

# マルチエージェントシステムのための 制約論理型言語 RXF の実現

大 園 忠 親<sup>†</sup> 新 谷 虎 松<sup>†</sup>

本論文では、制約論理型言語 RXF の実装とその応用について述べる。本研究は、マルチエージェントシステムを容易に構築するための言語の実装を主眼とする。RXF は、マルチエージェントシステムを容易に構築するための言語として、制約プログラミング、並行処理、およびメタ処理機能を提供する。ここでは、制約プログラミングとして制約論理型言語、並行処理としてマルチスレッド処理を導入する。複数エージェントは、マルチスレッド処理によって並行動作する。RXF では、マルチスレッドの利用に関連して、box モデルに基づくスレッド切替え処理を実現した。並行動作するエージェント間の非同期通信も可能である。RXF において分散した計算機間におけるエージェント間通信は、オペレーティングシステムの利用する通信プロトコルに依存しない。このことによって、ユーザーは、容易に分散環境におけるエージェント間通信を利用することが可能である。RXF において、エージェントプログラムの効率的な開発の支援という観点から、ユーザインタフェースを実装した。応用として、マルチエージェントシステムの例を示す。最後に本研究の有用性について議論し、今後の課題について述べる。

## On Constraint Logic Programming Language RXF for Implementing Multiagent Systems

TADACHIKA OZONO<sup>†</sup> and TORAMATSU SHINTANI<sup>†</sup>

In this paper, we describe an implementation of reflective constraint logic programming language RXF and its application. We develop the language RXF for multiagent programming. The language can provide functionalities for constraint logic programming, concurrent programming and meta-level processing. Multi-thread programming enables concurrent programming. Reflection is used for meta-level programming. In constraint logic programming, we can represent various data by predicates and process numerical data. An agent on RXF runs by using a thread. The agents can communicate each other by ports (message passing mechanisms) on RXF. The port is independent of any protocol. RXF users can program very easily to let agents communicate each other over network. We realize an interface of RXF as effective environment for programming agents. We have implemented applications to see how flexibly and effectively RXF can be used.

### 1. はじめに

近年、エージェント<sup>1)</sup>を用いたユーザサポートシステム<sup>2)</sup>の研究がさかんである。ここでは、互いに並行動作する知識処理主体をエージェントと呼び、複数のエージェントから構成されるシステムをマルチエージェントシステムと呼ぶ。本研究の主目標は、拡張性が高く、動的な環境に適応可能なマルチエージェントシステムを容易に構築するための論理型言語の実装である。本研究で試作した、制約論理型言語 RXF (Reflective Familiar) は、CLP(R)<sup>3)</sup>等と同様な制約論理型言語

を基礎とし、リフレクション<sup>4)</sup>、マルチエージェントシステムの構築の能力を付加した言語処理系である。本論文では、RXF の持つマルチエージェントの並行処理、エージェント間通信<sup>5)</sup>、RXF のユーザインタフェースについて述べる。制約プログラミングおよびリフレクションに関しては紙面の都合上別の機会に論じる<sup>14), 15)</sup>。

論理型言語に基づくエージェントプログラミングの関連研究として、I.C.Prolog<sup>6)</sup>の実装研究がある。I.C.Prolog は、マルチエージェントシステムを構築可能とする論理型言語処理系の実現という主眼から、複数プログラムの並行実行機能、プログラム間の非同期通信、ネットワークを介した通信機能の3点をProlog に付加した論理型言語である。一方、本研究にお

<sup>†</sup> 名古屋工業大学工学部知能情報システム学科  
Department of Intelligence and Computer Science,  
Nagoya Institute of Technology

ける RXF は I.C.Prolog の主目標であるマルチエージェントシステムを構築するための機能に加えて、さらに2つの拡張、すなわち、リフレクションおよび制約論理型プログラミング機能を実現した言語である。また、I.C.Prolog とは異なり、ユーザに対して、オペレーティングシステムや通信プロトコルに依存しないメッセージ通信機構を提供する。さらに、エージェントの構築という観点から、メッセージに対して割込み処理を定義することができる。これにより、問題解決中のエージェントに対する推論の停止処理を容易に実現できる。

エージェント間の通信機構の実現方式として、メッセージ通信方式と共有メモリ方式がある。RXF ではメッセージ通信方式を採用する。共有メモリ方式の通信機構の実装に関連して、SICStus Prolog では Linda<sup>7)</sup> に基づく通信機構が実現されている。Linda では Tuple Space と呼ばれる共有メモリを通信手段として利用する。分散環境における Linda に基づく通信機構の実装には、Tuple Space の実現のためのサーバが必要である。このようなサーバの必要性は、分散環境における通信機構の頑健性を著しくそこなわせる。また、全エージェントに対する共有メモリは、個々のエージェントにおけるデータ隠蔽にとって不都合である。メッセージ通信方式は、分散環境における Tuple Space のような特定のサーバを中継して通信する必要がないため、分散環境における柔軟な通信という点において優れている。また、一般的には、通信相手を探すためのネームサーバが必要となるが、本システムでは、エージェント名に計算機名の情報を付加することができるので(3.1 節参照)、ネームサーバを用いてエージェントがどの計算機上で動作しているのかを特定する必要はない。

ここで特筆すべき点は、RXF は単に Prolog に並行処理の機能や通信機能を付加しただけではなく、並行処理する主体をエージェントと見なし、各エージェントに独立な節データベースと入出力機構を実装し、エージェントの独立性に注目した点にある。また、メッセージ通信に関しても簡潔な書式で記述できるように設計し、ユーザに対して通信のための複雑な手続きなしでネットワークを介した通信機能を提供する。並列処理の関連研究として、PARLOG や KL1 等があげられる。これらの並列論理型言語<sup>8)</sup>と RXF との相違点は、並列論理型言語が並列実行による処理の高速化を主目標にしているのに対し、RXF は複数のエージェントを分散・並行実行する環境を提供することを目的とする点である。

本研究では、多重プログラミングによってエージェントの並行動作を実現する。多重プログラミング機能を持つ代表的なオペレーティングシステムとして UNIX や Mach があげられる。スレッドは、Mach<sup>9)</sup> 上で多重プログラミング機能を実現するために実装された。Mach では、計算機資源の管理の単位をタスクと呼ぶ。UNIX ではプロセスが仮想 CPU と計算機資源の管理の単位の両方を兼ねている。一方、Mach において、計算機資源の管理の単位と仮想 CPU は別のモデルによって表現される。Mach ではスレッドが仮想 CPU を表すモデルである。計算機資源の管理の単位と仮想 CPU を分離することによって、計算機資源を共有する複数の仮想 CPU の並行実行を実現する。

RXF の実装において、多重プログラミングの方式としてスレッドを選択した。理由は、複数のエージェントで節データベースを共有する場合、メモリの共有が容易なスレッドが、メモリの共有に特別な手段を必要とするプロセスよりも有利だからである。

本論文は、2 章において RXF におけるエージェントについて説明する。3 章で RXF の実装について説明する。4 章で RXF におけるプログラミングについて説明し、5 章ではまとめと今後の課題を述べる。

## 2. RXF におけるエージェント

### 2.1 構成

RXF におけるエージェントの特徴は、(a) 自律性<sup>10)</sup>を持つ、(b) 他のエージェントに対して独立である、(c) 複数のエージェントは並行動作する、の3点によって特徴づけられる。(a) の自律性は、エージェントがそれ自身で外部に対して自ら主体的に行動することを表す。(b) の独立性は、エージェントのデータが他のエージェントから保護されていることを表す。(c) の並行性は、複数のエージェントが、概念的には同時に実行していることを表す。RXF では、(a) を実現するために、エージェントが自分自身の状態を参照・変更するための機能として、リフレクションを提供する。

RXF におけるエージェントは制約論理型言語のインタプリタとエージェント制御機構の2つの部品から構成される(図1参照)。インタプリタは言語 RXF のプログラムを実行する部品である。本インタプリタには線形制約問題のための制約解消系が含まれ、与えられた制約を解く。エージェント制御機構は、メモリの管理、ファイルやメッセージなどの入出力管理、イベント管理、そして割込み管理を行う部分である。イベント管理は、イベントの発生と処理を管理する。割込み管理は、特定のイベントに対する割込み処理を管

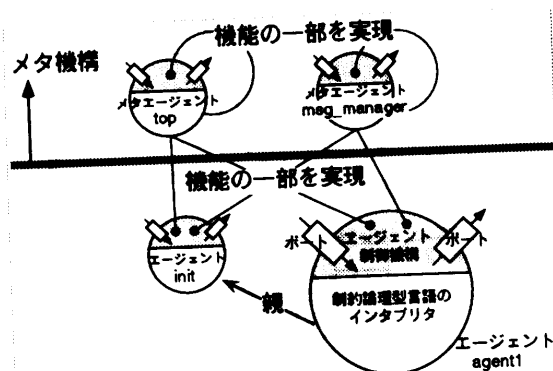


図1 エージェント間の関係  
Fig.1 Relation of agents.

理する。

## 2.2 ポート

RXF では、入出力の管理のためにポートと呼ぶ機構を実装している (図1 参照)。エージェントはポートを用いることによって、他のエージェントやファイルなどの、エージェントにとっての外界に干渉できる。ポートは、ファイルや通信プロトコルなどのオペレーティングシステムに依存するような低レベルな部分を RXF の言語仕様に合わせるための機構を持つ。つまり、ポートは、バイト列や C++ におけるクラスや構造体などの低レベルなデータを RXF の扱える述語形式に変換する機能を備える。エージェントは、デフォルトでは、メッセージ通信のポート、メタレベル表現生成用のポート、およびユーザインタフェース用のポートの3種類のポートを持つ。それぞれのポートには、入力用と出力用のポートがあるので、エージェントは初期状態で6つのポートを持つことになる。メッセージ通信のポートをメッセージポートと呼ぶ。メタレベル表現生成用のポートは、エージェントのメタレベル表現の生成と、メタレベル表現に基づくエージェントの状態変更を実現するためのポートである。メタレベル表現生成用のポートをエージェントポートと呼ぶ。

## 2.3 エージェント間の関係

RXF におけるエージェント間の関係を図1に示す。すべてのエージェントは名前を持つ。エージェントには、標準的なエージェントと、標準的なエージェントの機能の一部を実現するためのエージェントに分類される。本論文では、後者をメタエージェントと呼ぶ。エージェントを生成したエージェントをそのエージェントの親と呼ぶ。標準的なエージェントは、システム組込みの init と呼ばれるエージェントを親とする。たとえば、図1中のエージェント agent1 は、init を親

とする。標準的なエージェントとメタエージェント間には親子関係が存在しない。

システム組込みのメタエージェントとして、top エージェントと msg\_manager エージェントがある。top エージェントと msg\_manager エージェントを合わせてシステムエージェントと呼ぶ。システムエージェントは、標準的なエージェントに対してメッセージ通信やファイル入出力などの機能を提供するためのエージェントである。リフレクションに関連して、システムエージェント以外にもユーザが標準的なエージェントに対するメタエージェントを定義することもできる。リフレクションに関連して、問題解決のためのメタな推論を行うメタエージェントを定義することによって問題解決のための計算と問題解決のための計算を制御するための計算を分離することができる。

init エージェントは RXF が起動したときに3番目に起動される。最初に起動されるエージェントは top エージェントと呼ばれる。標準ではユーザインタフェースは top エージェント上で稼動する。2番目に起動されるエージェントは msg\_manager である。msg\_manager エージェントは、メッセージ通信機構に相当し、メッセージ通信を実現する。

これら複数のエージェントは、単一 RXF 上では、複数のスレッドとして並行実行される。標準ではエージェントはそれ自身がインタプリタを持ち、他のエージェントと独立な節データベースを持つ。エージェント生成時のオプションによって、親エージェントと子エージェントが同じ節データベースを共有することを指定することが可能である。デフォルトでは、子エージェント生成時に、親エージェントの持つ節データベースの内容は、子エージェントの持つ節データベースにコピーされる。また、オプションとして、エージェントの生成時に、子エージェントの節データベースに、親エージェントのプログラムをコピーしないことも指定できる。

## 2.4 動作状態

RXF におけるエージェントのインタプリタの動作状態として、RUN, WAIT, SUSPEND, SLEEP の4種類がある。これらの動作状態は節データベースやメッセージの相互排除や、デバッガによって使われる。RUN 状態は、インタプリタがプログラムの解釈・実行中であることを表す。WAIT 状態と SUSPEND 状態は、インタプリタがプログラムの解釈・実行を一時停止している状態を表す。WAIT 状態はイベント待ちや、共有メモリの相互排除などに利用される。SUSPEND 状態と WAIT 状態との相違点は、WAIT 状態

のときはインタプリタは新規のプログラムを解釈・実行することはできないが、SUSPEND 状態のときは新規のプログラムを解釈・実行できる点である。このとき一時中断されたプログラムはスタックに保存される。SUSPEND 状態のとき、resume イベントが発生するとエージェントはスタックの先頭にあるプログラムの実行を再開する。SLEEP 状態は解釈・実行中のプログラムがない状態を表す。インタプリタは、プログラムの実行を終了したときに SLEEP 状態に移行する。インタプリタは、SLEEP 状態のときに新規のプログラムの解釈・実行を依頼されると、RUN 状態に移行しプログラムの解釈・実行を開始する。

### 3. RXF の実装

筆者等は、Macintosh 上で動作する RXF と NEXTSTEP 上で動作する RXF を実装した。Macintosh 上で動作する RXF の実装は C++ 言語を用いた。Macintosh 上で動作する RXF の実装は HyperCard と簡単に通信するための機能をも備えている (4.2 節参照)。HyperCard は Macintosh 上で利用可能なオブジェクト指向プログラミング環境である。NEXTSTEP 上で動作する RXF の実装は Objective-C 言語を用いた。

#### 3.1 スレッドに基づく並行処理機能

1 つの CPU で複数のスレッドを並行動作させるには、実行するスレッドを次々に切り替えればよい。スレッドの実行順序のスケジューリングとし実行権の横取りを許すか許さないかで、プリエンプティブスケジュールとノンプリエンプティブスケジュールに分けられる。RXF の Macintosh 上の実装は、Macintosh プログラミング環境の制約のためノンプリエンプティブスケジュールを使用している。

ノンプリエンプティブスケジュールにおいて、実行中のスレッドを切り替えるためには、実行中のスレッドを切り替えるための手続き (Yeild) を呼ぶ必要がある。複数のスレッドを並行動作させるためには、必ず有限時間内に Yeild を呼ばなければならない。さらにスレッドを切り替えるオーバーヘッドを考慮して、Yeild をあまり頻繁に呼ばないことが望ましい。RXF では、以上の 2 点を考慮して、Yeild を呼ぶタイミングを決定する必要がある。

本実装において、Yeild を呼ぶタイミングは、box モデル<sup>11)</sup>に基づいて決定される。box モデルとは、Prolog インタプリタの実行モデルである。box モデルではプログラムの実行を 2 入力、2 出力の box の生成、消滅で表現する。入力端子には Call, Redo がある。

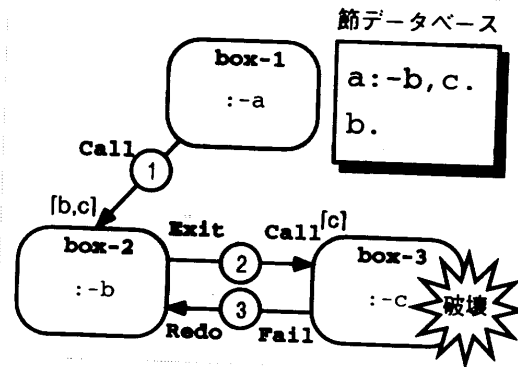


図2 box モデルに基づくスレッド切替えタイミングの検出  
Fig. 2 Timing of thread swithing based on the box model.

出力端子には Exit, Fail がある。Exit は Call と接続されるための端子で、Fail は Redo と接続されるための端子である。

box モデルに基づくスレッド切替え処理は、box が制約論理型プログラミングにおけるインタプリタの処理の区切と見なせることを利用する。以下に box モデルに基づくスレッド切替え方式を説明する。

図2の丸で囲まれた数字が Yeild を呼ぶタイミングを表す。角の丸い四角は box モデルにおける box を表す。ここで節データベースは図2の節データベースと書かれた四角の中で示し、RXF への質問が “a” であるとする。ここで、ゴール “:-a” はスレッド T で評価されるとする。始めに、ゴールを評価するために box が生成される。ここでは、box-1 が生成された。“a”を検索したところ “a:-b,c” が発見されたので “b” を評価するために box-2 が生成される。box-2 が生成された後に Yeild が呼ばれる (図2の1)。スレッド T の実行権は、Yeild が呼ばれることによって他のスレッドに渡される。他のスレッドから再びスレッド T に実行権が渡されると、box-2 において “b” が評価される。“b” が成功し、“c” を評価するために box-3 が生成される。ここでも Yeild が呼ばれ、スレッド T の実行権は他のスレッドへ渡される (図2の2)。スレッド T に実行権が戻ると、box-3 では “c” が評価されるが節データベースに “c” は存在しないので “c” は失敗する。質問が失敗すると box は破壊される。つまり、box-3 が破壊される。破壊された後 Yeild が呼ばれる (図2の3)。Yeild が呼ばれることによってスレッド T の実行権は他のスレッドに移る。他のスレッドからスレッド T に実行権が戻ると、この後 box-2 の “b” が再実行される。あとは “b” と “a” がそれぞれ失敗し質問 “a” は失敗する。以上のように box の生成・破壊時に Yeild を呼ぶことで、複数のプログラムの並行動

作が可能になる。

### 3.2 メッセージ通信

アプリケーションとして実行中の RXF 上では並行動作するエージェント群は互いに通信できる。もし、2 つの RXF が異なる計算機上で起動されていても、すべてのエージェントはお互いに通信可能である。Macintosh 上における RXF のエージェントは、アプリケーション間通信のための AppleEvent<sup>12)</sup>、および計算機間の標準的な通信プロトコルである TCP/IP を用いた通信が可能である。NEXTSTEP 上の実装では Mach の提供する IPC を用いた高速な通信も可能である。プログラマが簡単に利用できる通信機構を実装するためには、プログラマが実際にどのような通信機構を使っているか分からなくても通信できるようにする機構が必要である。異なるオペレーティングシステム間の場合（たとえば、Macintosh と NEXTSTEP 間）は、標準的に利用されている TCP/IP を標準の通信プロトコルとして利用する。

RXF で使われている次の通信用プリミティブについて説明する。

**send(To, Message)**

**receive(From, Message)**

send はエージェント To にメッセージ Message を送信するプリミティブである。To が単にアトム有的时候は、同一 RXF 上のエージェントを意味する。To が引数を 1 つ持つとき、その引数は計算機名を表す。たとえば、

**send(agent1, ok)**

は、同一 RXF 上のエージェント agent1 へメッセージ ok を送信する。また、

**send(agent2(host2), ok)**

は、計算機名 host2 における RXF 上のエージェント agent2 へメッセージ ok を送信する。現状では、同一計算機上の複数の RXF を区別することはできない。これは、エージェント名と計算機名だけで目的のエージェントを指定できるという簡潔なエージェント指定方法をプログラマに提供するためである。すなわち、単一計算機上の並行動作を実現するためには、RXF を複数起動する必要はなく、単一の RXF 上でエージェントの並行動作が記述可能である。

receive は Message と単一化可能なメッセージを受信し、その送り手として From を得る。たとえば、From として agent2(host2) が得られた場合、計算機名 host2 における RXF 上の agent2 が送り手となる。条件に適合するメッセージがなかった場合、receive のデフォルトの動作は失敗である。もし、条件に適合するメッ

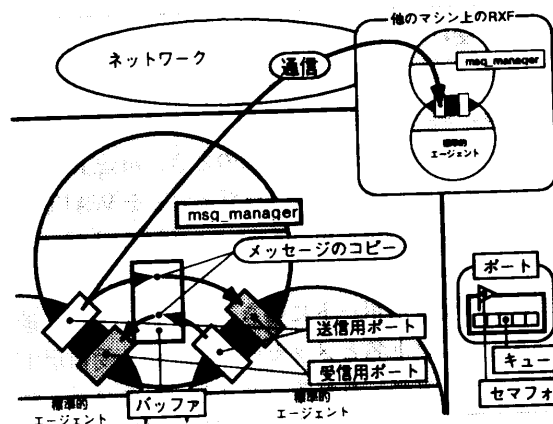


図3 RXFにおけるメッセージ通信  
Fig. 3 Message passing in RXF.

ッセージが到着するまでプログラムの実行を中断したいときは、以下の形式で receive を用いる。

**receive(From, Message, [wait])**

ここでは、メッセージの到着待ち時において、インタプリタは WAIT 状態になる。図3はRXFにおけるメッセージ通信機構を表す。

msg\_manager エージェントは、標準的なエージェントのエージェント管理機構の一部として動作する。msg\_manager エージェントと標準的なエージェントは、メッセージポートを共用する。メッセージ通信は、msg\_manager と標準的なエージェント間で共用されたメッセージポートによって実現される。メッセージポートには、送信用ポート、受信用ポートの2種類がある。送信用ポートと受信用ポートは、キューとセマフォを構成要素とする。メッセージポートへの書き込みは、メッセージポートに含まれるキューへの追加を意味する。メッセージポートからの読み込みは、メッセージポートに含まれるキューの先頭要素を取り出すことを意味する。メッセージポートは、msg\_manager エージェントと関連するエージェント間の共有メモリなので、相互排除を行う必要がある。メッセージポートにおける相互排除は、セマフォに基づく相互排除処理によって実現している。

エージェントはメッセージポートに受け手のエージェント名とメッセージの対を送信用ポートに書き込む。msg\_manager エージェントは標準的なエージェントの送信用ポートを監視している。msg\_manager エージェントは、送信ポートに対する書き込みを検出すると、その送信ポートに対して書き込みをしたエージェント、つまり、メッセージの送り手のエージェントを WAIT 状態にする。msg\_manager エージェントは、メッセージを msg\_manager エージェントの管理するバッファに

コピーし、送り手のエージェントを RUN 状態にする。このとき受け手のエージェントが、送り手のエージェントと同一 RXF 上に存在するとき、そうでないときがある。同一 RXF 上に存在するとき、かつ、受け手のエージェントが RUN 状態のとき、msg\_manager エージェントは、受け手のエージェントを WAIT 状態にする。その後、msg\_manager エージェントは、送り手のエージェント名とメッセージの対を受け手のエージェントの受信用ポートに書き込む。ここでは、送り手のエージェント名とメッセージの対が、受信用ポートのキューにコピーされる。メッセージのコピー処理前の受け手のエージェントの状態が RUN 状態だったならば、受け手のエージェントの状態を RUN 状態に戻す。ここでのメッセージのコピーに関する冗長な処理は、論理型言語の持つバックトラックの機能に対応するために行われる。すなわち、送り手のエージェントがバックトラックによってメッセージのデータを破壊しても、msg\_manager エージェントや受け手のエージェントに影響のないように冗長にコピーする。実際には送り手がバックトラックによってメッセージを破壊してしまうまではコピーする必要がないことに着目することにより本メッセージ通信の高速化を実現する。

指定された計算機が送り手のエージェントの存在する計算機と同一の計算機でないときは、指定された計算機上に存在する RXF の msg\_manager エージェントに対して送り手のエージェント名、受け手のエージェント名、そしてメッセージの 3 つ組を送信する。指定された計算機上で実行中の RXF が存在しなかったときは、送信を中止する。受け手の存在する RXF 上の msg\_manager エージェントは、メッセージを受け取ると、送り手のエージェントと同様にメッセージの送信処理を実行する。

RXF におけるメッセージ通信の特徴は、単一 RXF 上におけるエージェント間通信だけではなく、簡潔なエージェント指定方式や、msg\_manager エージェントを用いてネットワークを利用するための手続きの隠蔽を実現することにより、ネットワーク上に分散した計算機上で動作する RXF におけるエージェント間のメッセージ通信も容易に実現する点にある。

### 3.3 ユーザインタフェース

RXF におけるユーザインタフェースは、エージェントプログラムを効率的に実装するためのプログラミング環境を提供する。新たに生成されたエージェントには、標準的なユーザインタフェースが暗黙的に用意される。

図 4 は、Macintosh 上に実装された RXF のユーザ

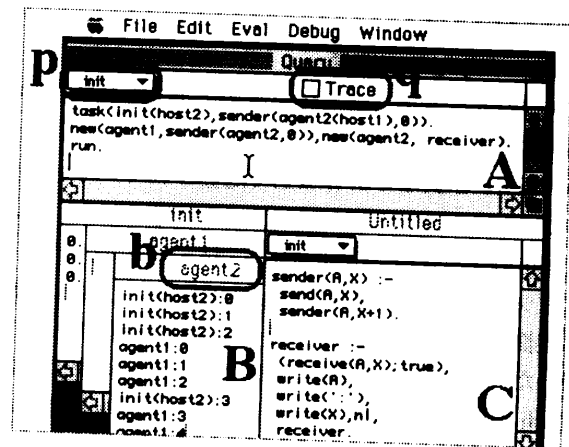


図 4 RXF のユーザインタフェース  
Fig. 4 The user interface of RXF.

インタフェースを表している。

本ユーザインタフェースにおいて、ウィンドウの種類は、コンソールウィンドウ (図 4 の A)、エージェントウィンドウ (図 4 の B)、プログラムウィンドウ (図 4 の C) の 3 種類がある。

コンソールウィンドウは、RXF のユーザインタフェースにおいて、必ず 1 つ存在する。ユーザは、コンソールウィンドウを用いることによって、システムエージェントを除くすべてのエージェントに対して質問できる。コンソールウィンドウの内容は、ポップアップメニュー (図 4 の p) で選択されたエージェントによって評価される。チェックボックス (図 4 の q) は、デバッグ使用の有無を指定する。

エージェントウィンドウは、各エージェントに対する標準的なユーザインタフェースである。たとえば、図 4 の B のエージェントウィンドウは、図 4 の b で示されたエージェントのインタフェースである。ユーザは、エージェントウィンドウを使ってエージェントに質問することもできる。エージェントは、出力をエージェントウィンドウへ出力する。

プログラムウィンドウは、プログラムを編集するためのウィンドウである。プログラムウィンドウにもコンソールウィンドウと同様なエージェント選択用のポップアップメニューがある。ポップアップメニューによって、プログラムウィンドウ中で定義されたプログラムをどのエージェントで利用するかを指定することが可能である。プログラムウィンドウのタイトルには、プログラムを保存するファイル名が記されている。

## 4. プログラム例

### 4.1 マルチエージェントシステム

図 4 を用いて RXF を用いた簡単なマルチエージェ

ントシステムの構築例を示す。本例では、図4のプログラムウィンドウで定義されたプログラムを実行する。図4は、計算機 host1 上の RXF 上でデフォルトのエージェント init を用いて、エージェント agent1, agent2 を生成し、プログラム編集用のウィンドウを開いた状態を表す。このとき、他の計算機 host2 上にも RXF が起動されている。エージェント init, agent1, agent2, init(host2) は、同一のプログラムを持っているとする。ここで、init(host2) は、計算機 host2 上の RXF におけるデフォルトのエージェント init を示す。図4中のCのプログラムウィンドウ中のプログラムは、ジェネレータとコンシューマのプログラムである。ジェネレータは、エージェント agent1 と init(host2) で実行され、エージェント agent2 で実行されるコンシューマに整数を送り続ける。コンシューマはジェネレータから送られてきた整数を受け取り、送り手のジェネレータの名前と受け取った整数を表示する(図4のエージェントウィンドウB)。図4のAで示されるコンソールウィンドウのように記述することによって、エージェントにプログラムを実行させることができる。ここでは、1行目で異なる計算機上の RXF 上のエージェント init(host2) に対して、sender(agent2(host1),0) を評価させる。2行目でエージェントを生成し、3行目でエージェントを RUN 状態にする。2行目では、名前が agent1 で sender(agent2,0) を評価するエージェント(ジェネレータ)と、名前が agent2 で receiver を評価するエージェント(コンシューマ)を定義している。図4のBには、実行結果が表示され、agent2 が同一計算機 host1 上のエージェント agent1 からと、異なる計算機 host2 上のエージェントからメッセージを受け取っていることが表示されている。

RXF を用いることによって、並行処理とネットワークを介した通信を利用するプログラムを容易に記述することが可能になる。

#### 4.2 データベースシステム

マルチスレッドを利用したデータベースの実装を示す。本例では、すべての標準のエージェントが init の持つ節データベースを共用している。本データベースは、サーバ・クライアント方式のデータベースである。本データベースシステムは RXF と HyperCard を用いて構築される。グラフィカルなユーザインタフェースの構築という観点から、RXF 上のクライアントは、HyperCard を利用したユーザインタフェースを実現する。ユーザは HyperCard によって検索の実行を司令する。HyperCard は RXF 上のクライアントに検索の指示を転送する。クライアントはサーバに検索を依

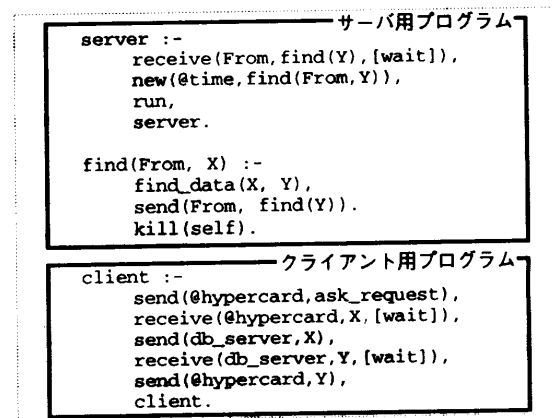


図5 データベースシステムのプログラム

Fig. 5 The program of the database system on RXF.

頼する。サーバは新規にエージェントを生成する。すなわち、クライアントごとに1つのエージェントが生成されることにより、新規に生成されたエージェントが実際に検索を実行する。新規に生成されたエージェントは検索結果をクライアントに返す。クライアントは検索結果を HyperCard に表示させる。クライアントの要求ごとにエージェントを生成することによって複数の要求を並行に処理することが可能となる。

本実装例ではデータベースの機能として検索だけを示す。図5はデータベースを実現するプログラムリストである。本プログラムでは、主に RXF の通信用プリミティブ send および receive が用いられる。

図5のプログラムにおいて、サーバ名は db\_server とする。図5にプログラムは3つある。上から2つがサーバ用で、一番下がクライアント用である。server プログラムは、“find(Y)”にマッチするメッセージの受信 (receive), 検索を実行するためのエージェントの生成 (new), そして、実行 (run) を繰り返す。@time は cpu 時間を返す関数であり、ここでは、cpu 時間をエージェント名として利用している。find に関連した2番目のプログラムは、server プログラムによって生成されたエージェントによって実行される。本プログラムは実際にデータを検索し、依頼主に検索結果を送り、エージェントを消滅させる。エージェントの消滅は kill(self) で行われる。client プログラムは、クライアント用のプログラムである。クライアント用のプログラムは、プログラム本体の最初で HyperCard にユーザからの要求を転送するように依頼する。ここでは @hypercard により、同じ計算機上で起動されている HyperCard を検索しメッセージの受け手として指定できる。本プログラムは、HyperCard に ask\_request というメッセージを送る。その結果、HyperCard で

は, ask\_request に関連した手続きが起動され, ユーザの要求をクライアントに送る. 次の receive は, HyperCard からのメッセージを待つ. HyperCard から送られてきたメッセージは送り手の名前が hypercard になる. 次に, サーバ db\_server に要求を送信する. その後, 検索結果を待ち, HyperCard に検索結果を送信する. クライアントは以上の処理を繰り返す.

## 5. おわりに

本論文では, 制約論理型言語 RXF の実装と応用について述べた. RXF における通信機構の特徴として, メッセージ通信方式でかつ分散した計算機間におけるオペレーティングシステムに依存しないエージェント間通信が可能となる点があげられる. この特徴によって, ユーザは, 分散環境におけるエージェント間の通信を容易に利用することが可能になった<sup>2),13)</sup>. また, RXF では, マルチスレッドの利用に関連して, box モデルに基づくスレッド切替え処理を実現した. box モデルに基づくスレッド切替え処理によってノンプリエンパティブスケジュールにおけるスレッド切替えを効率良く行うことができた. スレッドの並行実行に関連して, RXF では, エージェントの持つインタプリタの実行状態を定義することによって, 共有メモリの相互排除を可能とした. RXF は, 効率的にエージェントプログラムを構築するためのマルチスレッドに基づくユーザインタフェースを提供する. 今後の課題は, マルチエージェントシステムを効率的に開発するためのデバッガの実装であり, 現在, メッセージの監視に基づくツールを試作中である.

## 参考文献

- 1) 石田 亨: エージェントを考える, 人工知能学会誌, Vol.10, No.5, pp.663-667 (1995).
- 2) 川上義雄, 伊藤孝行, 新谷虎松: 分散 TMS に基づくマルチエージェントシステムの試作, 第9回人工知能学会全国大会講演論文集, pp.267-270 (1995).
- 3) Lassez, C.: Constraint Logic Programming, *BYTE Magazine*, August, p.17 (1987).
- 4) 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 (1994).
- 5) 大園忠親, 新谷虎松: アプリケーション間通信機構に基づくマルチエージェント制御機構の実現, 人工知能学会全国大会講演論文集, pp.295-298 (1994).

- 6) Chu, D.: I.C.Prolog: A Language for Implementing Multi-Agent Systems, *Proc. Special Interest Group on Cooperating Knowledge Based Systems* (1993).
- 7) Carreiro, N. and Gelernter, D.: Linda in Context, *Comm. ACM*, Vol.32, No.4 (1989).
- 8) 淵 一博: 並列論理型言語 GHC とその応用, 共立出版 (1987).
- 9) Boykin, J.: Mach オペレーティングシステム, トッパン (1994).
- 10) 伊藤正美, 市川惇信, 須田信英: 自律分散宣言, オーム社 (1995).
- 11) Clocksin, W.F. and Mellish, C.S.: *Programming in Prolog*, Springer-Verlag (1981).
- 12) Apple Computer, Inc.: *Inside Macintosh*, Vol.6, Addison Wesley (1991).
- 13) 大園忠親, 柴田正弘, 新谷虎松: プロダクションシステム KORE/IE の分散化とその応用について, 第48回情報処理学会全国大会論文集, Vol.2, pp.187-188 (1994).
- 14) 大園忠親, 新谷虎松: リフレクティブ制約論理型言語 RXF の設計とその実装, 第9回人工知能学会全国大会講演論文集, pp.299-302 (1995).
- 15) 大園忠親, 新谷虎松: 制約論理型言語 RXF におけるリフレクションについて, 第10回人工知能学会全国大会講演論文集, pp.123-126 (1996).

(平成8年10月6日受付)

(平成8年7月4日採録)



大園 忠親 (学生会員)

1972年生. 1995年名古屋工業大学工学部知能情報システム学科卒業. 同年同大学院工学研究科博士前期課程入学. 分散人工知能, ヒューマンインタフェースの研究に従事. 人工

知能学会会員.



新谷 虎松 (正会員)

1955年生. 1982年東京理科大学大学院理工学研究科修士課程修了. 同年富士通(株)国際情報社会科学研究所入所. 知識情報処理, 論理プログラミングなどの研究に従事. 現

在, 名古屋工業大学知能情報システム学科助教授. 工学博士. 分散人工知能, 意思決定支援システム, ヒューマンインタフェースの研究に従事. 電子情報通信学会, ソフトウェア科学会, 人工知能学会など各会員.