ネットワークのポリシー制御のための 2 つの部品化アーキテクチャの比較*

金田　泰

日立製作所 研究開発本部 IP ネットワーク研究センタ
〒244-0817 横浜市戸塚区吉田町 292 番地
E-mail: kanada@crl.hitachi.co.jp

あらまし ポリシーベース・ネットワークにおいては，高水準の機能あるいはポリシーを実現するためにしばしば 2 個以上の低水準のポリシーをくみあわせる必要がある．このような部品化されたポリシーをサポートするために，複数のポリシーをモデル化するための 2 つのアーキテクチャを開発した．それらはパイプ結合アーキテクチャとラベル結合アーキテクチャである．規則ベースの部品がポリシーベース・ネットワーク制御にとってすぐれていること，そしてラベル結合アーキテクチャが現在のところはよりよいことがわかった．しかし，また，ネットワーク環境においては非常に重要な並列性という点においてはパイプ結合アーキテクチャのほうがすぐれていることがわかった．

キーワード ポリシーベース・ネットワーク制御，ポリシーのくみあわせ，部品化アーキテクチャ，規則ベース言語，QoS ポリシー，Diffserv.

Two Rule-based Building-block Architectures for Policy-based Network Control*

Yasusi Kanada

Hitachi, Ltd., Research & Development Group, IP Network Research Center
Totsuka-ku Yoshida-cho 292, Yokohama, 244-0817, Japan
E-mail: kanada@crl.hitachi.co.jp

Abstract In a policy-based network, two or more policies must often cooperate to provide a high-level function or policy. To support such building-block policies, two architectures for modeling a set of policies have been developed: pipe-connection architecture and label-connection architecture. It is shown that *rule-based* building blocks are better for policy-based network control and that the label-connection architecture is currently better. However, the pipe-connection architecture is better in regards to parallelism, which is very important in network environments.

key words Policy-based network control, Policy combination, Building-block architecture, Rule-based language, QoS policy, Differentiated services (Diffserv).

* この報告の内容は金田 [Kan 00c] にもとづいている.

1. はじめに

ポリシーベース・ネットワークにおいては, しばしば 2 個以上のポリシーが協調してはたらく必要がある. たとえば, Diffserv (Differentiated Services) [Ber 99] においては, 同一のパケットに対してはたらく DSCP (Diffserv Code Point) [Nic 98] をマークするポリシーとキュー制御のポリシーとが協調しなければならない. なぜなら, 後者は前者がマークした DSCP にしたがって動作するからである. これは協調のもっとも単純なばあいであり, より複雑なばあいもある. ネットワークの高水準機能あるいはポリシーは低水準の機能あるいは部品化されたポリシーのくみあわせで実現される.

部品化された汎用のポリシー・システムを確立するため, 金田は規則ベースの部品化アーキテクチャをまず MIB のかたちで提案した [Kan 99][Kan 00a]. そして論理型言語にもとづくアーキテクチャを開発した [Kan 00b]. これらのアーキテクチャにおいては, ポリシーは規則ベースの部品, すなわち仮想フローラベル (virtual flow label) [Kan 99] か論理変数 [Kan 00b] によって結合された細粒度のポリシーによって構成される. 第 1 のアーキテクチャはドメインが QoS に限定され, 部品はつくりつけになっていたが, 第 2 のアーキテクチャは汎用であり, 部品は既存の部品から合成できる.

この報告では, これら 2 つのアーキテクチャにもとづく 2 つのあらたな部品化アーキテクチャをしめす. ひとつは前記の第 2 のアーキテクチャを洗練したパイプ結合アーキテクチャであり, もうひとつは前記の第 1 のアーキテクチャを一般化したラベル結合アーキテクチャである. 2 章においてはポリシーベース・ネットワーキングへの技術的な要求を分析し, 前記のようなアーキテクチャが必要な理由をしめす. 3 章においては 2 つの部品化アーキテクチャについてのべる. 4 章においては Diffserv のためのルータ設定についてのべる. そして, 2 つのアーキテクチャを 5 章において比較する.

2. なぜ規則ベースの部品をつかうのか?

ポリシーベース・ネットワークは, もともとネットワークとノードの設定の複雑さをへらすために開発された. ポリシーはベンダや機器に依存した設定コマンドをおきかえるものであり, IETF (Internet Engineering Task Force) のポリシー・フレームワーク WG においてまもなく標準化されようとしている. ポリシーは SLS (service-level specification) から生成される. SLS はネットワークのふるまいに関する仕様であり, ネットワークの運用者とユーザか複数の運用者間の契約である SLA (service-level agreement) にもとづいている.

ポリシーベース・ネットワークに関して 5 つの技術的な要求があるとかんがえられる. 第 1 の要求は SLS はそれが単純なものであれば人手でまたは機械的に容易にポリシーに変換できるべきだということである. SLS は自然言語や形式的な言語によって, 手続き的にではなく宣言的に記述される. もしポリシーが要求された機能を実現するための特定の手続きにつよく依存するなら, それは SLS から容易に生成することはできない. したがって, ポリシーは宣言的であるべきである. とくに, ポリシーは通常つぎのような if-then 規則 (条件-動作 型規則) によって記述される.

if (条件) 動作;

なぜなら, SLS は if-then 規則に容易に翻訳できると通常かんがえられているからである.

第 2 の要求は, ポリシーは実行できなければならないということである. ポリシーはネットワークやノードのふるまいをかえるから, ただのデータではなく, プログラムである. したがって, ポリシーがネットワーク・ノードに配布されたとき, それは実行できなければならない.

第 3 の要求は, ポリシーは動的に変更できなければならないということである. ネットワークは基本的に停止することがないので, ポリシーはその使用中に変更される必要がしばしば生じる. したがって, ポリシーはモジュラーでなければならない. もしポリシーが相互に依存のない規則によって構成されているなら, 他の規則に影響をあたえずに規則を追加したり, 変更したり, 削除したりすることができる.

第 4 の要求は, たとえ SLS が複雑であっても, SLS をポリシーに変換することができなければならないということである. 複雑な SLS は構造化された形式で表現されるべきである. したがって, その SLS から生成されるポリシーも構造化されているべきである. したがって, ポリシーを構造化するための手段があたえられなければならない. これは, ポリシーが部品 (components または building blocks) によって構成されるべきであるということを意味している.

第 5 の要求は, 最適化されたポリシーがもとのポリシーとおなじアーキテクチャにしたがって表現できるべきだということである. 素朴に表現されたポリシーは効率がわるいかもしれない. そのようなポリシーは自動的にまたは人手によって最適化されるべきである. もとのポリシーと最適化されたポリシーとは同一の言語によって表現されなければならない. そうでなければ, それらが意味的に同値のものだということをしめすことが困難だからである.

これらの要求をみたすためのひとつの方法は, ポリシーを規則ベースの部品化アーキテクチャ (building-block architecture) によって表現することである. 規則ベースの部品を使用した SLS から機器の設定までの可能な変換プロセスを**図 1** にしめす.

すなわち, 最初の 3 つの要求は Prolog ないし OPS5 [For 81] のような規則ベースのモデルないし言語を使用すればみたされる. ポリシーが複雑であれば, それを宣言的に定義し実行可能なプログラムに変換することはむずかしい. しかし, ポリシーの表現を適切に選択すれば, そのような変換はより容易になる. 人工知能の分野では, 1970 年代から 1980 年代にかけて知識表現がひろく, ふかく, 研究された. そして, 規則ベースのプログラミング言語, とくに

Prolog のような論理型プログラミング言語やプロダクション・システムにもとづいたエキスパート・システムを記述するための OPS5 のような言語が開発された．これらの言語は宣言的であると同時に実行可能である．我々はこれらの研究の成果をポリシーベース・ネットワーキングに適用することができる．これらの言語においては，規則は相互に依存しないかたちで記述される．すなわち，規則はその順序が変更されても特定の状況には特定の唯一の規則が適用されるように定義できる．

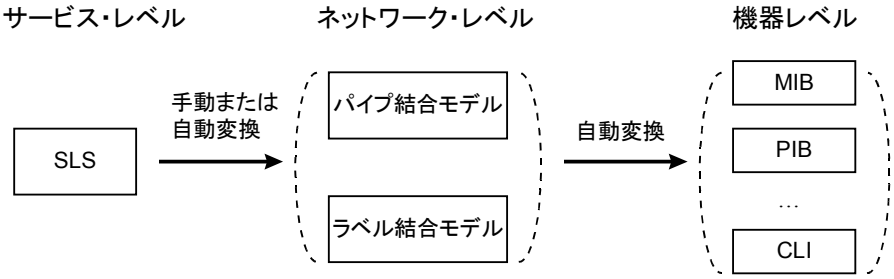


図 1 サービス・レベルから機器レベルまでのポリシー変換プロセス

第 4 と第 5 の要求は，部品化アーキテクチャによってみたすことができる．複雑なポリシーは部品をつかって表現することができる．部品のあつまりとして，よりおおきな部品を定義することによって，複雑さをへらすことができる．論理プログラミング言語を使用すれば，プリミティブな部品もくみあわせでつくられた部品も規則にもとづき，おなじセマンティクスにしたがう．ポリシーはプログラム変換によって最適化することができる．とくに局所的な最適化のばあいには，部品を他の部品によって交換することによって最適化できる．

3. 2 つの部品化アーキテクチャ

3.1 部品の構造

どちらのアーキテクチャにおいても，ポリシーあるいはポリシー規則は，複数の部品とそれらのあいだのつなぎとで構成される．部品は規則または規則の集合である．部品の構造は IETF のポリシー情報モデル [Moo 00][Sni 00] における構造とおよそひとしい．そして，規則の構造もそのモデルのポリシー規則の構造と似ている．ひとつの部品はつぎのように実行される．入力されたパケットが規則において指定された条件にマッチすると，その規則が選択される．そして，その規則において指定された動作が実行され，出力パケットが生成される．もしどの規則の条件も入力パケットにマッチしないときは，動作は起動されず，パケットは出力されない．したがって，部品はパケットのながれ，つまりフローを入力して，それをフィルタし，ばあいによっては複数のフローに分割したり複数のフローをひとつにあわせたりする．

ネットワーク・ノードは部品として，または部品のあつまりとしてモデル化される．部品は入力ポートと出力ポートとをもつ．複数の部品は入力ポートと出力ポートをつなぐことによって結合される．ネットワーク・ノードの機能は部品を頂点とするグラフによって表現され，ネットワーク全体も部品化モデルをつかってモデル化できる．ネットワーク・ドメインにおける各機能もグラフによって表現される．ポリシーサーバの仕事は，このグラフをサブグラフに分割して各サブグラフをドメイン内のノードに配布することである．サブグラフ間をむすぶ辺はノード間の (実 / 仮想) 回線にマップされる．

3.2 パイプ結合アーキテクチャ

図 2 を使用してパイプ結合アーキテクチャについて説明する．このアーキテクチャにおいては，各部品は固定数の入力ポートと出力ポートをもつ．各入出力ポートはポート識別子をもつ．ポート識別子は番号か英数字名であるが，ここでは順序番号であることを仮定する．図 2 は Diffserv ルータの設定例だが，これについては次章でさらに説明する．

部品は**パイプ**によって結合される．パイプの始点は部品の出力ポートに接続され，終点は他の部品の入力ポートに接続される．パケットは各パイプのなかをながれる．パケットが部品にながれこんで処理されると，1 個 (または 0 個) のパケットがその部品からながれだす．パイプは**タグ**によって識別される．パケットは入力ポートからながれこみ，出力ポートからながれだす．通常，パケットは出力ポートからただ 1 個出力される．つまり，パケットが暗黙に複写されることはない．また，ことなる入力ポートからながれこんだ 2 個以上のパケットが 1 個のパケットにマージされることはない．

図 2 においては 6 個の部品があたえられている．それらは分類器 (classifier)，計測器 (meter)，マーカ 1 (marker 1)，廃棄器 (dropper または discarder)，マーカ 2，スケジューラである．分類器と計測器はそれぞれ 2 個のよりちいさな部品 (規則) をふくんでいる．他の部品はそれ以上分割されていない．分類器と計測器は C1 という名称のパイプによって結合され，C1 は出力ポート 1 と入力ポート 1 とを結合している．計測器は 2 個の出力ポートをもっている．計測器に流

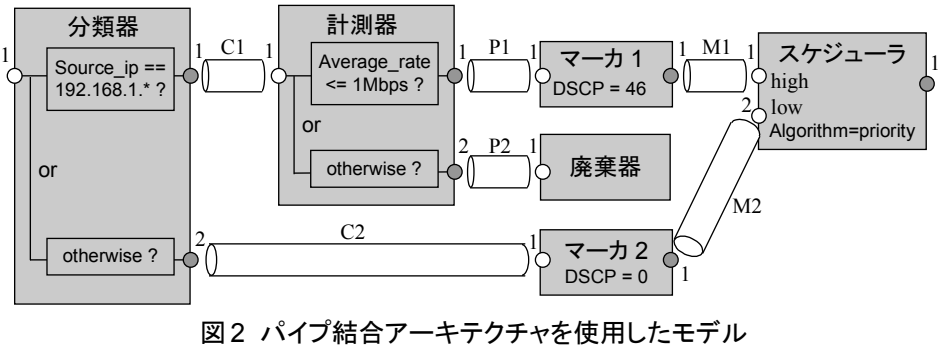


図 2 パイプ結合アーキテクチャを使用したモデル

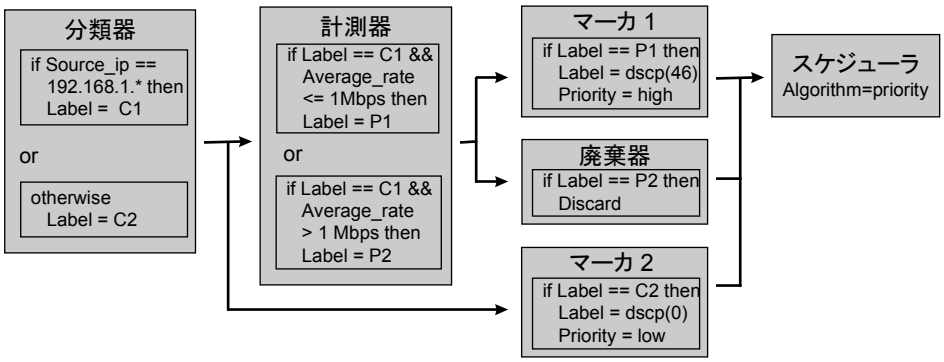


図 3　ラベル結合アーキテクチャを使用したモデル

入するパケットはこれらの出力ポートのどちらか一方から流出する．廃棄器は出力ポートがないので，ここに流入するパケットは出力されない．スケジューラは 2 つの入力ポートをもつが，他の部品は唯一の入力ポートをもっている．

このアーキテクチャは，GHC [Ued 85], Concurrent Prolog [Sha 86], Parlog [Cla 86] のような並列論理型言語によってうまく表現できる．並列論理型言語はデータ・ストリームの処理を記述するのに適しているので，パイプ結合モデルは直接的に表現できる．金田 [Kan 00b] はこのアーキテクチャのための SNAP (Structured Network programming by And-Parallel language) という言語を定義している．SNAP においては，各部品は述語によって表現され，述語は節 (すなわち規則) によって構成される．部品は論理変数によって結合される．すなわち，論理変数がパイプとしてつかわれる．図 2 のモデルは SNAP をつかうとつぎのように表現される．

```
ef_ingress(Si, So) :–
// 部品 ef_ingress はストリーム Si を入力してストリーム So を出力.
   or( filter[Source_IP = 192.168.1.*](Si, C1) |
          // (Si がふくむ) 始点 IP サブネットが 192.168.1.*
          // のパケットを C1 に出力.
       or( meter[Average_rate_max = 1Mbps](C1, P1) |
              // (C1 がふくむ) 契約帯域幅内のパケットを
              // P1 に出力.
              mark[DSCP = 46](P1, M1)
                  // P1 がふくむパケットをマークし M1 に出力.
        ;  otherwise(C1, P2) |
              // (C1 がふくむ) 他の条件 (ここでは唯一) に
              // あわないパケットを P2 に出力.
              discard(P2)  // P2 がふくむすべてのパケットを廃棄.
           )
    ;  otherwise(Si, C2) |
              // (Si がふくむ) 他の条件にあわないパケットを
              // C2 に出力.
           mark[DSCP = 0](C2, M2)
              // C2 がふくむパケットをマークして M2 に出力.
   ),
   schedule[Algorithm = priority](M1, M2, So).
      // ストリーム M1, M2 をマージして So とする. M1 にひとつ
      // のキューをわりあて, M2 にもうひとつのキューをわりあて
      // る. それらは優先度スケジューラによってスケジュールさ
      // れる. M1 のほうが優先度がたかい.
```

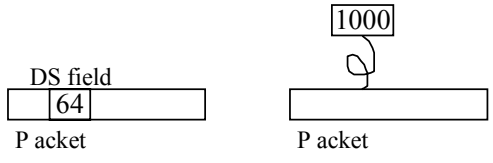
このプログラムは入力ポート，出力ポートを各 1 個もつ ef_ingress という部品を定義している．ここではこれ以上説明しないが，金田 [Kan 00b] は類似のプログラムを説明している．

3.3　ラベル結合アーキテクチャ

図 3 を使用してラベル結合アーキテクチャについて説明する．このアーキテクチャにおいては，各部品は 1 個の入力ポートと 1 個の出力ポートだけをもつ．部品は 1 個以上の規則をふくむ．たとえば，分類器，計測器の各部品はいずれも 2 個の規則をふくむ．部品はパイプのようなもので直接結合されることはないが，部品間で実行順序がきめられている．実行順序は有向グラフとして表現される．図 3 には 6 個の部品をふくむ有向グラフがあたえられている．ここで分類器は計測器およびマーカ 2 と結合されている．したがって計測器，マーカ 2 は分類の直後に実行される．計測器とマーカ 2 のうちのいずれが実行されるかは，それらの部品がふくむ規則の条件部とパケットの値に依存する．もしパケットが計測器の条件にマッチすれば，この部品が実行される．

各規則は**ラベル**とよばれる，整数値をふくむタグをパケット / フローにつける．ラベルには 2 種類ある (**図 4**)．一方はパケット内に格納される**実ラベル**であり，DSCP や MPLS ラベルはその例である．他方は仮想ラベルまたは VFL (virtual flow label) であり，図 3 において “Label” とよんでいるのがこれである．VFL の値はパケット内に格納されず，パケット外にタグとしてつける (図 4)．ひとつのフローまたはパケットにはただひとつの VFL をつけることができる．図 3 においては，分類器と計測器が VFL に値を代入し，マーカ 1 とマーカ 2 が DSCP に値を代入する．VFL の初期値は “未定義” (という特別の値) である．

フローあるいはパケットは 2 個以上のタグをもつこともありうる．図 3 においてはマーカ 1 とマーカ 2 において “Priority” という名称のタグをつけている．ラベル以外のタグを属性とよぶ．Priority 属性はスケジューラにおける優先度スケジューリングを指定するためにつかわれている．



(a) DSCP (**実ラベル**)　　(b) A VFL (**仮想ラベル**)

図 4　実ラベルと仮想ラベル

実行順序は VFL の値を定義・参照することによってきめられる．たとえば分類器の最初の規則は VFL に C1 という値を代入し，第 2 の規則は C2 という値を代入する．計測器の規則は VFL の値が C1 のときだけはたらくので，これらの規則は分類器の最初の規則が実行されたあとにだけ実行される．マーカ 2 の唯一の規則は VFL の値が C2 であること

を仮定しているので，この規則は分類器の第 2 の規則が実行されたあとだけに実行される．

ラベル結合アーキテクチャは，OPS5 のようなプロダクション・システム記述用言語を使用すればうまく表現できる．このような言語においては，各規則は if-then 規則として，つまり条件と動作とからなる規則として記述される．この規則構造はポリシー情報モデルにおけるポリシー規則の構造とよく似ている．しかし，OPS5 のような言語は規則を構造化する (部品によって構成する) 手段をもたないので，このアーキテクチャを表現するためにあらたな言語を定義する必要がある．[1] あらたな言語を使用すれば，図 3 のモデルはつぎのように表現できる．

```
MODULE ef_ingress IS
   RULE SET Classification, Metering, Marking1, Discarding,
         Marking2, Scheduling;
   RULE SET ORDER
      Classification –> Metering, Marking2;
      Metering –> Marking1, Discarding;
      Marking1, Discarding, Marking2 –> Scheduling;
   RULE SET Classification IS
      IF Source_ip == 192.168.1.* THEN
         Label = C1;
      OTHERWISE
         Label = C2;
      …
   RULE SET Scheduling IS
      IF true THEN   // このスケジューラがつねに使用される.
         Algorithm = priority;
END ef_ingress;
```

このプログラムは ef_ingress という部品を定義している．Ef_ingress は 6 個の規則をふくむ．実行順序は RULE SET ORDER 定義によってきめられる．RULE SET 定義は規則の集合を定義するが，その内容は図 3 にしめしたのとおなじなので，ここでは定義内容の大半を省略している．

4. 部品化モデルによる Diffserv

4.1 DiffServ のための部品

Diffserv のための 6 種類のプリミティブな部品を定義した．それらはフィルタ，計測器，マーカ，廃棄器，スケジューラ，マージャである．これらの部品の旧版は金田 [Kan 00b] が記述している．ここで定義する部品は Diffserv MIB [Bak 00], Diffserv PIB [Fin 00], QoS 情報モデル [Sni 00] などにおけるオブジェクトに似ている．しかし，この節で記述するモデルは規則ベースであり，部品は前記の情報モデルより細粒度なので，Diffserv MIB や PIB とはことなる．ほとんどの部品はそのままつかうこともできるし，Diffserv 以外のサービスのために拡張することもできるし，他の部品とあわせて使用することもできる．

規則はつぎのような構文にしたがう：

ruleTypeName[*parameters*].

フィルタ規則は MF や BA クラシファイアの一部である．フィルタ規則は IP パケット・ヘッダをテストする．つまり，DSCP, 始点と終点の IP アドレス，IP プロトコル，始点と終点の IP ポートなどのなかのひとつまたは複数個をテストする．これらの値はフィルタ規則のパラメタとして規定される．もしパケットが条件にマッチすれば，そのパケットは出力される．そうでなければそのパケットは出力されない．例をあげる．

```
filter[Source_IP = 192.168.1.*].   // MF 分類器の一部
filter[DSCP = 46].   // BA 分類器 (classifier) の一部.
```

パイプ結合モデルにおいては，フィルタ規則はひとつの入力ポートとひとつの出力ポートだけをもつ．入力ポートと出力ポートとをつなぐパイプ名を指定しなければならない．もし入力パイプ名が Si であり出力パイプ名が So であれば，規則はつぎのように記述される：

```
filter[Source_IP = 192.168.1.*](Si, So).
```

計測器規則は SLA における契約内容にあっているときだけトラフィックを通過させる．平均情報レートとバケット・サイズがパラメタとして指定できる．計測器規則はトークンバケット・メータや他の種類のメータをつかって実装される．例をあげる：

```
meter[Average_rate_max = 1Mbps].
```

パイプ結合モデルにおいては，計測器規則はひとつの入力ポートとひとつの出力ポートだけをもつ．

マーカ規則は入力パケットの DS フィールドに DSCP をかきこむ．入力された全パケットが出力される．マーカ規則の唯一のパラメタが DSCP である．例をあげる：

```
mark[DSCP = 46].
```

パイプ結合モデルにおいては，マーカ規則はひとつの入力ポートとひとつの出力ポートだけをもつ．

廃棄器規則は入力されたパケットを廃棄する．ラベル結合アーキテクチャにおいては 2 種類の廃棄器規則がある．それらは完全廃棄器規則とランダム廃棄器規則とである．ランダム廃棄器規則は一種の weighted random-early-discard (WRED) アルゴリズムにもとづいてパケットを廃棄する．パイプ結合モデルにおいてはランダム廃棄の機能はスケジューリング規則にふくまれている．したがって規則としては存在しない．完全廃棄器規則にはパラメタは存在しない：

```
absoluteDiscard.
```

パイプ結合モデルにおいては完全廃棄器規則は 1 個の入力ポートをもち，出力ポートをもたない．ランダム廃棄器規則の例をあげる：

```
randomDiscard[QMin = 10kB, QMax = 20kB, PMax = 0.1].
```

スケジューラ規則はスケジューリングしながら (パケットの順序をいれかえつつ) フローをマージする．スケジューラ規

[1] プロダクション・システムを使用したシステムの構造化手法としては黒板モデルが有名だが，ここで必要とされる構造はこれとはまったくことなっている．つまり，黒板モデルが共有記憶にもとづくモデルであるのに対して，ここでは基本的にはすべての情報がパケットに付随する (インスタンス変数に格納される) からである．

則のパラメタによってキューへのとりこみ，キューからのとりだしに関するスケジューリング法とパラメタとが指定される．また，スケジューラ規則はシェイピングも制御する．つまり最大と最小の出力レートが指定できる．例をあげる．

```
schedule. // 入力パケットは出力されるまで
          // キューイングされる. キュー・サイズは既定値どおり.
schedule[Algorithm = priority].
   // 入力パケットは優先度スケジューリング・アルゴリズム
   // によってスケジュールされる. 入力パケットは priority
   // 属性をもっていなければならない (図 3 参照).
```

パイプ結合モデルにおいては，キューを使用しないばあいにもフローのマージを明示的に記述する必要がある．バッファリングなしの 2 個以上のフローをマージするために**マージャ**規則が使用される :[1]

```
merge(Si1, Si2, Si3, So).
 // パイプ Si1, Si2, Si3 から入力されるパケットは So に出力される.
```

マージャ規則は Diffserv だけのものではなく，パイプ結合アーキテクチャにおいてはつねに必要である．それとちがって，ラベル結合アーキテクチャにおいてはフローは暗黙にマージすることができ，マージのための規則は必要とされない．マージャ規則に入力されるフローに関してはスケジューリングはおこなわれない．なぜなら，バッファリングなしにマージされるからである．もしバッファリングが必要なら，スケジューリング規則を使用する必要がある．

ラベル結合モデルにおいては実行順序を指定する．Diffserv における可能な順序を**図 5** にしめす．計測規則は反復することができる．なぜなら，フローは複数の計測条件にもとづいて分割でき，各サブ・フローがことなるあつかいをうけられるからである．階層的なスケジューリングやシェイピングのためにスケジューリング規則は反復することができる．

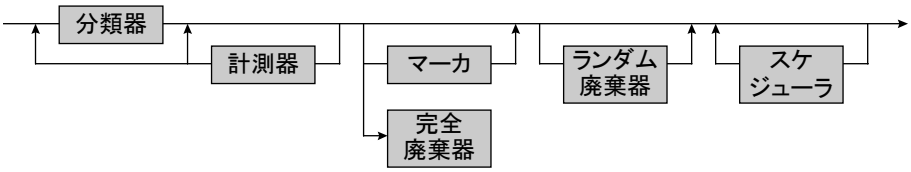


図 5 ラベル結合アーキテクチャにおける部品の実行順序

4.2 EF サービスの設定

Expedited forwarding (EF) サービス [Jac 99] は仮想専用線サービスである．各フローはネットワークの入口エッジ・インタフェース (ドメイン入口のルータ・インタフェース) において帯域幅を制限 (police) され集成 (aggregate) される．この DSCP をもつパケットはたかい優先度でコア・インタフェースつまり出入口エッジ以外のインタフェースに転送される．

EF サービスのための SLS の例をあげる．

```
IF フローはユーザ A からきた (始点 IP アドレスが 192.168.1.*)
THEN
   IF 情報レートが 1Mbps 以内 THEN
      そのフローを EF トラフィックとしてあつかう;
   OTHERWISE
      そのフローのパケットを全廃棄する;
OHERWISE
   そのフローをベスト・エフォート・トラフィックとしてあつかう;
```

[1] 単一代入の制約があるため，マージング規則はこのアーキテクチャに必須である．

このサービスのための設定は図 2 のパイプ結合モデルか図 3 のラベル結合モデルによって表現される．分類器，計測器，マーカ，廃棄器を各入口エッジ・インタフェースに配布し，スケジューラを各コア・インタフェースに配布する．[2]

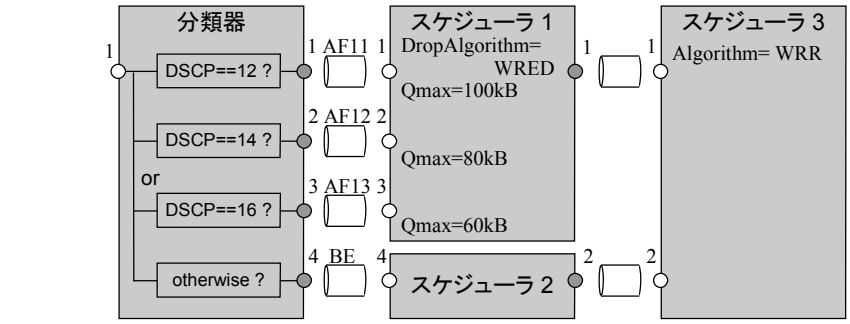
4.3 AF サービスの設定

Assured forwarding (AF) サービス [Hei 99] はさまざまなサービスのために使用できる．いわゆるオリンピック・サービスはそのひとつである．オリンピック・サービスにおいては金，銀，銅の各サービス・クラスがあるが，これらを AF クラスに対応させる．金クラスは最優先だが，銀クラスがそれにつづく．銅クラスの優先度は 3 番めだがベスト・エフォート・クラスよりは優先される．各 AF クラスには 3 つまでのサブクラス (AFn1, AFn2, AFn3) が存在しうる [Hei 99]．ネットワーク・ノードにおいて各サブクラスには同一のキューをわりあてるが，キューのふかさをかえるか，または WRED のパラメタをかえる．

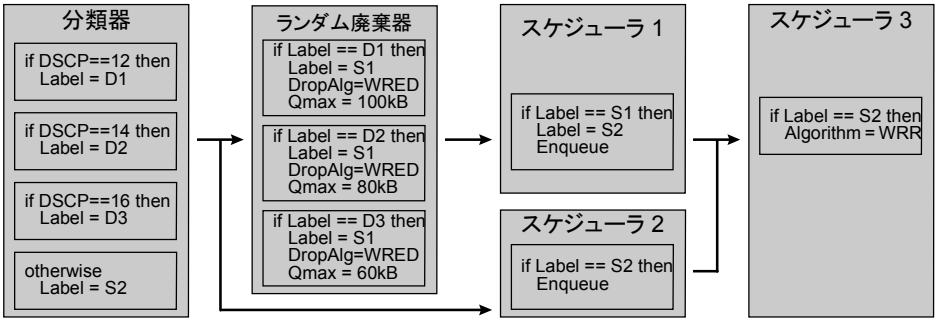
AF サービスのためのコア・インタフェースの設定例を**図 6** にしめす．ここには AF1 の設定だけをしめす．入力されたフローは BA 分類器によって 4 個 (AF11, AF12, AF13 とそれ以外) にわけられる．AF11 から AF13 のフローは 3 つのことなる廃棄閾値をもつ 1 個のキューにマージされる．AF11 のパケットつまり “schedule” の最初の引数は，キューが満杯になったときだけ，つまり 100 kB のデータでみたされたときだけ廃棄される．AF12 のパケットつまり第 2 引数は，キューに 80 kB のデータがふくまれるときに廃棄される．さらに，AF13 のパケットつまり第 3 引数は，キューに 60 kB のデータがふくまれるときに廃棄される．AF のフローにふくまれるパケットと他のフロー (ベスト・エフォート) にふくまれるパケットはおもみつきラウンドロビン (WRR) スケジューラによってマージされる．

スケジューラ 1 とスケジューラ 2 はキューを表現していて，パケットをキューにいれる．スケジューラ 3 は実際のスケジューラに対応していて，指定されたスケジューリング・アルゴリズムにしたがってスケジューラ 1, 2 (キュー) からパケットをひきぬく．ラベル結合モデルにおけるスケジューラ 1, 2 はラベルを置換する以外の動作をふくんでいない．しかし，それらはキューを表現するために必要なので記述している．

[2] 図 2 においては 1 個のインタフェース内でマーキングとスケジューリングをおこなうことを前提としているが，金田 [Kan 00c] はこれをエッジ・インタフェースとコア・インタフェースとにわけて設定するばあいのモデルも記述している．



(a) パイプ結合モデル



(b) ラベル結合モデル

図6 AF サービスのためのコア・インタフェース設定

5. 比較

2 個の部品化アーキテクチャの 5 つの主要な相違点を説明する.

1. **規則の構造**: ラベル結合アーキテクチャにおける規則は if-then 規則だが, パイプ結合アーキテクチャにおける規則は if-then 規則とはかぎらない. 後者において if-then 規則をシミュレートすることはできるが, 規則の構文は前者とはことなっている. つまり, 規則は条件と動作によってではなく, 一様な部品によって構成されている. SNAP の部品はガード (“|” 以前の部分) と本体とで構成されていて, ガードは条件にちかく, 本体は動作にちかい. しかし, それらのセマンティクスはことなっている. したがって, パイプ結合アーキテクチャをつかうなら, ポリシーの開発を容易にする方法を開発する必要があるだろう.

2. **制御のながれの指定**: パイプ結合アーキテクチャにおいては制御のながれはパイプによって規定されるデータのながれからきまるから, 明示的な制御のながれの指定は必要ない. しかし, ラベル結合アーキテクチャにおいては制御のながれはパケット上のタグだけでは一意に指定されないから, 制御のながれの指定が必要である. したがって, ラベル結合アーキテクチャにおいてはポリシーの実行順序を明示しなければならない.

3. **複数の入出力ポートとモジュラリティ**: パイプ結合アーキテクチャにおいては複数の入出力ポートをもつ部品が必要とされるが, ラベル結合アーキテクチャでは入出力ポートはそれぞれ 1 個だけでよい. 前者においては, もしことなるやくわりをもつ 2 個以上の入出力が必要なときには, それらはポートによってくべつする必要がある. それに対して後者においてはことなるやくわりはポートによってではなくフローラベルによってくべつされる. この相違点がスケジューラやマージャのモジュラリティに関するつぎの 2 つのちがいをもたらしている.

第 1 に, (Diffserv のための) スケジューラは通常, 2 個以上の規則からパケットを入力する. パイプ結合アーキテクチャにおいてはその規則の各出力ポートはスケジューラの入力ポートにパイプをつかって結合する必要がある. したがって, 規則が増減するたびに入力ポートの数も増減する必要がある. しかし, ラベル結合アーキテクチャにおいては入力ポートの数はつねに 1 でよいので, スケジューラ規則を変更する必要はない.

第 2 に, マージャはパイプ結合アーキテクチャだけでつかわれるが, 規則が増減するたびにやはりその入力ポート数を増減する必要がある. 一方, ラベル結合アーキテクチャにおいてはフローはマージャをつかわず暗黙にマージされるので, このような必要はない.

4. **タグの用法**: 2 つのアーキテクチャでタグの用法に 2 つの相違がある. 第 1 の相違は, パイプ結合アーキテクチャにおいては各パイプが一意のタグをもつ必要があるのに対して, ラベル結合アーキテクチャにおいては同一のラベルを複数回使用できることである.[1] このため後者においては DSCP をフローラベルとして使用できる. つまり, いったんマークしたあとは通常 DSCP はドメイン内では変更されないが, 部品ごとにことなる処理を指定するタグとして使用できる. MPLS のラベルなどについてもそれは同様である.

第 2 の相違は, ラベル結合アーキテクチャにおいては複数のタグをつかえるが, パイプ結合アーキテクチャにおいてはパイプにタグを 1 個だけしかつけられないことである. もしことなるフローにはことなるパラメタを適用したければ, フローはことなる入力ポートに入力する必要がある. この条件は Diffserv の例においてはスケジューラにおける相違となってあらわれている. パイプ結合アーキテクチャにおいては, WRED のパラメタ (たとえば QMax =

[1] もしフローラベルが一意であれば, すべての規則は 1 個の規則集合にいれることができ, 制御フローを指定する必要はなくなる. そうすると, ラベル結合アーキテクチャはパイプ結合アーキテクチャに非常にちかいものになる.

100 kB) またはスケジューリング優先度 (たとえば図 3 の Priority = high) などというスケジューラへのパラメタは，(キューイング規則ではなく) スケジューラ規則のなかで定義する必要がある．これに対して，ラベル結合アーキテクチャにおいては，そのようなパラメタはそれぞれことなる属性としてあたえる．したがって，WRED のパラメタはスケジューラにうめこむことなしに，それから独立したランダム廃棄器規則において指定できる (図 6 (b) 参照)．したがって，部品は細粒度になり，部品の設計がより柔軟になっている．

5. **並列性**: ネットワーク環境においては並列性に関する制約は最低限にするべきである． パイプ結合アーキテクチャにおいてはパケットの順序がスケジューラ以前の部分において保持されるなら，各部品の実行を並列にできる．しかし，ラベル結合アーキテクチャは基本的に逐次実行のモデルなので，並列化には 2 つの障害が存在する． 第 1 はラベルと属性への代入は逆順にしてはならないことである． 第 2 は規則集合の実行順序が指定されているため，それが並列実行を制約することである． (この制約は第 2 の相違点から生じている.) パイプ結合アーキテクチャのための言語 SNAP は，並列処理用プログラムを記述するための並列論理型言語にもとづいているので並列実行を制約するものはすくない．

相違点 1, 3, 4 においてはラベル結合アーキテクチャのほうがすぐれている． 現状では従来アーキテクチャのポリシーベース・システムを汎用化するには，ラベル結合アーキテクチャに移行するのが容易であり有利である．しかし，相違点 5 においてはパイプ結合アーキテクチャのほうがすぐれている． 従来アーキテクチャからパイプ結合アーキテクチャに移行するのはかならずしも容易ではないが，著者はそれには理由があるとかんがえる． つまり，並列性は必要であり，並列実行のセマンティクスは明確でなければならない．

6. 結論

パイプ結合アーキテクチャとラベル結合アーキテクチャという，ポリシーをモデル化するための 2 つの規則ベースの部品化アーキテクチャを開発した． いますぐ使用できる解はラベル結合アーキテクチャだけだが，パイプ結合アーキテクチャのほうが並列性という非常に重要な点においてすぐれていることがわかった． また，後者のほうがセマンティクスも明確だとかんがえられる．したがって，今後の研究によってその欠点をなくすことができれば，後者のほうがよりよい解になるとかんがえられる．

謝辞

日立アメリカの 吉澤 聡 氏からこの論文に関するふかい洞察にもとづいたコメントをいただいたので感謝する．

参考文献

[Bak 00] Baker, Chan, F., K., and A. Smith: Management Information Base for the Differentiated Services Architecture, draft-ietf-diffserv-mib-04.txt, *Internet Draft*, July 2000.

[Ber 99] Bernet, Y., et al.: A Framework for Differentiated Services, draft-ietf-diffserv-framework-02.txt, *Internet Draft*, February 1999.

[Cla 86] Clark, K., and Gregory, S.: PARLOG: Parallel Programming in Logic, *ACM Trans. on Programming Languages and Systems*, Vol. 8, No. 1, pp. 1–49, 1986.

[Fin 00] Fine, M., McCloghrie, K., Seligson, J., Chan, K., Hahn, S., Smith, A., and Reichmeyer, F.: Differentiated Services Quality of Service Policy Information Base, draft-ietf-diffserv-pib-01.txt, *Internet Draft*, July 2000.

[For 81] Forgy, C. L.: *OPS5 User's Manua*l, Technical Report CMU-CS-81-135, Carnegie Mellon University, Dept. of Computer Science, 1981.

[Hei 99] Heinanen, J., Baker, F., Weiss, W., and Wroclawski, J.: Assured Forwarding PHB Group, RFC 2597, June 1999.

[Jac 99] Jacobson, V., Nichols, K., and Poduri, K.: An Expedited Forwarding PHB, RFC 2598, June 1999.

[Kan 99] Kanada, Y., Ikezawa, M., Miyake, S., and Atarashi, Y.: SNMP-based QoS Programming Interface MIB for Routers, draft-kanada-diffserv-qospifmib-00.txt, *Internet Draft*, October 1999, http://www.kanadas.com/-activenet/draft-kanada-diffserv-qospifmib-00.txt.

[Kan 00a] 金田 泰: Rule-based Modular Representation of QoS Policies, ネットワーキング・アーキテクチャ・ワークショップ 10 周年記念大会, pp. 106-113, 電子情報通信学会, 2000.

[Kan 00b] Kanada, Y.: A Representation of Network Node QoS Control Policies Using Rule-based Building Blocks, *International Workshop on Quality of Service 2000* (*IWQoS 2000*), pp. 161–163.

[Kan 00c] Kanada, Y.: Two Rule-based Building-block Architectures for Policy-based Network Control, *2nd International Working Conference on Active Networks* (*IWAN 2000*), Lecture Notes in Computer Science, Springer, October 2000.

[Moo 00] Moore, B., Ellesson, E., Strassner, J., and Westerinen, A.: Policy Framework Core Information Model — Version 1 Specification, draft-ietf-policy-core-info-model-07.txt, *Internet Draft*, July 2000.

[Nic 98] Nichols, N., Blake, S., Baker, F., and Black, D.: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474, December 1998.

[Sha 86] Shapiro, E.: Concurrent Prolog: A Progress Report, *IEEE Computer*, August 1986, pp. 44–59, 1986.

[Sni 99] Snir, Y., Ramberg, Y., Strassner, J., and Cohen, R.: Policy Framework QoS Information Model, draft-ietf-policy-qos-info-model-01.txt, *Internet Draft*, April 2000.

[Ued 85] Ueda, K.: Guarded Horn Clauses, *Logic Programming Conference '85*, pp. 225–236, 1985. Also in *ICOT Technical Report*, TR-103, Institute for New Generation Computer Technology, 1985, and in *New Generation Computing*, Vol. 5, pp. 29–44, 1987.