記 号 処 理 57－4
（1990．11．22）

計算モデルLindaのTAOへの導入
― 近傍系理論に基づく計算モデルのための動的計算場について ―

村上 健一郎† 明石 修† 天海 良治† 奥乃 博‡
† NTT ソフトウェア研究所 ‡NTT 基礎研究所

概要

NUE-Lindaは、ネットワークで結合された、さまざまな種類のコンピュータが混在する非均質システムにおいて、近傍系理論に基づく計算モデルのための計算場を提供するシステムである。近傍系理論では、システムの個々の要素が、これに近い要素との相互作用によって計算を進め、互いにオーバラップした多くの近傍の系によってトータルな機能が実現されると考える。NUE-Lindaにおける近傍の計算場は、複数のローカルな計算場の集合を論理的に単一の空間として取り扱うことによって実現される。本論文では、この近傍の計算場(タプルスペース)の概念、および、それを可能とするアクティブなタプルスペースの構造と動作について説明する。また、記号処理言語TAOを用いて、アクティブなタプルスペースの核となる仮想マシンの構造を説明し、それに用いられるNUE-Linda通信プロトコルについて詳しく述べる。

NUE-Linda: Design of a Computation Space for Heterogeneous Systems based on Connectics Theory

Ken-ichiro Murakami† Osamu Akashi† Yoshiji Amagai† Hiroshi G. Okuno‡

† NTT Software Laboratories ‡ NTT Basic Research Laboratories
3-9-11, Midori-cho, Musashino-shi, Tokyo 180, Japan

NUE-Linda is designed for heterogeneous systems consisting of various kinds of and a large number of computers. It provides a common computation space for cooperative work by multiple agents based on "Connectics Theory". Connectics Theory assumes that cooperative work of neighboring objects defines behavior of its enverous and the overlap of these multiple enverous eventually defines total system behavior. This paper describes the structure of the common computation space, that is, tuple space in NUE-Linda. It focuses on dynamically configurable tuple space and a virtual machine which realizes active tuple space of NUE-Linda.

1 はじめに

近傍系理論 (Connectics)[1] は、個々の要素の総和を越える全体というものを、個々の要素間の所産と見る立場で“科学”し、それを“工学”することによって、新しいシステム分析論やシステム構成論を展開するものである。近傍系理論では、システムの個々の構成要素の性質や機能よりも、個々の要素とこれに近い要素が作る (情報空間や距離空間での)“近傍”の性質や機能がむしろ重要な概念となる。すなわち、システム全体の性質や機能が、お互いにオーバラップしたたくさんの近傍の系によって決定されると考える。(図 1-1) ここで、システムと呼んでいるものは、非常に一般的な意味を持つ。例えば、阪谷、野島らは、心理学の立場から、ネットワークユーザ間の利用、操作に関する情報の伝達構造 [2] にも、同様の構造があることを指摘している。

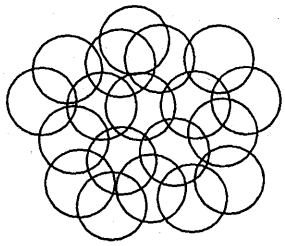

図 1-1 Connectics

一方、コンピュータの世界では、米国を中心とした巨大なコンピュータ複合体“インターネット (Internet)” [3] が出現しており、十数万台のコンピュータを擁しながら、なお爆発的な成長を見せている。このような巨大なシステムの特徴は、さまざまなベンダの供給する、さまざまな種類のコンピュータから構成される非均質システム (heterogeneous system) となっていることである。また、システムは、近傍のスケールで日々変化している。これにより、システムは、いわば“コンピュータの生態系”という様相を示している。このような生態系をシステム設計の立場で見れば、近傍系を適切に設定、設計して、高度に知能的な問題を解かせることも重要な課題である。

そこで、近傍系計算のモデルを、このような非均質システムへ適用することを試みる。NUE-Linda では、計算モデル Linda[4] におけるタプルスペースを基本的な計算場とし、近傍の計算場というものをクラスタ化されたタプルスペース群として実現する。NUE-Linda における近傍の関係は、静的ではない。即ち、時間により、その近傍の関係が変化し、オーバラップの関係も異ったものとなる。本論文では、まず、NUE-Linda における計算モデルを説明する。次に、近傍の計算場を実現するための核となる仮想マシン (NUE-Linda インタプリタ) の構造について、記号処理言語 TAO [5] を用いて説明する。更に、仮想マシンがタプルスペースおよびタプル操作を行うために利用する通信プロトコルについて言及する。

2 NUE-Linda の計算モデル

多数の異ったベンダの供給する、多数の異ったコンピュータを持つ非均質システムでは、その基盤となる共通のパラダイムあるいは計算モデルを導入する場合には、それが、個々のシステムに依存しないような、簡易なものでなければならない。Linda は、タプルを操作するわずか 4 つのプリミティブによって、生成的会話 (generative communication)[6] を行い、計算を進めてゆく簡易な計算モデルである。しかし、このモデルは、全てのプロセスに共有される仮想的な単一仮想空間タプルスペースを前提としているため、インターネットのような巨大で、動的に各構成要素の関係が変化する非均質システムには向かない。そこで、これに近傍の概念を導入する。この新たな計算モデルを NUE-Linda と呼ぶ。

2.1 Linda 計算モデル

Linda モデルには、全てのプロセスに共有される仮想的な一つの空間、タプルスペース (tuple space) が存在する。ユーザプロセスは、タプルスペースを介し、オブジェクトであるタプル (tuple) をやりとりし、生成的会話を行なうことにより、並行計算を実現する。

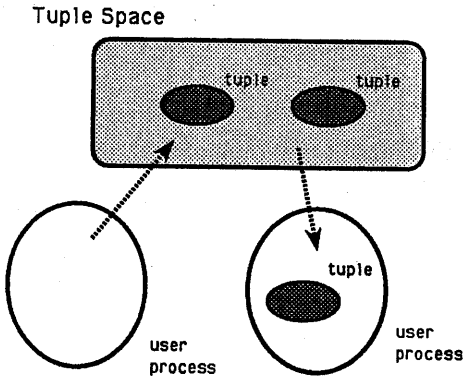

図 2-1 Linda の生成的会話

タプルスペースは、抽象化された共有メモリであり、その実体は、共有メモリであろうと、ネットワークを介

して接続されているものであろうとユーザ側は意識する必要はない。

Lindaモデルにおけるプリミティブは、rd、in、out、evalの4つ[1]である。evalは、タプルを出すという操作に関してはoutと似ているが、引数の評価を別プロセスにまかせる。このため、Lindaモデルにおいて新たなプロセスを作る手段ともなる。

primitive	操作の内容
rd	タプルの値をタプルスペースから読む
in	タプルをタプルスペースから取り除き読む
out	タプルをタプルスペースに出す
eval	引数を別プロセス中で評価し、その結果をタプルとしてタプルスペースに出す

表 2-1 Lindaのプリミティブ

rd、in操作において、その対象となるタプルの選択は、outされたタプルの値と、それらの操作の引数とのパターンマッチングによって決まる。すなわち、論理名によるアクセスが実現されている。マッチするタプルが複数存在する時は、そのなかから非決定的に1つのタプルが選ばれる。逆に、マッチするタプルが存在しなかった場合は、適当なタプルがタプルスペースに出されるまで、ブロックされる。すなわち、Lindaでは、操作自体に同期機構が組み込まれており、明示的にロックをする必要がない。

また、out操作時に変数を書いた場合には、タプルのそのフィールドの部分は、rd、in操作時のどの値ともマッチする。また、rd、in操作時に引数として変数を与えると、その部分はどの値ともマッチし、そのマッチした値が変数に代入される。ただし、変数と変数はマッチしない。このマッチングの例を表2-2に示す。

タプル	マッチするパラメータ
("abc" 1 2)	○ ("abc" 1 2) ○ ("abc" ? x ? y) × ("abc" ? x ? x)
("abc" 1 ? x ? x)	○ ("abc" 1 2 2) ○ ("abc" ? x 2 2) × ("abc" 1 2 3)

表 2-2 マッチングの例

[1]C-Lindaでは、4つのプリミティブに加え、サスペンドしないinである inp、およびサスペンドしないrdである rdpがある。

2.2 近傍系の計算モデル NUE-Linda

NUE-Lindaでは、複数のタプルスペースをクラスタ化し、単一のタプルスペースとしてタプル操作を行うことが可能である。このため、同一のタプルスペースを包含する近傍の各プロセッサは、このオーバラップした計算場上において相互に影響を及ぼしあったり、情報を共有することが可能となる。(Clustered Multiple Tuple Space) (図2-2) また、操作の対象となるタプルスペースは、動的に変更可能である。即ち、タプルスペースのスコープは、動的に変化する。(Dynamically Configurable Tuple Space) このことは、刻々と近傍の関係を変化させて計算を進めることができることを意味している。

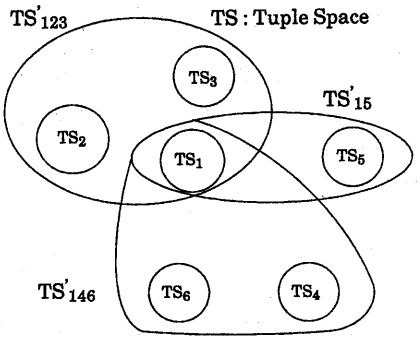

図 2-2 タプルスペースのオーバラップ

操作の対象となるタプルスペースが複数となっても、Lindaにおけるタプル操作のためのプリミティブに変更はない。近傍(対象とするタプルスペース)は、オプションとして指定する。これには、2つの方法がある。まず、プリミティブneighborhoodを使用する方法である。もう一つの方法は、inやoutのキーワードアーギュメント:neighborhoodを使用する方法である。どちらの場合も、タプルスペースの名前のリストで、近傍を指定する。名前には、ts0.ntt.jpのようなドメイン形式表記[3]を用いる。これら2つの方法によって、いつでも近傍の指定を変更することができるので、タプルを操作するごとに、対象となる近傍のタプルスペースを変更することもできる。但し、この操作の対象は、ディフォルトではローカルなタプルスペースであるため、従来のプログラムは、変更することなしに実行可能である。(上位互換性)

neighborhoodと同様にプリミティブおよびキーワードアーギュメントとして用意されているものに、time-to-liveやprivilegeがある。前者は、outに対してはタプルの生存時間、そして、inやrdに対してはタイ

ムアウトを指定するものである。後者は、タプルおよびタプルスペース操作のための特権を指定するものである。

次に、複数のタプルスペースを対象とするoutとinの動作を説明する。

(1) 複数のタプルスペースを対象とするout

outでは、対象となるタプルスペースのいずれかが、タプルの実体を受け取る。このタプルスペースをプライマリタプルスペース(primary tuple space)と呼ぶ。受け取ったタプルスペースでは、このタプルのコピーを残りのタプルスペースへout[2]する。プライマリタプルスペースは、コピーされたタプルに対するアクセス権制御(排他制御)に使用するキー(タプル)を生成し、プライマリタプルスペース自身にoutする。そのタプルへのin要求が、どこかで発生した場合には、まず、コピーされているタプルのアクセス権を取るため、プライマリタプルスペース上のキー(タプル)をinpする必要がある。これに成功した場合には、すべてのコピーされたタプルをinし、コピーを消去する。

(2) 複数のタプルスペースを対象とするin

inでは、対象となるタプルスペースのすべてに対して、in要求を行う必要がある。これは、outの場合と同じく、対象となるタプルスペースのいずれかが、要求を受け取る。このタプルスペースもプライマリタプルスペースと呼ぶ。要求を受け取ったプライマリタプルスペースでは、残りのタプルスペースへ要求を出す。どれかのタプルスペースで、このin要求が成功すれば、他のタプルスペースへの要求は、キャンセルされ、ただ一つのタプルスペースからのinだけが実行されることになる。

2.3 アクティブタプルスペース

Lindaでは、タプルスペースは、単なる通信のための空間であった。(Passive Tuple Space) しかし、NUE-Lindaでは、近傍の計算場を実現するために、タプルスペース自体がアクティブとなる。即ち、タプルスペース上に仮想的なマシンが存在する。(Active Tuple Space) この仮想マシンは、タプル操作要求の中継やコピーされたタプルの管理(クラスタ化)を行うだけではない。expireしたタプルのGC、タプルスペースのモニタおよび制御、仮想マシン自体を変えてしまうメタな操作なども行う。これらのプリミティブを、表2-3に示す。

ここでは、8個のプリミティブ[3]しか示していないが、新たな機能をtrapやevalを利用することによって定義し、アクティブタプルスペースの機能の一部として実行することも可能である。例えば、特定のタプルに対するinやoutなどの要求が発生した場合に、そのタプルのアクセスの履歴を表示するように、あらかじめtrapで指定しておくことができる。また、freezeを実行すると、in、rd、outなどのすべてのLindaのプリミティブの実行はサスペンドされ、タプルやタプルスペースのインスペクトが可能となる。しかし、プリミティブの実行ができなければ、インスペクトも行なえない。そこで、特権を持ったプリミティブの実行要求に限り、サスペンドを行わない。

primitive	タプルスペースに対する操作
freeze	タプルスペースへのアクセスを一時停止する
restart	タプルスペースへのアクセスを再開する
step	freezeされた状態から、1回だけタプルスペースアクセスを行なう
reset	タプルスペースを空にし、処理を再開する
trap	ある事象が起こった時に、実行する部分を定義する
dump	タプルスペースの内容をダンプする
filter	条件にマッチするタプルのリストを返す
log	過去のタプルスペースへのアクセスの履歴を表示する

表2-3 タプルスペースに対するプリミティブ

2.4 タプルオブジェクトと特殊タプル

従来のLindaモデルではタプルは単なるデータの列であったが、NUE-Lindaでは、近傍のタプルスペースへコピーされたタプルの制御や、デバッグなどの手がかりとして、タプル自体にもデバッグ用の情報を含めた属性を持たせることが必要となる。このため、タプルスペース内のタプルは、表2-4に示したような、インスタンス変数を持つオブジェクトと考えることができる。

[2]このout操作に、マルチキャストやブロードキャストを使用した場合、ネットワーク上に同期した多数のパケットが発生にし、一時的なふくそうが起こる可能性がある。このため、これをユニキャストの繰り返しによって実現することにしている。

[3]dump, filter, logなどの機能は、trapやevalを利用することによって定義できるので、必ずしも必要ないが、効率などの面からプリミティブとしてある。

インスタンス変数	意味
primary	プライマリタプルスペースを示す。
neighborhood	コピーを持つ近傍のタプルスペースを示すリスト
tuple	タプルそのもの
derivation	タプルの生成者(使用プロトコル、アドレス、プロセス番号など)
time-stamp	タプルの生成時間
time-to-live	タプルの生存時間
trap	トラップの定義
log	アクセスの履歴

表 2-4 タプルの持つインスタンス変数

タプルオブジェクトは、デバグに利用する情報のほか、タプルの生存時間を保持しており、システムがタプルのGCを行うために使用する。これによって、自動消滅するタプルが実現できる。従来のLindaでは、ガーベージとなったタプルはinする以外に、消去する手段が無いのに加え、そのマッチング条件がわからない限り消去も出来なかった。

NUE-Lindaには、このような自動消滅するタプル以外にも、システムが生成した特殊なタプルが存在する。例えば、タプルスペースに対する操作の特権を得るには、("privilege" x)という特殊なタプルをinする。このタプルは、タプルスペースに対する操作の排他制御のために、システムが生成したものである。また、("usage" "cpu" x)というタプルは、現在のCPUの負荷を知るためのものであり、rdしか許されない。

3 アクティブタプルスペースを実現する仮想マシン

アクティブタプルスペースは、受け取った操作要求に従って、タプルをタプルスペース上で操作し、その結果を要求元に返す仮想マシンとして定義できる。但し、このマシンは、タプル操作だけでなく、別プロセスを作り、その中で引数を評価し、その結果をタプルスペースに出すevalというメタな操作も実行する。更に、近傍の計算場を実現するための、操作要求の中継やタプルのコピー操作なども行う。

NUE-Lindaの仮想マシンは、ユーザプロセスの位置と操作の対象とするタプルスペースの位置とが完全に独立したものとなるような構造になっている。ユーザプロセスは、たとえ同一マシン上のタプルスペースをアクセスする場合にも、一旦、通信路を通じて仮想マシンへ要求を送り、その結果を通信路を通じて受け取る。これにより、同一マシン上のプロセスからの要求と、別のマシン上のユーザプロセスからの要求を区別することなく、処理を行うことができる。(図3-1)付録1は、TAOで記述した仮想マシンのインタプリタである。これは、処理の概要を示すために記述したものであり、実装する場合には、最適化が必要である。

NUE-Lindaでは、近傍の関係を実現する機能、モニタ機能、デバッグ機能のほか、マッチングの機能が強化されており、inやrdの引数、即ち、タプルに、正規表現を用いることも可能である。これにより、multiple-waitの機能が自然にinやrdに加わる。例えば、(in <"nue"|"elis"> *)を実行すれば、("nue" 1 2), ("elis" "tao" 3)などのように、"nue"または"elis"で始まるタプルとのマッチによって、プロセスのサスペンドが解ける。これにより、従来、Lindaでは、記述が複雑であったorの記述を簡単にすることができる。

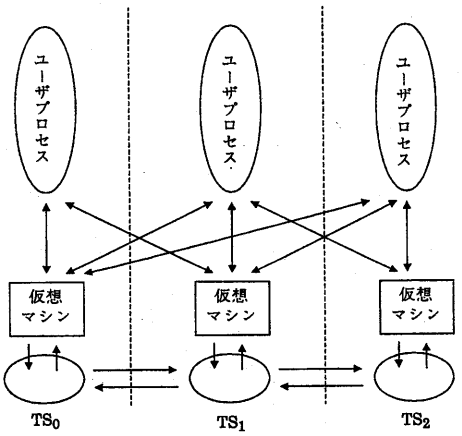

図 3-1 NUE-Linda interpreterの構造

4 NUE-Linda通信プロトコル

以上で述べてきた機能を、非均質システムにおいて実現するためには、共通(標準)の通信プロトコルを規定しなければならない。Lindaでは、これまで、このようなプロトコルの規定は行われていなかったため、非均質システムへの適応は困難であった。NUE-Lindaでは、従来のLindaモデルを包含するスーパーセットのプロトコルを規定している。このため、このプロトコルは、一般のLindaにも適用できるという利点を持つ。

NUE-Linda通信プロトコルの特徴は、以下の点である。

(1) 事実上の標準であるTCP/IPプロトコル群上のプロトコルとして規定したため、多くのマシンに実装可能である。しかし、TCP/IP以外のプロトコル群でも利用できるように下位プロトコルとの独立性が確保されている。

(2) ユーザプロセスとタプルスペースとの位置関係に独立性を持たせるため、たとえ、同一プロセッサ内にあるユーザプロセスがタプルスペースを操作する場合にも、通信プロトコルを使用するように設計されている。

(3) ネットワーク上におけるデータ表現法として、XDR[7]やASN.1[8]などの、さまざまなプロトコルの使用できる自由度がある。

(4) 同期したトラフィックによるネットワークの一時的な過負荷を避けるため、近傍のタプルスペースへのタプルのコピーなどにマルチキャストやブロードキャストを使用せず、極力、中継を行って、ネットワーク負荷の分散化を図っている。

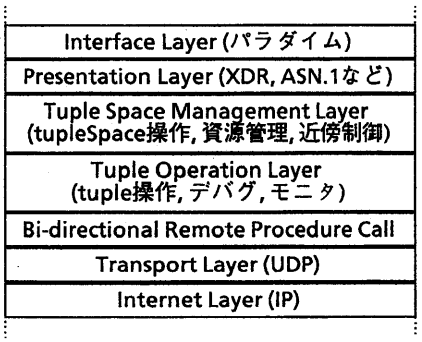

図 4-1 NUE-Linda Protocol Layer

NUE-Lindaプロトコルでは、Lindaのタプル操作が、トランザクションの性質を持つので、TCP/IPプロトコル群のなかで、高速で通信のオーバヘッドの少ないUDP(User Datagram Protocol)[10]を利用している。(図4-1)但し、UDPは、転送の誤りを回復しないので、NUE-Lindaプロトコルでは、簡単なBi-RPC(Bi-directional Remote Procedure Call)プロトコルで、誤りの検出と再転送による回復も規定している。タプルオペレーションレイヤは、Bi-RPCを利用して、タプルの操作を行う。近傍の計算場を実現するために、タプルのコピーや同期のための操作を行なうのは、その上のタプルスペース管理レイヤである。以下の節では、各レイヤにおけるプロトコルの動作について説明する。

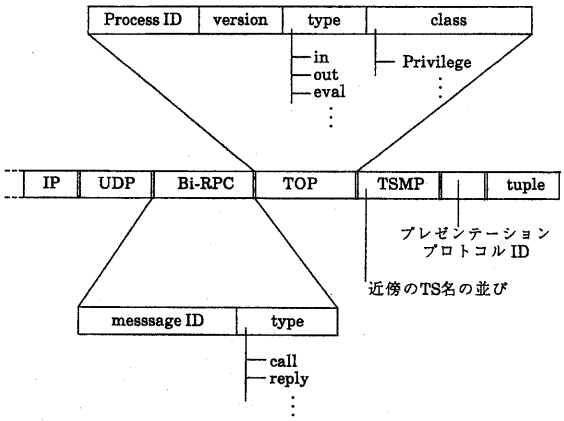

図 4-2 NUE-Linda Packet Format

4.1 Bi-RPC

図4-2に、NUE-Lindaのパケットフォーマットを示す。Bi-RPCでは、Callのパケットを送ったあと、そのmessage-IDに対するACK(受理を示すパケット)が送られてくるまで、監視する。あらかじめ決められた時間内にACKが送られてこなかった場合には、もとのパケットを再転送する。もし、IDが、もとのIDよりも大きい値を持つACKパケットが送られてきた場合には、もとのパケットは受理されたものと考える。Bi-RPCでは、ACKパケットにCallのパケットを乗せること(双方向にRemote Procedure Callを行なうこと)が可能で、その場合には、IDが1だけインクリメントされるからである。

4.2 タプルオペレーションレイヤ

このレイヤでは、inやoutなどの、タプル操作のための通信を行なう。このプロトコルをTOP(Tuple Operation Protocol)と呼ぶ。TOPでは、inなどのように、サスペンドが発生する可能性のある場合、操作要求とその結果は別のトランザクションとなる。即ち、一回のタプル操作は、2回のBi-RPCに分けられる場合がある。各パケットのTOPヘッダには、要求を出したプロセスのIDが入っているので、受け取った結果を、要求元のプロセスへ返すことができる。また、TOPには、サスペンド中のin要求をキャンセルするための機能も含まれている。これは、一般のユーザプロセスが使用するので

はなく、上位のタプルスペース管理レイヤが、近傍への同時 in 要求をキャンセルする場合などに使用する。

4.3 タプルスペース管理レイヤ

このレイヤで、最も重要な仕事は、近傍の計算場をサポートするためのタプル操作を行なうことである。このレイヤにおけるプロトコルを TSMP(Tuple Space Management Protocol) と呼ぶ。TSMP のパケットヘッダには、近傍に指定されたタプルスペースの名前もしくはネットワークアドレスが入っている。この部分を参照し、タプルを近傍のタプルスペースにコピーしたり、コピーされたタプルに対する排他制御を行なう。また、近傍に対して同時に in 要求を出し、どれかの要求が成功すれば、残りをキャンセルするのも、このレイヤの機能である。

4.4 プレゼンテーション層

プレゼンテーション層では、Linda システムで使われるデータ表現を規定する。ここでのデータ表現は一つに固定するのではなく、色々な方式が混在できるように、データ表現形式に関する情報を持つ。例えば、記号処理計算機 TAO/MacELIS II[9] 上の実現においては、独自のデータ表現方式と OSI の ASN.1 (Abstract Syntax Notaion 1) が混在している。

5 おわりに

本論文では、まず、NUE-Linda における近傍系の計算モデルを説明した。次に、近傍の計算場を実現するための核となる仮想マシン (NUE-Linda インタプリタ) の構造および、仮想マシンが使用する通信プロトコルについて述べた。現在、プロトタイプを TAO/MacElis II 上で作成しており、NUE-Linda の一部が動作している。今後、TAO/MacElis II 上の実装作業を更に進めると共に、ATMS や Genetic Algorithm を例題として NUE-Linda システムの評価を行なう予定である。

謝辞

本研究を進めるにあたって、NTT 基礎研究所 竹内郁雄リーダの提唱する “Connectics”(近傍系理論) に触発されるところが大でありました。ここに感謝いたします。

参考文献

[1] 竹内郁雄: ”Connectics(コネクティクス、近傍理論) の提案”, NTT 研究所における新水曜会資料、July, 1990

[2] 阪谷徹、野島久雄: ”ネットワークユーザ間の利用、操作に関する情報の伝達構造”、情報処理学会コンピュータネットワークのヒューマンウェアシンポジウム報告集、1989, pp.33-40

[3] Douglas Comer, Internetworking with TCP/IP, 1988, Prentice-Hall

[4] Nicholas Carriero and David Gelernter, LINDA IN CONTEXT, CACM, April 1989, pp.444-458

[5] Ikuo Takeuchi, Hiroshi G. Okuno and Nobuyasu Osato, A List Processing Language TAO with Multiple Programming Paradigm, New Generation Computing Ohmsha / Sprige-Verlag Vol.4 Num.4 1986, 401-444

[6] David Gelernter, Generative Communication in Linda, ACM TOPLAS, January 1985, 80-112.

[7] Sun Microsystems, XDR:External Data Representation standard, June, 1987.

[8] Information Processing - Open Systems Interconnection - Spacification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1). Internationl Organization for Standardization, 1987. International Standard 8825.

[9] 三上、村上、マルチプロセッサ Lisp マシン Mac ELIS II, 情報処理学会第 31 回プログラミングシンポジウム報告集、1990、pp.135-146

[10] J. Postel, User Datagram Protocol, RFC 768, Aug 1980

付録 1 TAO で記述した NUE-Linda インタプリタ (概要)

```
(de Active-TS (rcv xmit suspend-list &aux request x)
    (loop
     (!request (get-Linda-request rcv))    ;;; 操作要求オブジェクトを得る
     (select [request command]             ;;; コマンド解析 (trap はこの中で実行)
       (in-request                         ;;; in のリクエストの場合
        (cond
         ((!x (inp [request tuple]))       ;;; マッチするタプルがあるか?
          (if [x has-copy]                 ;;; 近傍のチェック
              (cond
               ((inp [x key] :neighborhood [x primary])     ;;; キーを取る
                (for i [x neighbor] (in x :neighborhood i)) ;;; コピー消去
                (send-Linda-reply x xmit) )                 ;;; タプルを返す
               (t                         ;;; キーが取れない時
                (out x) (send-Linda-reply 'suspend xmit request) ;;; undo して要求をサスペンド
                (push request suspend-list)) )              ;;; 待ち行列に入れる
              (send-Linda-reply x xmit request)))           ;;; 近傍がない
         (t                               ;;; マッチングに失敗
          (send-Linda-reply 'suspend xmit request)          ;;; 要求のサスペンド
          (push request suspend-list)) )) ;;; 待ち行列に入れる
       (out-request
        (if (!x (waiting? suspend-list request))            ;;; 待ちをチェック
            (or (for i x (send-Linda-request 'resume xmit (bind i request))
                  (if [i in?] (exit-for t) ))               ;;; リジューム処理
                (outp request))             ;;; rd のリジュームだけの場合
            (outp request))                 ;;; 何も待ちのなかった場合
        (send-Linda-reply 'ok xmit request))
       (rd-request ..............................)
       .
       .
       (multicast-out-request               ;;; 近傍が対象の out 要求
        (cond
         ([request primary]                 ;;; この要求がプライマリなら
          (out "key" (gensym)) (out request)                ;;; キーを生成
          (for i [request neighbor] (out request :neighborhood i))) ;;; コピー
         (t
          (out request)) )                  ;;; プライマリでなければ、単に out
        (send-Linda-reply 'ok xmit request))
       (multicast-in-request                ;;; 近傍が対象の out 要求
        (trap 'out [request tuple]          ;;; 最初の in が成功した時の trap で、
           (trap 'out [request tuple]       ;;; それ以降は戻す (undo する)trap
               (out request :neiborhood [request source]) (!request nil)))
        (for i [request neighbor] (eva (out (in request :neighbor i)))) ;;;TS 毎にプロセス
        (send-Linda-reply 'suspend xmit request)        ;;; 要求のサスペンド
        (push request suspend-list) )       ;;; 待ち行列に入れる
       (rd-request ..........)
       .
       .
       (t (send-Linda-reply 'unknown-RPC-command xmit request)) )))
```