リフレクティブエージェントコンピューティングによるネットワーク制御方式

中沢 実　　阿部 倫之　　服部 進実
金沢工業大学 人間・情報・経営系
E-mail: {nakazawa,abe,hattori}@infor.kanazawa-it.ac.jp

インターネットの急拡大は，インターオペラビリティ・スケーラビリティを重視した広域分散型ネットワークが今後の主流になることを示している。また，インターネットはローカルスキーマの積み上げによるボトムアップ的なアプローチをとっており，非決定性・非形式性を含む動的なネットワーク環境である。そこで本稿では，環境変動に対応するために，リアクティブな対応のみならず，環境変動の予測学習機能を含めたデリベラティブな対応を実現するメカニズムを有しているリフレクティブエージェントによるプラットフォームの提案を行なう。これによって，環境変動に対して有効な動作記述を適応的に導出することができ，マルチメディアサービスに柔軟に対応可能にするものである。

1　まえがき

近年のインターネットの急拡大に代表されるように，インターオペラビリティ(相互接続・運用性)やスケーラビリティ(規模拡張性)を重視したよりフレキシブルな広域分散型ネットワークが今後の主流になるものと思われる。特にTelescript[8]などに代表されるモバイルエージェントシステムでは，インターネット・無線網・公衆網などのあらゆるネットワーク上を，ユーザの要求記述に基づき，さまざまなネットワーク上に存在しているサービス資源上に移動し，サービス資源のメソッド実行が行なえるなどの，ネゴシエーション・情報収集を行なうメカニズムが有望視され 多くの研究・開発が発表されている。一方，INAやTINAに見られるように，分散オブジェクト指向に基づくネットワークのモデル化に関する研究も活発化しており，コンピュータ上のサービスアプリケーションが相互に通信し合えるネットワークアーキテクチャの研究が進められている。[9]

本稿では，インターネットに代表されるようなローカルスキーマの積み上げによるボトムアップ的なアプローチ手法の構築が主流になると思われる開放型情報ネットワークにおいて，その動的な環境変動に対応できるネットワーク制御を実現するために，リフレクティブエージェント機能を持つプラットフォームの提案を行なうことを目的としている。

本プラットフォームは，環境変動に対しリアクティブな対応のみならず，環境変動の予測学習を含めたデリベラティブな対応を実現するメカニズムを有しており，これによって，環境変動に対して有効な行為を決定する動作記述を適応的に導出することができる。[2]

2章では，開放型情報ネットワークの要件を示し，3章では，我々の提案するリフレクティブエージェントコンピューティングのアーキテクチャを示し，これらを実現するための分散ルール協調によるマルチエージェントシステムの機能と学習機能さらには，プラットフォームが環境に適応する方法について述べる。4章においては，開放型情報ネットワークの動的環境に対する本プラットフォームの柔軟性を利用したネットワーク管理方式の例を示す。

2　開放型情報ネットワークの要件

開放型情報ネットワークにおいては，インターオペラビリティ・スケーラビリティを重視した広域分散型ネットワークが今後の主流になるものと思われる。これは従来の電話網に見られるように，トップダウンアプローチによる計画性，形式性，保守運用性を重視したグローバルスキーマに基づくネットワーク研究・開発アプローチとは異なり，現在のインターネットのようなローカルスキーマの積み上げによるボトムアップアプローチで行なわれ，それらは，非決定性・非形式性を含む広域分散型ネットワークである。

一方，ユーザからネットワークに対し，システムとサービスの構築・運用，定義・実行する仕組みとな

Network Control Mechanism Based on the Reflective Agent Computing, Minoru NAKAZAWA, Noriyuki ABE, Shimmi HATTORI, Kanazawa Institut of technology

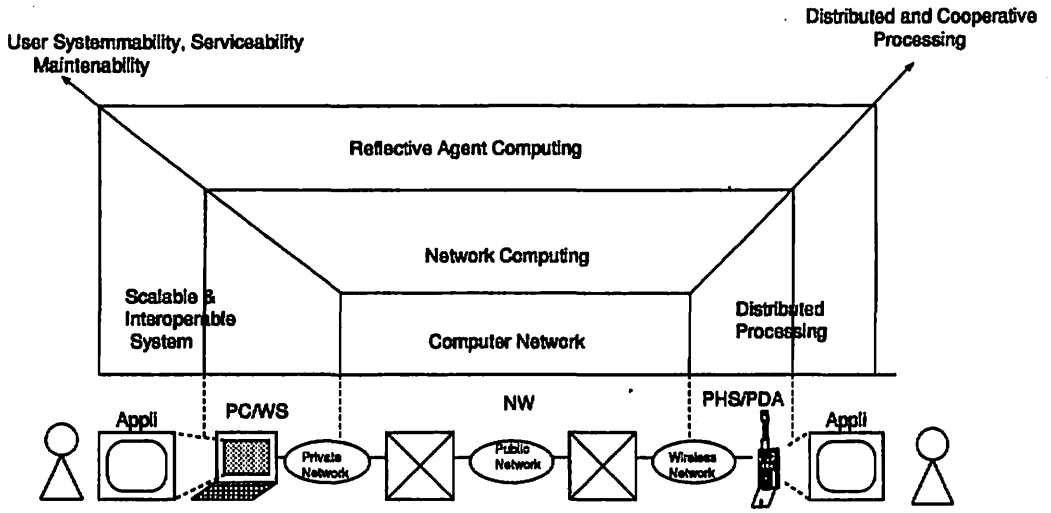

図 1: リフレクティブエージェントコンピューティングの位置付け

るユーザシステマビリティとユーザプログラマビリティが開放型情報ネットワークに対して必要になると考えられる。[1]

Telescript では, 開放型情報ネットワークのようなさまざまなネットワークにおいてもユーザの要求記述をネットワーク上に直接, エージェントをスクリプトの形式で転送することで, メソッド呼び出し単位での相互通信を可能としている。しかしながら, エージェントの振舞いを記述した動作シナリオは, 動作前にすべて記述する必要があるため, 広域かつ動的に変化すると考えられるようなネットワークに対しスムーズに対応するのは困難といえる。

また, 現在のインターネットでは, マルチメディアトラヒック, マルチコネクションを要求するマルチメディアサービスに対し, ネットワークリソースを適応的に配置する仕組みやリソース予約機構などについても, アプリケーション側の要求仕様と, ネットワーク側の制御機能のマッピングに関しては, 実際に適用してみない限り分からないのが現状と思われる。

このため, これらのネットワーク制御機能をひとつの動作シナリオとして実行しながらも, 環境変動に追従しながら適応的に動作シナリオの改変がリアクティブに行なわれる機能を実現することが必要と考えられる。それとともに, そのような動作シナリオの実行結果に基づいて, デリベラティブに実行内容を確認し, 動作シナリオの導出方法の改変 (学習) を行なうことで, リアクティブに改変された動作シナリオの行動の結果をフィードバックし, リアクティブに実行された結果に対して追補させるメカニズムを実現することも必要と考えられる。その結果, ユーザにとっては, 分散環境などの動的な環境変動を意識することなく, ユーザ自らの表現・定義を用いて, 情報通信サービスを要求し, 実行することができるユーザ主導型のネットワークシステムとサービスの実現が可能になると思われる。[1, 4]

そこで, このような機能を実現するマルチエージェントシステムでは, リアクティブに行なわれる自己の行為の適用結果によってリフレクティブに行為 (知識) の洗練や導出を行なう経験強化型の学習メカニズムが必要である。また, エージェントの動作シナリオは, 理解性や記述性の点から, 分類子システムのようなビット列ルールやニューラルネットワークのような非記号処理ベースではなく, プロダクションルールやスクリプト言語のような記号処理ベースで記述できることが重要である。

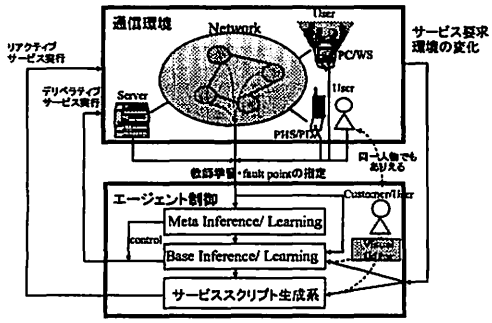

図 2: 通信環境とエージェント制御

3 リフレクティブエージェントシステム

開放型情報ネットワークの基本的枠組みとして, リフレクティブエージェントコンピューティングを提案する。(図 1)

現在のネットワークコンピューティングがユーザサイドの PC や WS 上の OS をネットワーク OS として包含した分散処理体系であり, インターオペラビリティ・スケーラビリティを実現するといった, いわば端末技術の向上による成果であり, 今後もこの傾向は変わらないものとも考えられる。これにリフレクティブエージェントコンピューティングの世界では図 1の如く, ユーザ主導型ネットワークコンピューティングの考え方が基本となっており, ミドルウェアプラットフォームやその上の付加価値アプリケーションやさらにはユーザ自身を織り込んだネットワーク体系である。

3.1 リフレクティブタワー

自己反映計算可能なメカニズムを有するシステムをリフレクティブシステム (Reflective System) と呼ぶ。[7]

End-to-End のインターオペラビリティや QoS 制御などが要求される開放型情報ネットワーク環境や, さらに構造そのものが動的に変化するモバイルコンピューティング環境下では, それに対する適応性の実現が必須であるが, すべての可能なインタラクションをあらかじめ予測することは不可能である。提案するリフレクティブエージェントコンピューティングのプラットフォームでは, このような動的な環境変動に対し, (1) 分散ルール協調と分散強化学習に基づくリアクティブなリフレクションと, (2) その強化学習方式や推論方式の学習を行なうメタ推論に基づくデリベラティブなリフレクションにより構成している。そして, ユーザ自身がネットワークサービスを定義し実行し得ること (ユーザサービサビリティ) や, ユーザ自身がネットワークを構築し得ること (ユーザシステマビリティ), さらにユーザ自身が構築したネットワークをローカルに管理運用が可能になること (ユーザメインテナビリティ) を目指した, リフレクティブエージェント制御による通信環境のイメージを図 2に示している。

このように, 動的な通信環境に対しサービス実行を行なうためには, ユーザのサービス要求を忠実に実行するだけでなく, 通信環境の変化や, ユーザのサービス要求の結果を使用してのサービスの再構築が適応制御的に行なえるリフレクティブなシステムが有効であると考える。

これを実現するためのメカニズムとして, 図 3に示すような分散強化学習機能に基づくリフレクティブタワーを考案した。この図のように, リフレクティブシステムを構成するために, 本来の目的となる問題対象のシステム (**Base System**) と, その計算／推論／記述方式を制御あるいはカスタマイズするシステム (**Meta System**), そしてその間のインタラクションの記述を同一の記述とすることで, 動的な環境変動に適応できるソフトウェアを自然な形で記述することができる。

さらに本システムでは, 以下に述べるように, 複数の学習方式を用いて, 環境変動に合わせて, **Base System, Meta System** を改変するメカニズムを実現しているため, ユーザの記述としては環境変動を意識する必要がなくなる。

分散ルール協調と分散ルール強化学習に基づくリフレクション

ルールベースによって記述されるエージェントの振舞いは, Profit Sharing 法・Bucket Brigate 法・この二つを組み合わせた Hybrid 法 [2] のいづれかの分散ルール強化学習によるベースルールの淘汰状況に

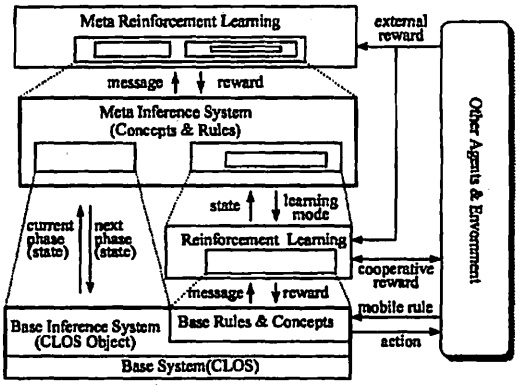

図 3: 分散強化学習型リフレクティブタワー

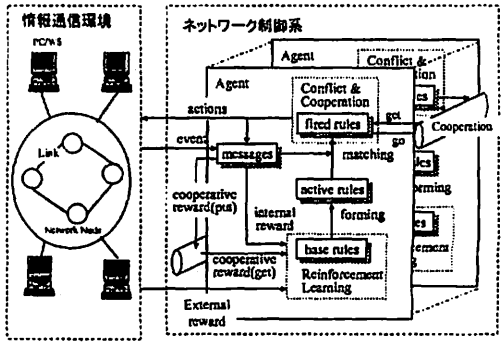

図 4: 分散ルール協調によるマルチエージェントシステム

依存する。リフレクティブな動作は, その時の環境状況に反応できるルールの即時的な実行を行なうことで対応する。それとともに分散ルール強化学習を制御している学習ルールは, ルールベースと同様のプロダクションルールによって記述されており, ルールベースの動作結果に基づいて強化学習が実施される。これにより, 競合解消に勝利するルールが変化し, さらに活性化するルールが変化して, 動作モードのリフレクションが進行する。

また, 分散ルール協調では, 図 4に示すように推論時にはエージェント間において発火ルール (fired rules) の転送 (以後, 転送されるルールをモバイルルールと呼ぶ) が行なわれる。

各エージェントでは, このようなモバイルルールを取り込み, 競合解消フェーズを経て実行することにより, 動作モードのリフレクションが実現される。モーバイルルールが実行される可能性は, エージェント内の競合ルールのその時点におけるルール実行の有効性に依存している。この有効性は, ルール強化学習によって得られた そのルールの過去の貢献度で定まる。

メタ推論に基づくリフレクション

推論フェーズと強化学習モードを制御するために, メタ推論によるリフレクションを導入する。これは分散ルール協調と分散ルール強化学習を含むベース推論過程を, 次節の 3.2 において説明する 6 つの推論フェーズに分割し, 各推論フェーズの実行がメタ推論によって決定されるものである。さらに, 分散ルール強化学習における学習モード (profit sharing 法, backet brigate 法, hybrid 法) をメタ推論で決定可能にする。従って, マルチエージェントシステムの場合, 各エージェントは異なる学習モードで動作する可能性があり, このとき モバイルルールは, それを取り込んだ受信側エージェントの学習モードに依存する。このメタ推論に基づくリフレクションでは, 各フェーズの推論工程をメタルールで記述し, メタ作業記憶の内容に基づいて動作するフェーズを特定する。これにより, エージェントの推論方式を動的に変更し, 動作モードを間接的に改変するメカニズムを実現する。

3.2 システム構成

本システムは, ベース推論エンジン (Base inference engine), メタ推論エンジン (Meta inference engine), 分散学習エンジン (Distributed learning engine), 分散競合解消器 (Distributed conflict arbiter), 効果器 (effector) から成る。このシステム構成を図 5に示す。

ここで, 各推論フェーズを次に示す。

(P1) 作業記憶内のメッセージに関係するコンセプトの活性度を増加する。

(P2) 閾値以上の活性度を持つコンセプトを活性化コンセプトとして選別し, そこから, 活性化ルール群を形成する。

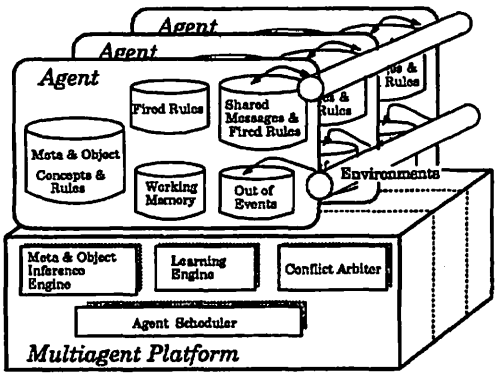

図 5: 分散 PI モジュールによるマルチエージェントシステム

(P3) 作業記憶内のメッセージと活性化ルールをマッチングし, 発火ルールを生成する。

(P4) 発火ルールの行為部が相互排他行為の場合, 他のエージェントからモバイルルールを取り込んで, 発火ルールに追加し, ルールに付加されている付け値 (そのルールの有効性を表す値) を用いて競合解消する。

(P5) 競合解消に勝利した発火ルールとモバイルルールの行為部を実行する。この時, bucket brigate 法では, 発火したルールから, そのルールを発火させたルールへ報酬 (internal reward) を, またモバイルルールを取り込んだ受信側エージェントから, それを送った送信側エージェントへ報酬 (cooperative reward) を伝搬してルールの強度を修正することで分散ルール強化学習を実施する。また, profit sharing 法では, ルールの実行履歴を用いて, 目標達成に貢献したルールに対して報酬値を均等配分するために, ルールの実行履歴を登録しておく。

(P6) ルール行為部の実行によって目標に到達した場合, そのルールに対して報酬 (external reward) を与え強度を修正する。その報酬値は, 適用行為の有効性から報酬値を判定する学習ルールによって決められる。この時, profit sharing 法では, ルールの実行履歴を用いて, 目標達成に貢献したルールに対して報酬値を均等分配する。ルールがモバイルルールの場合, その送信側エージェントに対して, 報酬を分配することで, 分散ルール強化学習を実現する。

この処理アルゴリズムに基づいてシステム構成要素の機能を次に示す。

(1) メタ推論エンジン

メタ作業領域においてエージェントの状態を表す AgentStatus メッセージと環境からのメッセージに基づいて, エージェントの次処理フェーズと学習モードを決定する。決定した結果は, エージェントの次処理となり作業領域に書き込まれる。学習モード変更の場合は, 学習エンジンに変更する学習モードをメッセージ通知 (LearningMode メッセージ) する。以下の記述は、メタ推論エンジンによって使用されるルール例であり、「複数ルールが発火し, しかもそれらのルールが競合している場合は, モバイルルールを取り込みなさい。」を表している。

```
IF ((AgentStatus (fired-mode) true)
    (AgentStatus (conflict) true))
THEN
   ((AgentExec (get-mobile-rule) true))
```

(2) ベース推論エンジン

作業領域内の AgentStatus メッセージに従い処理フェーズを実行する (P3,P5)。また, メッセージの内容に基づいて, 分散競合解消器, 効果器, 分散学習エンジンを呼び出す。実行後, 現在のエージェントの状態を AgentStatus メッセージとしてメタ作業記憶に書き込む

(3) 分散学習エンジン

メタ推論エンジンからの LearnMode メッセージに従って, 学習モード (profit sharing 法, backet brigate 法, あるいは, 両者の学習モードを実施する hybrid 法) を選択する。また, 作業記憶のメッセージと学習ルールに基づいて報酬 (罰則) を決定し, ルールベースの強化学習を実施する。また, モバイルルールを強化する場合は, その送信側エージェントに対して報酬 (罰則) を送り, 該当するベースルールの強化を依頼する。

(4) 分散競合解消器

相互排他的カテゴリに属する行為を持つ発火ルール (fired rule) を検出する。(P4 の実施) この時, AgentExec メッセージによってモバイルルールの獲得が指示された場合, 協調領域からモバイルルールを取り込み, 発火ルールを含めて競合解消する。相互排他的カテゴリーはエージェント内に定義されており,

例えば, action1 と action2 が相互排他の場合, 次のようなメッセージリストで表現される。

```
((action1 (=ID)true)(action2 (=ID)true))
```

(5) 分散学習エンジン
メタ推論エンジンからの LearnMode メッセージに従って, 学習モード (prohit sharing 法, backet brigade 法, hybrid 法) を選択し, P1,P2,P5,P6 の学習に関係する処理を実施する。また, 作業記憶のメッセージと学習ルールに基づいて報酬 (罰則) を決定し, ルールベースの強化学習を実施する。罰則を与える学習ルールの例を次に示す。

```
IF   ((Event(=ID check) true)
      (Event(=ID overload) true))
THEN ((Reward ((Action1(=ID) ture) -1)
       true))
```

4 QoS 制御方式

QoS 制御としては, 遅延・遅延変動・セル損失率・帯域管理などが考えられる。また, アプリケーションによっては, 遅延・誤り率のどちらかを優先するかで再送制御ついても考慮する必要がある。しかし, 現状では, すべての QoS について厳密に管理を行なうことは, 困難であると考えられる。例えば, End-to-End で帯域管理を行なう場合も, VBR 的なトラヒック, あるいは要求帯域が時間的に変動するもの (例えば参加者が増減するテレビ会議) が入ると, セル／バースト／コールの各レベルで帯域管理を行なう必要が生ずる。

しかも, 帯域予約を行なうためには, メディア自体に帯域予約できること (物理的・統計的) が必要であるが, shared Ethernet のような帯域予約できないメディアが広く使用されている場合, 厳密な帯域予約はできなくなる。

従って, 全ての指標に関して厳密な QoS 管理を行なうのは現実的ではないものと思われる。むしろ QoS 管理としては, 容易に測定可能なデータを用いるかなり粗い指標を用いて行なう方が現実的である。例えば, 各ノードの load factor, リンクの使用率, パケットの廃棄率などを適正な範囲に管理することは少なくとも必要であると考えられる。この時, QoS 管理についても, 各ノード毎にローカルに測定可能な指標を用いて, 適応制御的な QoS 管理を行なうことが望ましい。そこで, 下記においては, リフレクィブエージェントを用いた QoS 制御モデルについて述べる。

4.1 リフレクィブエージェントを用いた QoS 制御モデル

図 6にリフレクティブマルチエージェントによって制御される QoS 制御アーキテクチャモデルの構成を, そして, 図 7に QoS ネゴシエーションのための手順についてに示す。Netscape 等の WWW Client によって, クライアント (ユーザ) からの要求を受理すると, まず, サービスコンテンツに従って, WWW server 側からユーザ要求の QoS 値を設定するための情報を ReflectiveAgent によって取り込まれ, Java Applet の形式でユーザ側の WWW Client に転送する。この時, 具体的には, サーバ側からはサービスコンテンツを実行するのに必要な制約条件 (例えば, サーバ側の連続データの転送が行なえるフォーマットや, その際に必要な通信速度) がデータとして送られる。

次に, WWW Client 上にて, Java Applet が load されると, 要求したサービスコンテンツの内容を表示し, ユーザに対し, 希望する QoS 値の入力 (Frame rate / Display size / Resolution) と, もし, QoS が守れない時に, どの入力値を優先して保証するかを入力させる。[11]

そして, Client のハードウェアの制約を入力することで, ユーザから要求条件とサーバ側の要求条件が整う。ここで, 図 7に示すように, これらの条件から, End-Node 間 (中継ノードを除いた) の通信条件をルール生成する。例えば, 以下のようなルールが出来上がるものと考える。

```
IF ((FRAME_RATE(=x) true)
    (WIND_SIZE(400 600) true)
    (RESOLV("low" ) true)
    (loss_packet(
     (range (=X [0.05 - 0.10]) true))
      true))
THEN ((Action (FRAME_RATE( (- (=x 5) true))
       true) true))
```

このルール例の解釈は次の様になる。「フレームレートが毎秒 x frame, ウィンドウサイズが 400x600,

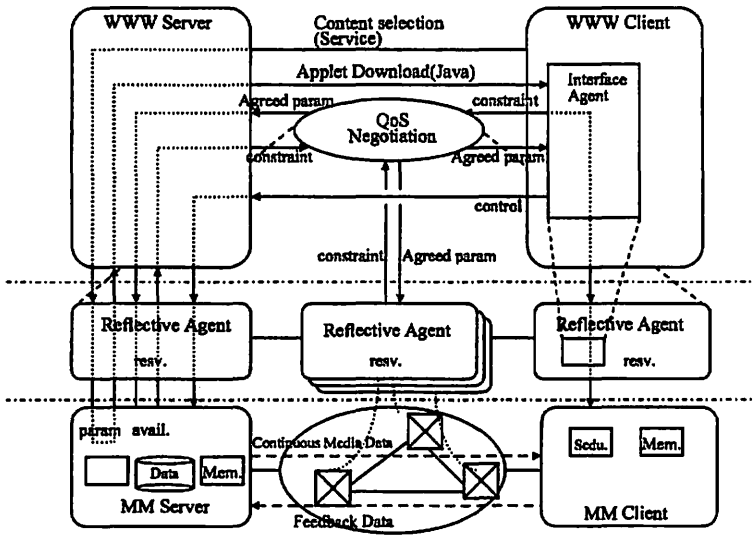

図 6: QoS 制御モデルの概要

解像度が低レベルの時に, パケットの損失率が, 0.05 - 0.1 の間の時は, フレームレートを今の状態から 5 フレーム品質を下げる。」

このようなプロダクションルール群を上述した制約条件下で生成し, 競合部分は, 強化学習によってルールの選択が行なわれる。すなわち, ルール生成部分に関しては, 完全なルール生成を行なう必要はなく, ユーザの入力・ハードウェアの制約条件・基本的な通信条件を入力することで, 考えられるパターンのルール生成が自動的に行なわれる。

次に, マルチメディアサーバ・マルチメディアクライアント・ネットワーク内における経路上のルータ毎に対応するリフレクティブエージェント間の協調によってマッピングされる QoS 値をネゴシエーションする。即ち, 上記で得られるルールに対して, ネットワークリソース・サーバリソースなどの環境情報を入力することで, 動的環境に見合ったコンセプト・ルールの活性化と実行が行なわれ, その時のネットワーク環境に対する適切な QoS 値をトランスポート層レベルで決定する。

次に, 実際のネットワーク環境にマッピングを行なうために, source と sink 間の経路を把握する必要があるが, これは, RSVP モデル [10] と同様に, 受信者サイドより帯域を予約することにより, サーバの load factor(CPU 使用率・バッファ使用率) や, ネットワーク上のリンク使用率・パケット廃棄率を各ノードごとに分散配置することで, トランスポート層以下での QoS の保証を図る。ここまでが, リアクティブ (反応的) に行なわれる処理である。

次にネットワークや, サーバの負荷は, 動的な環境変動としてとらえられるため, エージェントに対してその変動情報を入力することで, ルーティング状態・フロー状態を監視し, 問題があれば, リザベーションのアレンジのやり直し, 経路設定のやり直し, さらに状況に従っては, メディアスケーリング・呼の切断などを行なうことで, ユーザが要求する QoS 値を反映することを試みており, これらは, 一種のネットワークに対するユーザプログラマビリティとして現在詳細検討を行なっている。

5 まとめ

本稿では, 分散強化学習機能を持つルールベースシステムに, メタ推論・メタ学習の考え方を融合し, 動的な環境変動においても, 自己の処理をリフレクティブに変更可能なリフレクティブマルチエージェ

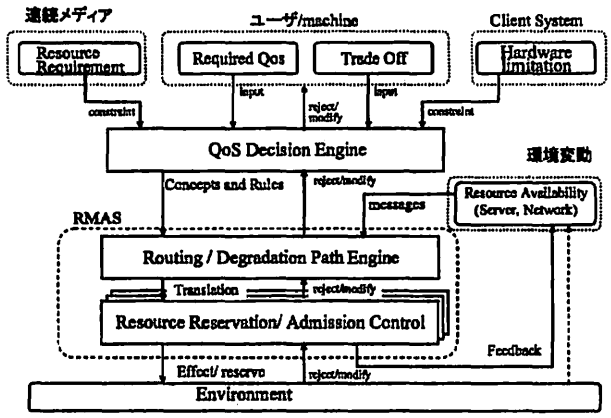

図 7: QoS ネゴシエーション

ントプラットフォームを示した。そして, これを用いた QoS 制御アーキテクチャ を本プラットフォーム上に適用するモデルの提案を行なった。QoS 制御アーキテクチャでは, リアクティブにユーザ要求の QoS を設定し, その後のネットワークの監視を行ないながら QoS を保証するといったモデルを提案することで, 動的なネットワーク環境に対しても, ユーザが何ら意識することなく要求を行なえるユーザプログラマビリティの実現性を示した。現在, リフレクティブマルチエージェントシステムを分散環境上に実験的に構築し, エンドユーザによるサービス定義能力や QoS 制御アーキテクチャにおける処理性能に関する検証を行なっている。今後は, より実時間処理能力を高めるために, プラットフォームでは, ルール生成／修正方式の検討が必要と思われる。また, リフレクティブな処理を行なうために, ネットワークに対してモバイルエージェント機能をもつ自己改変可能なインタプリタータイプのスクリプト言語の検討・開発を行なっていく予定である。

参考文献

[1] 服部進実: “ ユーザ自身によるシステムとサービスの構築運用を可能とする通信ネットワークを目指して” 信学論 (B-I), Vol.J79-B-I, No.5,pp.185-194,(1996.5)

[2] 阿部倫之, 中沢実, 服部進実: “コンセプトネットワークによるルール強化学習に基づくマルチエージェントシステム” 信学論 (B-I), Vol.J79-B-I, No.5,pp.226-238, (1996.5)

[3] J.H.Holland, K.J.Holyoak, R.E.Nisbett, and P.R.Thagard: “Induction: Process of Inference Learning and Discovery,” MITpress, 1986.

[4] 西ヶ谷岳, 栗田敏彦, 飯田一郎, 村上孝三 : “エージェント指向ネットワークアーキテクチャDUET の提案”, 信学論 (B-I), Vol.J79-B-I, No,5 , pp.216-225, (1996.5)

[5] K. Nahrstedt and J.M.Smith: “The QoS Broker” IEEE Multimedia, vol.2, no.1, pp.53-67,1995

[6] A.Campbell, G.Coulson, D.Hutchison:“A Quality of Service Architecture”, ACM SIGCOM, Computer Communication Review, Vol.2, pp.6-27 (1994)

[7] 渡部卓雄: “ リフレクション” コンピュータソフトウェア, Vol.11, No.3, pp.5-14, 日本ソフトウェア科学会, 1994

[8] http://www.genmagic.co.jp/

[9] TINAC, “Overall Concepts and Principles of TINA”,TB_MDC 018.10.94 (1994)

[10] R.Braden,L.Zhang,S.Berson,S.Herzog,S.Jasmin: “Resource ReSerVation Protocol(RSVP) Version.1 Functional Specification”, IETF draft-ietf-rsvp-spec-12.ps, 1996

[11] http://www.dstc.edu.au/FastWeb/