごく一部を取り挙げて表現したに留まっており、すべての問題について検討したわけではない。しかし、本研究で明らかになった主要なことがらを挙げてみると、

1. ネットワークの故障診断が、プロトコルスタックの探索問題として認識でき、モデルを利用することが可能であること
2. ハードウェアとソフトウェアに関する深い知識（モデル）を用いてネットワークの故障診断を行なうことが可能であること
3. 深い知識を用いて行なう故障診断は、非常に詳細な故障原因の候補を提示することが可能であること
4. 仮想レイアが提供するサービスと、物理的なパラメタの実際のやりとりが把握できれば、深い知識の構築は非常に容易であること

これらの可能性は、これまで“勘”や“ノウハウ”といった個人的で他者との共有が難しい知識で行なわれていたネットワーク故障診断に、客観性と汎用性を与えることが可能であるということを示唆している。

この点で、本研究はネットワークＥＳの枠組としてのひとつの切口を提示し得たと考える。

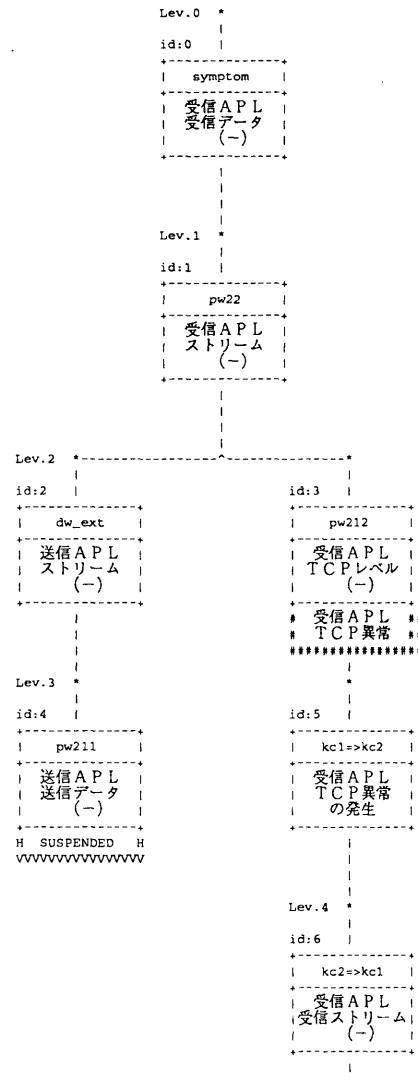
参考文献

[1] de Kleer, J., Williams, B.C.: Diagnosing Multiple Faults, Artificial Intelligence, Vol.32, No.1,pp.97-130 (1987)

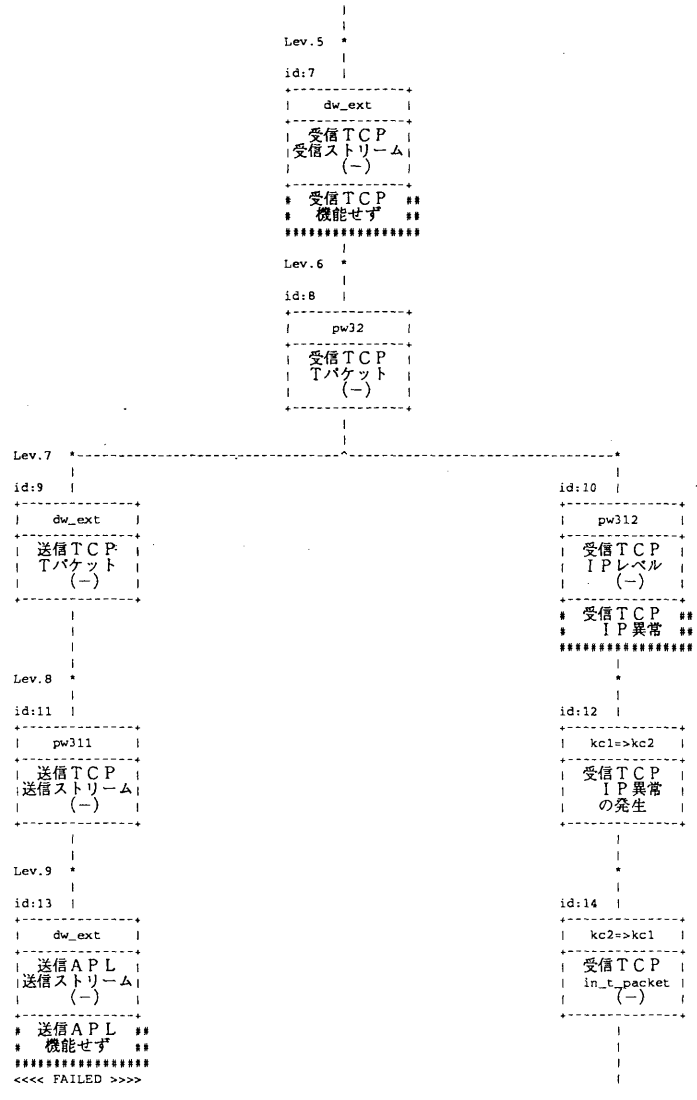
[2] R. Reiter : A Theory of Diagnosis from First Principles, Artificial Intelligence, Vol.32, No.1,pp.57-96 (1987)

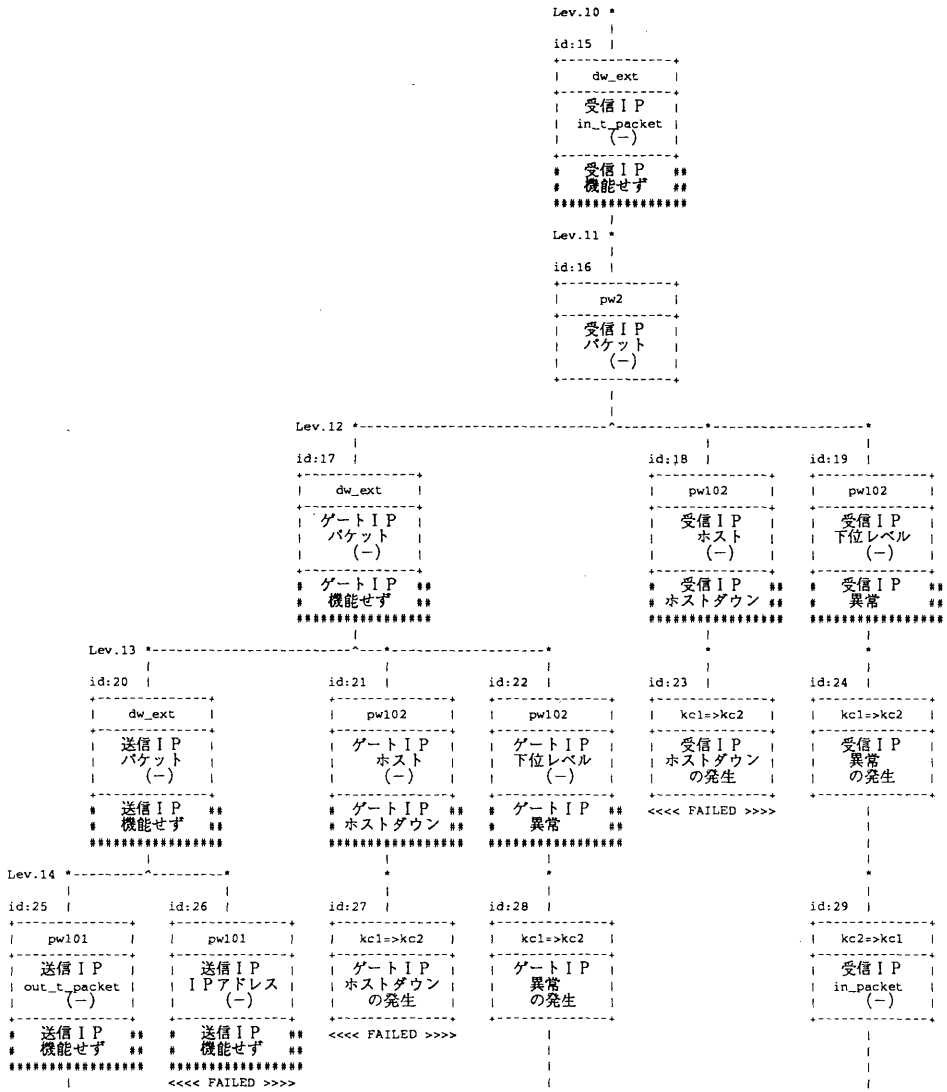
[3] 長田、篠田、山口、落水:' ヘテロなネットワークの故障診断に関わる知識の獲得とモデル化', 日本ソフトウェア科学会第１０回大会論文集,pp265-268 (1993).

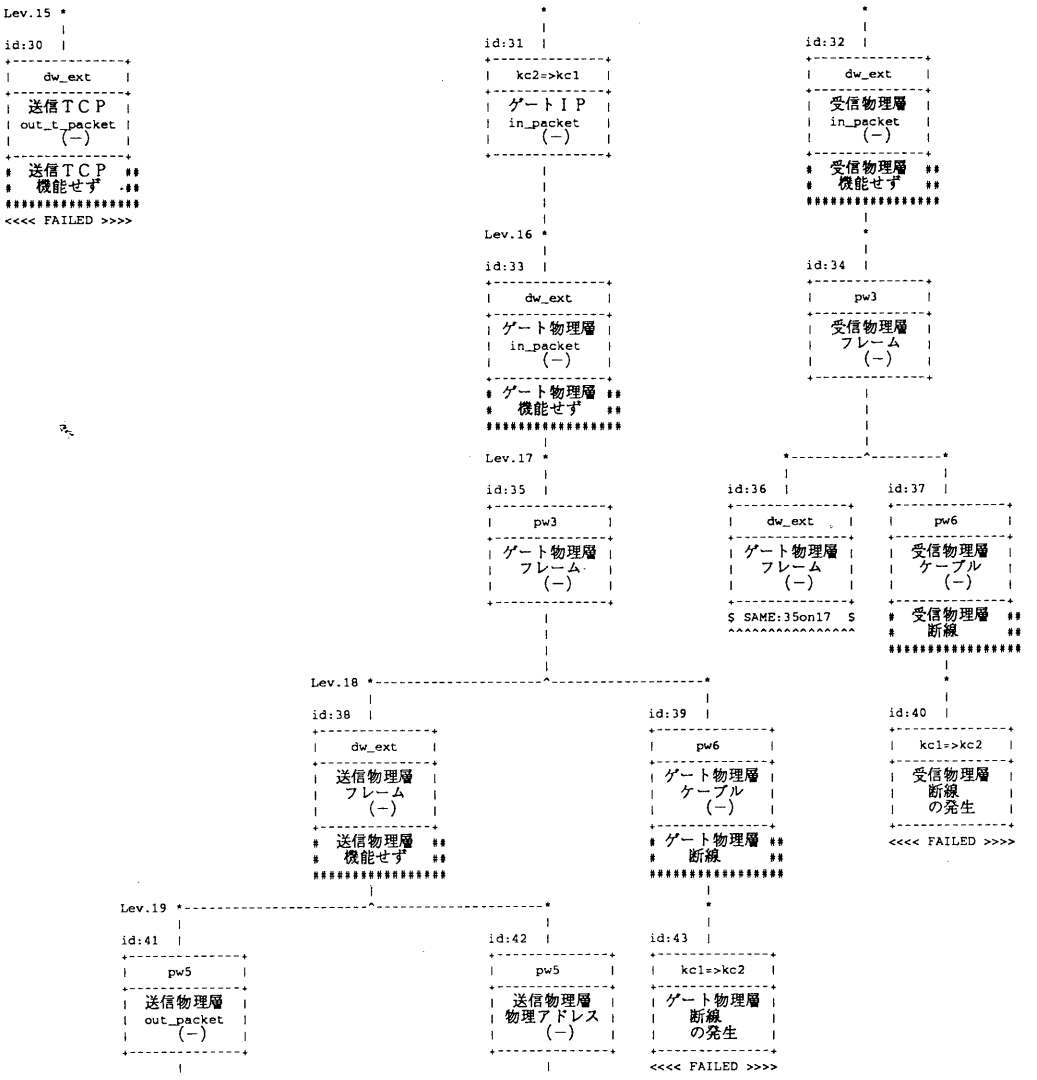
[4] 山口:' 深い知識に基づくエキスパートシステム', チューリング・マシン,Vol.2 No.5,pp9-19(1989).

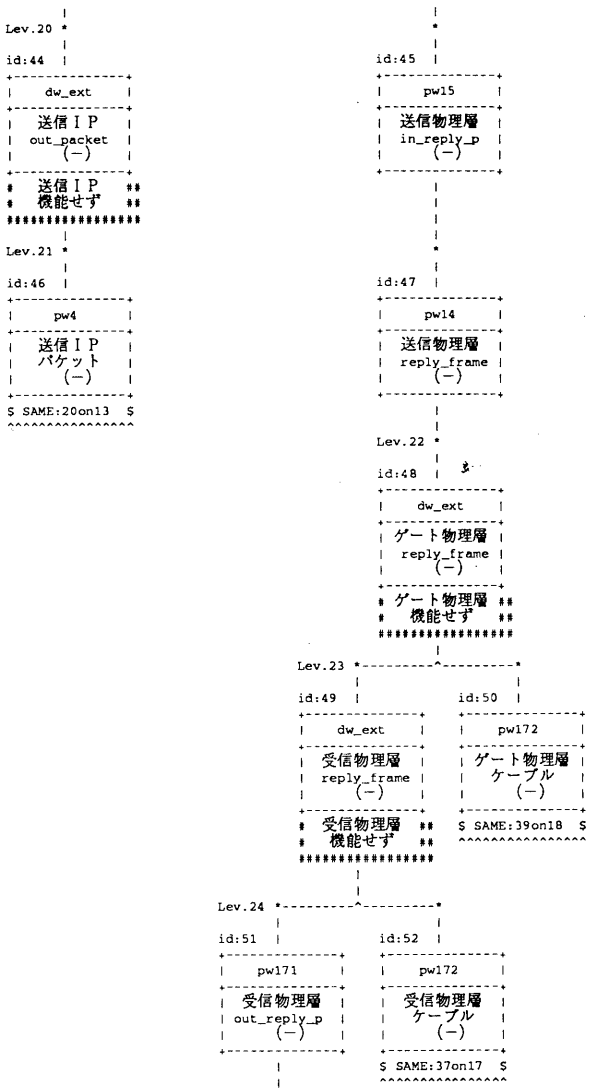


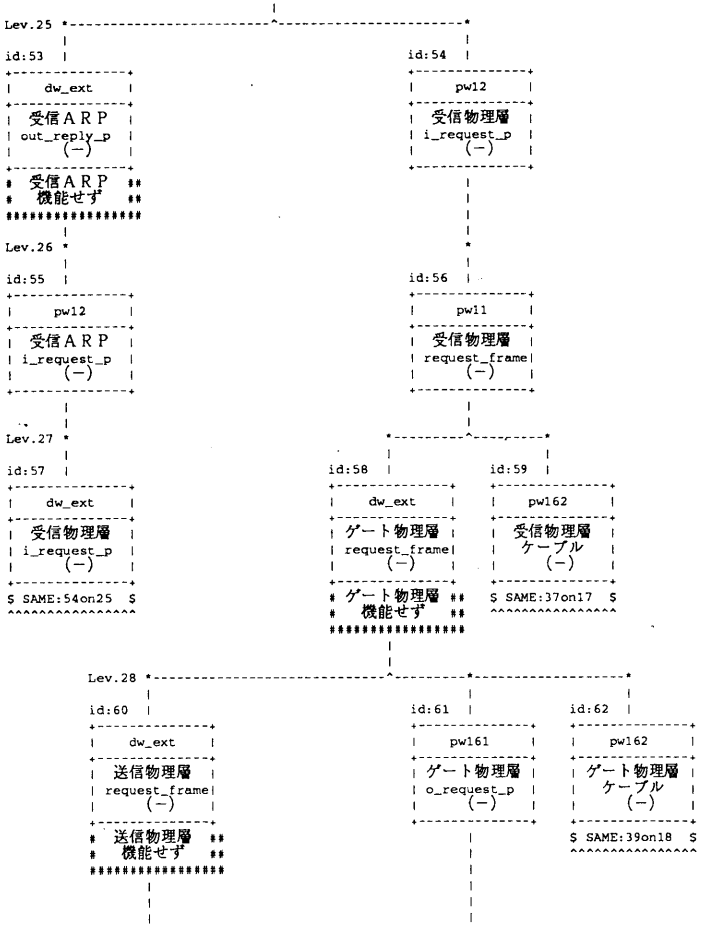
添付資料Ａ：故障木（今回の実装で出力されるすべてのノードを含む）

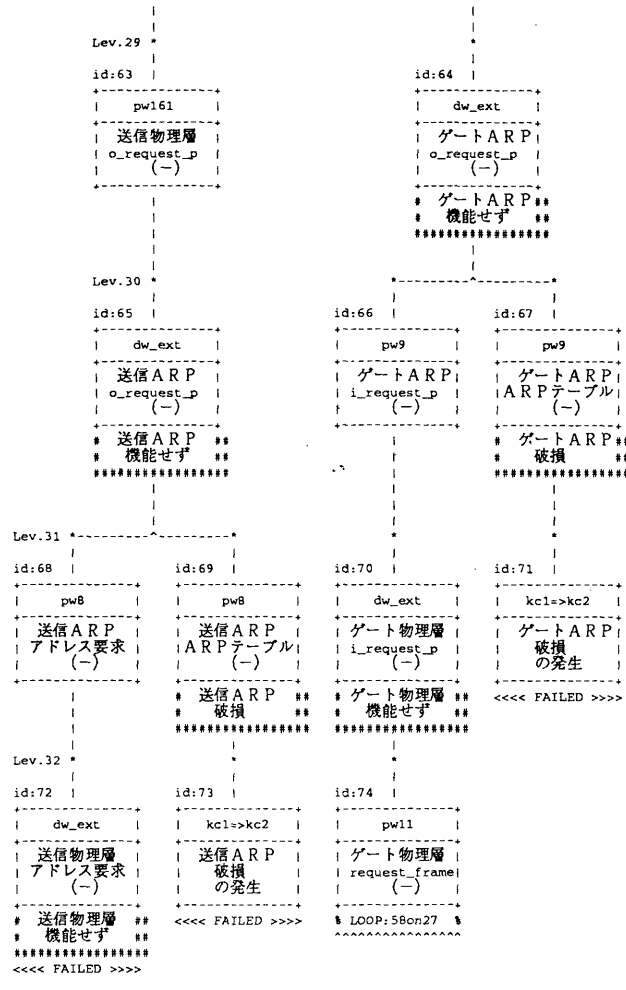

Lev.5
id:7
dw_ext
受信ＴＣＰ
受信ストリーム
(−)
受信ＴＣＰ
機能せず
Lev.6
id:8
pw32
受信ＴＣＰ
Ｔパケット
(−)
Lev.7
id:9
dw_ext
送信ＴＣＰ
Ｔパケット
(−)
id:10
pw312
受信ＴＣＰ
ＩＰレベル
(−)
受信ＴＣＰ
ＩＰ異常
Lev.8
id:11
pw311
送信ＴＣＰ
送信ストリーム
(−)
id:12
kc1=>kc2
受信ＴＣＰ
ＩＰ異常
の発生
Lev.9
id:13
dw_ext
送信ＡＰＬ
送信ストリーム
(−)
送信ＡＰＬ
機能せず
<<<< FAILED >>>>
id:14
kc2=>kc1
受信ＴＣＰ
in_t_packet
(−)


Lev.10
id:15
dw_ext
受信ＩＰ
in_t_packet
(−)
受信ＩＰ
機能せず
Lev.11
id:16
pw2
受信ＩＰ
パケット
(−)
Lev.12
id:17
dw_ext
ゲートＩＰ
パケット
(−)
ゲートＩＰ
機能せず
id:18
pw102
受信ＩＰ
ホスト
(−)
受信ＩＰ
ホストダウン
id:19
pw102
受信ＩＰ
下位レベル
(−)
受信ＩＰ
異常
Lev.13
id:20
dw_ext
送信ＩＰ
パケット
(−)
送信ＩＰ
機能せず
id:21
pw102
ゲートＩＰ
ホスト
(−)
ゲートＩＰ
ホストダウン
id:22
pw102
ゲートＩＰ
下位レベル
(−)
ゲートＩＰ
異常
id:23
kc1=>kc2
受信ＩＰ
ホストダウン
の発生
<<<< FAILED >>>>
id:24
kc1=>kc2
受信ＩＰ
異常
の発生
Lev.14
id:25
pw101
送信ＩＰ
out_t_packet
(−)
送信ＩＰ
機能せず
id:26
pw101
送信ＩＰ
IPアドレス
(−)
送信ＩＰ
機能せず
<<<< FAILED >>>>
id:27
kc1=>kc2
ゲートＩＰ
ホストダウン
の発生
<<<< FAILED >>>>
id:28
kc1=>kc2
ゲートＩＰ
異常
の発生
id:29
kc2=>kc1
受信ＩＰ
in_packet
(−)


Lev.15
id:30
dw_ext
送信ＴＣＰ
out_t_packet
(−)
送信ＴＣＰ
機能せず
<<<< FAILED >>>>
id:31
kc2=>kc1
ゲートＩＰ
in_packet
(−)
Lev.16
id:33
dw_ext
ゲート物理層
in_packet
(−)
ゲート物理層
機能せず
Lev.17
id:35
pw3
ゲート物理層
フレーム
(−)
id:32
dw_ext
受信物理層
in_packet
(−)
受信物理層
機能せず
id:34
pw3
受信物理層
フレーム
(−)
id:36
dw_ext
ゲート物理層
フレーム
(−)
$ SAME:35on17 $
id:37
pw6
受信物理層
ケーブル
(−)
受信物理層
断線
id:40
kc1=>kc2
受信物理層
断線
の発生
<<<< FAILED >>>>
Lev.18
id:38
dw_ext
送信物理層
フレーム
(−)
送信物理層
機能せず
id:39
pw6
ゲート物理層
ケーブル
(−)
ゲート物理層
断線
Lev.19
id:41
pw5
送信物理層
out_packet
(−)
id:42
pw5
送信物理層
物理アドレス
(−)
id:43
kc1=>kc2
ゲート物理層
断線
の発生
<<<< FAILED >>>>


Lev.20
id:44
dw_ext
送信ＩＰ
out_packet
(−)
送信ＩＰ
機能せず
Lev.21
id:46
pw4
送信ＩＰ
パケット
(−)
$ SAME:20on13 $
id:45
pw15
送信物理層
in_reply_p
(−)
id:47
pw14
送信物理層
reply_frame
(−)
Lev.22
id:48
dw_ext
ゲート物理層
reply_frame
(−)
ゲート物理層
機能せず
Lev.23
id:49
dw_ext
受信物理層
reply_frame
(−)
受信物理層
機能せず
id:50
pw172
ゲート物理層
ケーブル
(−)
$ SAME:39on18 $
Lev.24
id:51
pw171
受信物理層
out_reply_p
(−)
id:52
pw172
受信物理層
ケーブル
(−)
$ SAME:37on17 $


Lev.25
id:53
dw_ext
受信ＡＲＰ
out_reply_p
(−)
受信ＡＲＰ
機能せず
id:54
pw12
受信物理層
i_request_p
(−)
Lev.26
id:55
pw12
受信ＡＲＰ
i_request_p
(−)
id:56
pw11
受信物理層
request_frame
(−)
Lev.27
id:57
dw_ext
受信物理層
i_request_p
(−)
$ SAME:54on25 $
id:58
dw_ext
ゲート物理層
request_frame
(−)
ゲート物理層
機能せず
id:59
pw162
受信物理層
ケーブル
(−)
$ SAME:37on17 $
Lev.28
id:60
dw_ext
送信物理層
request_frame
(−)
送信物理層
機能せず
id:61
pw161
ゲート物理層
o_request_p
(−)
id:62
pw162
ゲート物理層
ケーブル
(−)
$ SAME:39on18 $


Lev.29
id:63
pw161
送信物理層
o_request_p
(−)
id:64
dw_ext
ゲートARP
o_request_p
(−)
ゲートARP
機能せず
Lev.30
id:65
dw_ext
送信ARP
o_request_p
(−)
送信ARP
機能せず
id:66
pw9
ゲートARP
i_request_p
(−)
id:67
pw9
ゲートARP
ARPテーブル
(−)
ゲートARP
破損
Lev.31
id:68
pw8
送信ARP
アドレス要求
(−)
id:69
pw8
送信ARP
ARPテーブル
(−)
送信ARP
破損
id:70
dw_ext
ゲート物理層
i_request_p
(−)
ゲート物理層
機能せず
id:71
kc1=>kc2
ゲートARP
破損
の発生
<<<< FAILED >>>>
Lev.32
id:72
dw_ext
送信物理層
アドレス要求
(−)
送信物理層
機能せず
<<<< FAILED >>>>
id:73
kc1=>kc2
送信ARP
破損
の発生
<<<< FAILED >>>>
id:74
pw11
ゲート物理層
request_frame
(−)
LOOP:58on27

```
%/*********************************************************
%     ======= 知識コンパイラ ver1.0 =======
%        ＤＷ(Device World)：対象モデル
%                                          oct 7 1993
%                  アプリケーションへ拡張したモデル構成の実装
%*********************************************************/

%/*---------------------- ＤＷの書式 -----------------------

%dw(部品名,部品の分類,部品の耐久度(ＣＷ２),
%       部品の役割(ＩＷ１)のリスト,
%       ポートの接続関係のリスト,
%       外部構造のリスト,内部構造のリスト,
%       関連パラメータのリスト,
%       部品依存型パラメータの変動容易性のリスト).

%・部品の分類：          流体(stuff),導管(conduit),部品(component)
%・部品の耐久度：        ０なら壊れやすい、１なら壊れにくい。
%・部品の役割のリスト：  [[目標部品,パラメータ名,与える／奪う],...]
%・ポートの接続関係：    [[in(_) or _,パラメータ名,out(_) or _,
%                                generate or consume or communicate],...]
%・外部構造：    [[この部品のポート,目標部品,目標部品のポート,パラメータ],...]
%・内部構造：    [[inポートとパラメータのリスト,
%                        outポートとパラメータのリスト],...]
%・部品依存型パラメータ：[[パラメータ名,変n動容易性(ＣＷを参照)],...]

%---------------------------------------------------------*/

%---------------------- ＵＳＥＲ ---------------------------

dw(user,component,1,
        [       [transmit_apl,  out_data,       +]
        ],
        [       [in(22),receive_apl,            _,      consume],
                [_,     in_stream,              out(20),generate]
        ],
        [       [in(22),receive_apl,    in(22), in_data],
                [out(20),transmit_apl,  in(20), out_data]
        ],
        [],
        [out_data, in_data],
        []).

%---------------------- ＡＰＬ ---------------------------

%/*================ 送信 ＡＰＬ ================*/
dw(transmit_apl,component,0,
        [       [transmit_tcp,  out_data,       +]
        ],
        [       [in(20),out_data,       _,      consume],
                [_,     stream,         out(21),generate],
                [_,     out_stream,     out(30),generate]
        ],
        [       [in(20),user,           out(20),out_data],
                [out(21),receive_apl,   in(21), stream],
                [out(30),transmit_tcp,  in(30), out_stream]
        ],
        [],
        [out_stream, out_data, stream],
        []).

%/*================ 受信 ＡＰＬ ================*/
dw(receive_apl,component,0,
        [       [user,  in_data,        +]
        ],
```

```
66          [       [in(21),stream,         _,      consume],
67                  [in(32),in_stream,      _,      consume],
68                  [_,     in_data,        out(22),generate]
69          ],
70          [       [in(21),transmit_apl,   out(21),stream],
71                  [in(32),receive_tcp,    out(32),in_stream],
72                  [out(22),user,          in(22), in_data]
73          ],
74          [],
75          [in_data, stream, in_stream, tcp_level],
76          []).
77
78  %---------------------- ＴＣＰ ----------------------------
79
80  %/*================ 送信 ＴＣＰ ================*/
81  dw(transmit_tcp,component,0,
82          [       [transmit_ip,   out_t_packet,   +]
83          ],
84          [       [in(30),out_stream,             _,      consume],
85                  [_,     transport_packet,       out(31),generate],
86                  [_,     out_t_packet,           out(0), generate]
87          ],
88          [       [in(30),transmit_apl,   out(30),out_stream],
89                  [out(31),receive_tcp,   in(31), transport_packet],
90                  [out(0),transmit_ip,    in(0),  out_t_packet]
91          ],
92          [],
93          [out_t_packet, out_stream, transport_packet],
94          []).
95
96  %/*================ 受信 ＴＣＰ ================*/
97  dw(receive_tcp,component,0,
98          [       [receive_apl,   in_stream,      +]
99          ],
100         [       [in(31),transport_packet,       _,      consume],
101                 [in(2), in_t_packet,            _,      generate],
102                 [_,     in_stream,              out(32),generate]
103         ],
104         [       [in(31),transmit_tcp,   out(31),transport_packet],
105                 [in(2), receive_ip,     out(2), in_t_packet],
106                 [out(32),receive_apl,   in(32), in_stream]
107         ],
108         [],
109         [in_t_packet, in_stream, transport_packet, ip_level],
110         []).
111
112 %*---------------------- ＩＰ ------------------------
113
114 %/*================ 送信 ＩＰ ================*/
115 dw(transmit_ip,component,0,
116         [       [gate_way_ip,           packet, +],
117                 [transmit_ethernet,     out_packet,     +],
118                 [transmit_ip,           ip_address,     +]
119         ],
120         [       [in(0), out_t_packet,   _,      consume],
121                 [_,     packet,         out(1), generate],
122                 [_,     out_packet,     out(3), generate]
123         ],
124         [       [in(0), transmit_tcp,           out(0), out_t_packet],
125                 [out(1),receive_ip,             in(1),  packet],
126                 [out(3),transmit_ethernet,      in(3),  out_packet]
127         ],
128         [],
129         [out_t_packet, packet, out_packet, ip_address],
130         []).
```

```
131
132 %/*================ ゲートウェイ ＩＰ ================*/
133 dw(gate_way_ip,component,0,
134         [       [receive_ip,            packet, +],
135                 [gate_way_ethernet,     out_packet,     +]
136         ],
137         [       [in(1), packet,         out(1), connection],
138                 [in(5), in_packet,      _,      consume],
139                 [_,     out_packet,     out(3), generate]
140         ],
141         [       [in(1), transmit_ip,            out(1), packet],
142                 [in(5), gate_way_ethernet,      out(5), in_packet],
143                 [out(1),receive_ip,             in(1),  packet],
144                 [out(3),gate_way_ethernet,      out(3), out_packet]
145         ],
146         [],
147         [packet, in_packet, host_up, lower_level],
148         []).
149
150 %/*================ 受信 ＩＰ ================*/
151 dw(receive_ip,component,0,
152         [       [receive_tcp,           in_t_packet,    +]
153         ],
154         [       [in(1), packet,         _,      consume],
155                 [in(5), in_packet,      _,      consume],
156                 [_,     in_t_packet,    out(2), generate]
157         ],
158         [       [in(1), gate_way_ip,            out(1), packet],
159                 [in(5), receive_ethernet,       out(5), in_packet],
160                 [out(2),receive_tcp,            in(2),  in_t_packet]
161         ],
162         [],
163         [in_t_packet, packet, in_packet ,host_up, lower_level],
164         []).
165
166 %---------------------- データリンク 層 --------------------
167
168 %/*================ 送信 ＡＲＰ ================*/
169 dw(t_arp,component,0,
170         [       [transmit_ethernet,     out_request_packet,     +],
171                 [transmit_ethernet,     ethernet_address,       +]
172         ],
173         [       [in(6), request,                _,      consume],
174                 [in(12),in_reply_packet,        _,      consume],
175                 [_,     out_request_packet,     out(7), generate],
176                 [_,     reply,                  out(13), generate]
177         ],
178         [       [in(6), transmit_ethernet,      out(6), request],
179                 [in(12),transmit_ethernet,      out(12),in_reply_packet],
180                 [out(7),transmit_ethernet,      in(7),  out_request_packet],
181                 [out(13),transmit_ethernet,     in(13), reply]
182         ],
183         [],
184         [request, arp_table, reply,
185                 out_request_packet, in_reply_packet],
186         []).
187
188 %/*================ ゲートウェイ ＡＲＰ ================*/
189 dw(g_arp,component,0,
190         [       [gate_way_ethernet,     out_request_packet,     +]
191         ],
192         [       [in(9), in_request_packet,      _,      consume],
193                 [_,     out_request_packet,     out(7), generate]
194         ],
195         [       [in(9), gate_way_ethernet,      out(9), in_request_packet],
```

```
196                 [out(7),gate_way_ethernet,      in(7),  out_request_packet]
197         ],
198         [],
199         [ arp_table, out_request_packet, in_request_packet],
200         []).
201 %/*================ 受信 ＡＲＰ ================*/
202 dw(r_arp,component,0,
203         [       [receive_ethernet,      out_reply_packet,       +]
204         ],
205         [       [in(9), in_request_packet,      _,      consume],
206                 [_,     out_reply_packet,       out(10), generate]
207         ],
208         [       [in(9), receive_ethernet,       out(9), in_request_packet],
209                 [out(10),receive_ethernet,      in(10), out_reply_packet]
210         ],
211         [],
212         [arp_table, out_reply_packet, in_request_packet],
213         []).
214
215 %---------------------- Ｅｔｈｅｒｎｅｔ --------------------
216
217 %/*================ 送信 Ｅｔｈｅｒｎｅｔ ================*/
218 dw(transmit_ethernet,component,0,
219         [       [gate_way_ethernet,     frame,          +],
220                 [gate_way_ethernet,     request_frame,  +],
221                 [t_arp,                 request,        +],
222                 [t_arp,                 in_reply_packet,+]
223         ],
224         [       [in(3), out_packet,             _,      consume],
225                 [in(7), out_request_packet,     _,      consume],
226                 [in(11),reply_frame,            _,      consume],
227                 [in(13),reply,                  _,      consume],
228                 [_,     frame,          out(4), generate],
229                 [_,     request,        out(6), generate],
230                 [_,     request_frame,  out(8), generate],
231                 [_,     in_reply_packet,out(12),generate]
232         ],
233         [       [in(3), transmit_ip,            out(3), out_packet],
234                 [in(7), t_arp,                  out(6), out_request_packet],
235                 [in(11),gate_way_ethernet,      out(11),reply_frame],
236                 [in(13),t_arp,                  out(13),reply],
237                 [out(4),gate_way_ethernet,      in(4),  frame],
238                 [out(6),t_arp,                  in(6),  request],
239                 [out(8),gate_way_ethernet,      in(8),  request_frame],
240                 [out(12),t_arp,                 in(12), in_reply_packet]
241         ],
242         [],
243         [frame, request_frame, reply_frame, out_packet, reply,
244                 request, out_request_packet, ethernet_address,
245                 in_reply_packet],
246         []).
247
248 %/*================ ゲートウェイ Ｅｔｈｅｒｎｅｔ ================*/
249 dw(gate_way_ethernet,component,0,
250         [       [receive_ethernet,      frame,          +],
251                 [receive_ethernet,      request_frame,  +],
252                 [transmit_ethernet,     reply_frame,    +],
253                 [gate_way_ip,           in_packet,      +],
254                 [g_arp,                 in_request_packet,      +]
255         ],
256         [       [in(3), out_packet,             _,      consume],
257                 [in(4), frame,                  _,      consume],
258                 [in(7), out_request_packet,     _,      consume],
259                 [in(8), request_frame,          _,      consume],
260                 [in(11),reply_frame,            out(11),connection],
```

```
261                 [_,     frame,                  out(4), generate],
262                 [_,     in_packet,              out(5), generate],
263                 [_,     request_frame,          out(8), generate],
264                 [_,     in_request_packet,      out(9), generate]
265         ],
266         [       [in(3), gate_way_ip,            out(3), out_packet],
267                 [in(4), transmit_ethernet,      out(4), frame],
268                 [in(7), g_arp,                  out(7), out_request_packet],
269                 [in(8), transmit_ethernet,      out(8), request_frame],
270                 [in(11),receive_ethernet,       out(11),reply_frame],
271                 [out(4),receive_ethernet,       in(4),  frame],
272                 [out(5),gate_way_ip,            in(5),  in_packet],
273                 [out(8),receive_ethernet,       in(8),  request_frame],
274                 [out(9),g_arp,                  in(9),  in_requset_packet],
275                 [out(11),transmit_ethernet,     in(11),  reply_frame]
276         ],
277         [],
278         [frame, request_frame, reply_frame, out_packet, in_packet,
279                 in_request_packet, out_request_packet, wire_snap],
280         []).
281
282 %/*================ 受信 E t h e r n e t ================*/
283 dw(receive_ethernet,component,0,
284         [       [r_arp,                 in_request_packet,      +],
285                 [gate_way_ethernet,     reply_frame,            +],
286                 [receive_ip,            in_packet,              +]
287         ],
288         [       [in(4), frame,                  _,      consume],
289                 [in(8), request_frame,          _,      consume],
290                 [in(10),out_reply_packet,       _,      consume],
291                 [_,     in_packet,              out(5), generate],
292                 [_,     in_request_packet,      out(9), generate]
293         ],
294         [       [in(4), gate_way_ethernet,      out(4), frame],
295                 [in(8), gate_way_ethernet,       out(8), request_frame],
296                 [in(10),r_arp,                  out(10),out_reply_packet],
297                 [out(5),receive_ip,             in(5),  in_packet],
298                 [out(9),r_arp,                  in(8),  in_requset_packet]
299         ],
300         [],
301         [frame, request_frame, reply_frame, in_packet,
302                 in_request_packet,  out_reply_packet,
303                 wire_snap],
304         []).
305
```

```
%/************************************************************
%      ======= 知識コンパイラ ver1.0 =======
%    ＰＷ(Physical World)：物理原理データベース
%
%                                                  oct. 8.1993.
%                                                  nov.25.1993.
%                    アプリケーションへ拡張したモデル構成の実装
%************************************************************/

%/*--------------------- ＰＷの書式 ----------------------

%pw(名称,物理式の一般表現,
%       [全パラメータのリスト],
%       [左辺のパラメータのリスト],
%       [右辺のパラメータのリスト],
%       [パラメータ間の関係を表すリスト*]).

%註）パラメータ間の関係は以下のように表す。
%       ａが増加でｂが増加の時：［ａ，＋，ｂ］
%       ａが増加でｂが減少の時：［ａ，－，ｂ］と表す。

%-------------------------------------------------------------*/

%---------------------- ＡＰＬ ---------------------------
pw(pw211,'stream=out_data+M',
        [stream, out_data],
        [stream],
        [out_data],
        [[stream,+,out_data]]).

pw(pw212,'stream=tcp_level',
        [stream, tcp_level],
        [stream],
        [tcp_level],
        [[stream,+,tcp_level]]).

pw(pw22,'in_data=[stream]+M',
        [in_data, stream],
        [in_data],
        [stream],
        [[in_data,+,stream]]).


%---------------------- ＴＣＰ ---------------------------
pw(pw311,'transport_packet=out_stream+M',
        [transport_packet, out_stream],
        [transport_packet],
        [out_stream],
        [[transport_packet,+,out_stream]]).

pw(pw312,'transport_packet=ip_level',
        [transport_packet,ip_level],
        [transport_packet],
        [ip_level],
        [[transport_packet,+,ip_level]]).

pw(pw32,'in_stream=[transport_packet]+M',
        [in_stream, transport_packet],
        [in_stream],
        [transport_packet],
        [[in_stream,+,transport_packet]]).


%---------------- ＩＰ 機能の記述 ------------------------
```

```
66 %*パケット ＝ アウト・Tパケット ＋ ＩＰ・アドレス
67
68 pw(pw101,'packet=out_t_packet+ip_address',
69         [packet,out_t_packet,ip_address],
70         [packet],
71         [out_t_packet,ip_address],
72         [[packet,+,out_t_packet],[packet,+,ip_address]]).
73
74 pw(pw102,'packet=host_up+lower_level',
75         [packet,host_up,lower_level],
76         [packet],
77         [lower_level,host_up],
78         [[packet,+,host_up],[packet,+,lower_level]]).
79
80 %*イン・Tパケット ＝ 受信 ＋ α
81 pw(pw2,'in_t_packet=packet+M',
82         [in_t_packet,packet],
83         [in_t_packet],
84         [packet],
85         [[in_t_packet,+,packet]]).
86
87 %---------------- 物理レイア 機能の記述 -------------------------
88
89 %* イン・パケット = フレーム + α
90 pw(pw3,'in_packet=[frame]+M',
91         [in_packet, frame],
92         [in_packet],
93         [frame],
94         [[in_packet,+,frame]]).
95
96 %* アウト・パケット = パケット + α   → 不必要か？ ----------------**
97 pw(pw4,'out_packet=[packet]+M',
98         [out_packet,packet],
99         [out_packet],
100        [packet],
101        [[out_packet,+,packet]]).
102
103 %*  フレーム ＝ 出力パケット ＋ 物理アドレス          ** t -> g **
104 pw(pw5,'frame=out_packet+ethernet_address',
105        [frame, out_packet, ethernet_address],
106        [frame],
107        [out_packet, ethernet_address],
108        [[frame,+,out_packet],[frame,+,ethernet_address]]).
109
110 %*  フレーム ＝ ケーブル                          ** g -> r **
111 pw(pw6,'frame=wire_snap',
112        [frame, wire_snap],
113        [frame],
114        [wire_snap],
115        [[frame,+,wire_snap]]).
116
117
118 %*  物理アドレス要求 ＝ ＩＰアドレス + α
119 pw(pw7,'request=[ip_address]+M',
120        [request, ip_address],
121        [request],
122        [ip_address],
123        [[request,+,ip_address]]).
124
125
126 %*  out_request_packet ＝ 物理アドレス要求 ＋ arp_table  ** transmit **
127 pw(pw8,'out_request_packet=request+arp_table',
128        [out_request_packet, request, arp_table],
129        [out_request_packet],
130        [request, arp_table],
```

```
131         [[out_request_packet,+,request],[out_request_packet,+,arp_table]]).
132
133 %*  out_request_packet = in_request_packet + arp_table  ** transmit **
134 pw(pw9,'out_request_packet=in_request_packet+arp_table',
135         [out_request_packet, in_request_packet, arp_table],
136         [out_request_packet],
137         [in_request_packet, arp_table],
138         [[out_request_packet,+,in_request_packet],
139          [out_request_packet,+,arp_table]]).
140
141 %*out_request_packet = 入力アドレス要求パケット+arp_table** gate_way **
142 pw(pw10,'out_request_packet=in_repuest_packet+arp_table',
143         [out_request_packet, in_repuest_packet, arp_table],
144         [out_request_packet],
145         [in_repuest_packet, arp_table],
146         [[out_request_packet,+,in_repuest_packet],
147          [out_request_packet,+,arp_table]]).
148
149 %*  入力アドレス要求パケット = ＡＲＰ送信フレーム + α
150 pw(pw11,'in_request_packet=[request_frame]+M',
151         [in_request_packet, request_frame],
152         [in_request_packet],
153         [request_frame],
154         [[in_request_packet,+,request_frame]]).
155
156
157 %*  出力応答パケット = in_request_packet + α  ** receive **
158 pw(pw12,'out_reply_packet=[in_request_packet]+M',
159         [out_reply_packet, in_request_packet],
160         [out_reply_packet],
161         [in_request_packet],
162         [[out_reply_packet,+,in_request_packet]]).
163
164 %*  出力応答パケット = 入力応答パケット + α  ** gate_way **
165 pw(pw13,'out_reply_packet=[in_reply_packet]+M',
166         [out_reply_packet, in_reply_packet],
167         [out_reply_packet],
168         [in_reply_packet],
169         [[out_reply_packet,+,in_reply_packet]]).
170
171 %*  入力アドレス要求パケット = ＡＲＰ送信フレーム + α
172 pw(pw14,'in_reply_packet=[reply_frame]+M',
173         [in_reply_packet, reply_frame],
174         [in_reply_packet],
175         [reply_frame],
176         [[in_reply_packet,+,reply_frame]]).
177
178 %*  物理アドレス = 入力応答パケット + α  ** transmit **
179 pw(pw15,'ethernet_address=[in_reply_packet]+M',
180         [ethernet_address, in_reply_packet],
181         [ethernet_address],
182         [in_reply_packet],
183         [[ethernet_address,+,in_reply_packet]]).
184
185
186 %*  ＡＲＰ送信フレーム = out_request_packet + ケーブル
187 pw(pw161,'request_frame=out_request_packe',
188         [request_frame, out_request_packet],
189         [request_frame],
190         [out_request_packet],
191         [[request_frame,+,out_request_packet]]).
192
193 pw(pw162,'request_frame=wire_snap',
194         [request_frame, wire_snap],
195         [request_frame],
```

```
196         [wire_snap],
197         [[request_frame,+,wire_snap]]).
198
199 %*  ＡＲＰ返信フレーム ＝ 入力応答パケット
200 pw(pw171,'reply_frame=out_reply_packet',
201         [reply_frame, out_reply_packet],
202         [reply_frame],
203         [out_reply_packet],
204         [[reply_frame,+,out_reply_packet]]).
205
206 %*  ＡＲＰ返信フレーム ＝  ケーブル
207 pw(pw172,'reply_frame=wire_snap',
208         [reply_frame, wire_snap],
209         [reply_frame],
210         [wire_snap],
211         [[reply_frame,+,wire_snap]]).
212
213
214
215
216
217
218
219
220
221
222
223
```

```
%/*************************************************************
%      ======= 知識コンパイラ ver1.0 =======
%   CW(Controle World)：制御知識データベース
%                                          oct 7 1993
%                  アプリケーションへ拡張したモデル構成の実装
%*************************************************************/

%/*--------------------- CWの書式 ----------------------

%CW1：パラメータの変動容易性

%       cw1(パラメータ名,変動容易性).

%       註）変動容易性＝（0：変動し易い、1：変動しにくい
%                        dependent：この場合パラメータの変動容易性は
%                                   部品情報に依存するので、
%                                   DWに記述されている。）

%CW2：部品の耐久性

%       註）CW2は部品情報としてDWに記述されています。

%-------------------------------------------------------*/

%--------------------------- APL ---------------------------
cw1(out_data,            1).          % 送信データ
cw1(in_data,             0).          % 受信データ
cw1(stream,              0).          % ストリーム

%--------------------------- TCP ---------------------------
cw1(out_stream,          0).          % 送信ストリーム
cw1(in_stream,           0).          % 受信ストリーム
cw1(transport_packet,    0).          % Tパケット

%--------------------------- IP ----------------------------
cw1(packet,              0).          % 送信パケット
cw1(out_t_packet,        0).          % 下位レベルへのパケット
cw1(in_t_packet,         0).          % 上位レベルからのパケット
cw1(ip_address,          0).          % IPアドレス

%--------------------------- 物理層 ------------------------
cw1(out_packet,          0).          % 下位レベルへのパケット
cw1(in_packet,           0).          % 上位レベルからのパケット
cw1(frame,               0).          % 物理レベルでの送信フレーム
cw1(request_frame,       0).          % ARP送信フレーム
cw1(reply_frame,         0).          % ARP返信フレーム
cw1(ethernet_address,    0).          % 物理アドレス

%--------------------------- ARP ---------------------------
cw1(request,             0).          % アドレス要求
cw1(out_request_packet,  0).          % 出力アドレス要求パケット
cw1(in_request_packet,   0).          % 入力アドレス要求パケット
cw1(reply,               0).          % 応答パケット
cw1(out_reply_packet,    0).          % 出力応答パケット
cw1(in_reply_packet,     0).          % 入力応答パケット
cw1(arp_table,           0).          % アープ・テーブル

cw1(host_up,             0).          % ホストが稼働状態か
cw1(wire_snap,           0).          % ケーブルの異常
cw1(tcp_level,           0).          % TCPレベルでの異常
cw1(ip_level,            0).          % IPレベルでの異常
cw1(lower_level,         0).          % 下位レベルでの異常
cw1(user,                0).          % ユーザ
```

```
%/*************************************************************
%       ======= 知識コンパイラ ver1.0 =======
%ＩＷ(Interpretation World)：解釈知識データベース
%
%                                                  oct 7 1993
%                  アプリケーションへ拡張したモデル構成の実装
%*************************************************************/

%/*---------------------- ＩＷの書式 -----------------------

%ＩＷ１：部品の役割による故障仮説の生成

%       註）これは部品依存型の知識なのでＤＷに記述されています。

%ＩＷ２：部品の分類による故障仮説の生成

%       iw2(分類名,パラメータ名,定性値(増減),故障仮説名).

%註）「分類名」には、流体(stuff)、導管(conduit)、部品(component)
%       の３種類があり、ＤＷに記述されている。


%---------------------------------------- ネットワーク知識

iw2(component,  arp_table,-,injury).     % ＡＲＰテーブル破損
iw2(component,  host_up,-,down).         % ホストダウン
iw2(component,  tcp_level,-,tcp_break). % ＴＣＰサービス異常
iw2(component,  ip_level,-,ip_break).   % ＩＰサービス異常
iw2(component,  lower_level,-,break).   % 物理サービス異常
iw2(component,  wire_snap,-,snap).       % ケーブル断線
```

```
%/*-------------------------------------------------------------
%                  知識コンパイラver2.0
%        ＦＭＷ（故障メカニズム用汎化知識）データベース
%                                              oct 7 1993
%                  アプリケーションへ拡張したモデル構成の実装
%*************************************************************/

%/*
%書式)   ｆｍｗ（［左辺の物質／状態，動作 ，部品名］，右辺の状態の集合）．

%例)
% ［一般形］      腐食の発生  <=  酸化の発生      微生物の存在      有機物の存在

% ［ｆｍｗ書式］ fmw([corrosion,generate,OnTheComponent],
%                        [oxidation,generate,OnTheComponent] or
%                        [microbe,exist,OnTheComponent] and
%                        [organism,exist,OnTheComponent]).
%*/

%/*----------------
% ［オペレータ定義］
%----------------*/

:- op(990 ,xfy,[or]).
:- op(980 ,xfy,[and]).


%/*------------------
% ［共通に使える知識］
%------------------*/

%/* （物質）の存在  <=  （物質）の発生      （物質）の移動 */
fmw([Substance,exist,OnTheComponent],
        [Substance,generate,OnTheComponent] or
        [Substance,move,OnTheComponent]).

%/*-----------------------------------------------
%fmw1 :  ［KC1->KC2のマッピングを行なう知識ベース］
%-----------------------------------------------*/
%/*
%書式) fmw1([KC2の状態],[KC1のノードの内容のリスト]).
%*/


%/* 下位レベルの異常 <= 受信ＴＣＰレベルからイン・ストリームが出力されない */
fmw1([tcp_break,generate,OnTheComponent],
        [[OnTheComponent,in_stream,-]]).

%/* 下位レベルの異常 <= 受信ＩＰレベルからインＴパケットが出力されない */
fmw1([ip_break,generate,OnTheComponent],
        [[OnTheComponent,in_t_packet,-]]).

%/* 下位レベルの異常 <= 受信物理レベルからイン・パケットが出力されない */
fmw1([break,generate,OnTheComponent],
        [[OnTheComponent,in_packet,-]]).
```