マルチランデブチャネルの動的設定機構を持つ言語 LOTOS/M の提案

梅津 高朗[†] 寺島 芳樹[†] 安本 慶一[‡] 中田 明夫[†] 東野 輝夫[†] 谷口 健一[†]

[†] 大阪大学大学院基礎工学研究科情報数理系

[‡] 滋賀大学経済学部情報管理学科

本論文では，アドホックネットワークにおいて複数の移動可能エージェント間のマルチランデブチャネルの動的な設定が記述可能な言語 LOTOS/M を提案する．あるエージェントの対が互いに通信可能な状態にあるとき，LOTOS/M で導入した新機能：(1) 指定したゲートでの同期相手の募集，および (2) 募集されている同期に対する参加，により，指定したゲート (チャネル) を介した同期関係を動的に設定することができる．同じゲートを用いた同期募集を繰りかえすことにより，マルチランデブ (複数並行プロセス間で指定された条件が成立する時に同期通信によりデータ交換を行う機構) を行うためのチャネルを段階的に設定できる．あるエージェントが他と通信できない状態に陥った時には，そのエージェントだけが同期関係を解消し，他のエージェントと独立に動作できる．幾つかの例題の記述，実装実験を通して，典型的な無線モバイルアプリケーションが LOTOS/M により容易に記述でき，作成したコンパイラを用いて効率良く実装できることを確かめた．

Proposal of LOTOS/M: A Language with Dynamic Establishment of Multi-way Synchronization Channels among Mobile Agents

Umedu Takaaki[†], Yoshiki Terashima[†], Keiichi Yasumoto[‡], Akio Nakata[†], Teruo Higashino[†], and Kenichi Taniguchi[†]

[†] Dept. Info. and Math. Sci., Osaka Univ., Japan

[‡] Dept. Info. Proc. and Man., Shiga Univ., Japan

In this paper, we propose a new language called LOTOS/M which enables dynamic establishment of multi-way synchronization channels among multiple mobile agents on ad hoc networks, and show how it can be applied to designing wireless mobile applications. In LOTOS/M, when a pair of agents is in a state capable of communicating with each other, a synchronization relation on a given gate (channel) list can dynamically be assigned to them by a new facility of LOTOS/M: (i) advertisement for a synchronization peer on a gate list and (ii) participation in the advertised synchronization. The synchronization relation on the same gate list can also be assigned to multiple agents to establish a multi-way synchronization channel incrementally so that the agents can exchange data through the channel . When an agent goes in a state incapable of communication, a synchronization relation assigned to the agent is canceled and it can run independently of the others. Through some examples, we confirm that typical wireless mobile systems can be easily specified in LOTOS/M, and that they can be implemented efficiently with our compiler.

1 はじめに

近年の携帯端末および無線通信環境の発展・普及により，モバイルノード間の無線通信を伴うモバイルアプリケーションが注目されている．現在までに，ロケーションアウェアシステム [1] や，動的に構築したアドホックネットワーク [2] 上での電子会議システムなど，様々なアプリケーションが考案されつつある．

こういった無線モバイルアプリケーションでは，インターネットのような相手を特定した通信のみならず，モバイルノードの自由な移動に応じて，偶然互いの通信可能範囲に居合わせた相手と通信を行う機能や，無線範囲にいない通信相手との間に幾つかの中間ノードを介して通信路を動的に確保する機能 [2] などが必要になることが考えられ，これらの機能が容易に取り扱える言語および開発環境が望まれている．

通信プロトコル向けに開発された仕様記述言語 LOTOS[3] は，複数プロセス (イベント系列) 間の選択，並列，割込などの高度な制御機構に加え，複数並行プロセス間でイベントを同期実行しデータ交換を行なうためのマルチランデブと呼ばれる機構を持つ．マルチランデブの利用により，ブロードキャストや資源への排他制御などを含む分散システムを容易に記述できることが知られている．

しかし，従来の LOTOS には，プロセス (以下，モバイルノードで実行されるプロセスをエージェントと呼ぶ) 同士が互いの無線範囲に入り通信可能状態となったことを検知し，それらの間に動的にマルチランデブのためのチャネルを設定するための機能が用意されておらず，それゆえ，モバイルシステムの記述に適用するのは困難であった．

一方，プロセス代数に，互いに交信可能な範囲のエージェント間での動的なチャネル確立機能を追加した π 計算 [4] や，同様な機能を LOTOS に取り込んだ M-LOTOS[5] などでは，エージェント群が互いの通信可能範囲に入った時にのみ通信

を行う，という動作をうまく記述できるが，LOTOS の特徴であるマルチランデブがエージェント間に指定できない．

本論文では，複数モバイルエージェント間での動的なマルチランデブチャネルの確立・通信が記述可能な言語 LOTOS/M を提案する．LOTOS/M では，モバイルシステムは，互いに独立に動作するエージェントの集合として与えられる．任意の 2 つのエージェントが (無線範囲内に移動したなどで) 互いに通信可能な状態になった時のみチャネル (あるゲートでの同期関係) の確立を可能とする次の新しい構文：(i) あるゲートリストでの同期相手の募集，(ii) 同期募集に対する参加表明，を LOTOS/M に導入した．同期募集を行ったエージェントとそれへの参加表明を行ったエージェントの間には，指定されたゲートリストでの同期関係が割り当てられる．この際，同じゲートリストが割り当てられたエージェント群を新たな一つのエージェントとしてとらえ，別のエージェントを同じゲートリスト上に段階的に結合することによって，複数エージェント間にマルチランデブチャネルを確立できる．マルチランデブチャネルを確立したエージェント群を**エージェントグループ**と呼び，それらの間では，チャネルが解放されるまで，LOTOS のマルチランデブにより指定したゲートリスト上のイベントを同期実行できる．グループには，直接，物理的な通信ができないエージェント群が含まれてもよい (それらのエージェントは中間のエージェントを介して間接的に通信可能)．あるエージェントが移動などによって無線範囲から離脱した場合，結合している他のエージェント群との間の同期関係を解消し，以後独立に動作できるよう意味定義を行なった．

LOTOS/M では，各エージェントグループは，同期オペレータを節，エージェントを葉とする 2 分木 (**同期木**と呼ぶ) を形成するようセマンティクスが定義されているため，我々が従来から開発してきた標準 LOTOS 用のコンパイラ [6] の技術を用いて容易に実装できる．

幾つかの例題の記述，実装実験を通して，典型的な無線モバイルアプリケーションが LOTOS/M により容易に記述でき，作成したコンパイラを用いて効率良く実装できることを確かめた．

2 モバイルシステムの記述に必要な機能

2.1 仕様記述言語 LOTOS の概要

LOTOS[3] は ISO により開発された通信プロトコル向けの仕様記述言語であり，並列性を含むシステムの挙動を簡潔に表すための強力な構文を有する．LOTOS では，システムを幾つかの並列プロセスとして記述し，各プロセスの動作 (**動作式**) は，**イベント**と呼ばれるゲート (環境との相互作用点) を介したプロセス外部との相互作用 (値の入出力) の実行系列として定義される．

イベント間の実行順序を指定するため，連接 $(a;B)$，選択 $(B1[]B2)$，同期並列 $(B1|[G]|B2)$ [1]，非同期並列 $(B1|||B2)$，割込み $(B1[>\ B2)(B2$ は $B1$ に割り込むことができる)，逐次 $(B1 >> B2)(B2$ は $B1$ が正常終了したときのみ実行される) などのオペレータが任意の部分動作式間に指定される．特に，同期並列オペレータを使用することにより，複数のプロセスが指定されたゲート上のイベントを同時に実行しデータ交換を行なう，といった動作を記述することができる (マルチランデブと呼ばれる)．さらに，$[guard]->B$により，論理式 $guard$ が成立する時のみ動作式 Bを実行するよう制限したり，hide G in Bにより，動作式 Bでのみ用いる新しいゲートリスト G を新たに生成することを指定できる．

分散システムの複数ノード間にマルチランデブを指定することで，データのブロードキャストや資源への排他制御[2]など複雑な機構を含むシステムが簡潔に記述でき，さらには制約指向記述スタイル [7] による，ノード間の動作制約の段階的な追加などシステムの設計・保守上有用な機能が利用できることが知られており [6]，モバイルシステムのエージェント間においてもマルチランデブが使えることが望まれる．

図 1: 移動に伴う通信可能なエージェントの移り変わり

2.2 モビリティの記述に必要な機能

無線モバイルシステムでは，図 1に示すようなエージェントの移動により，互いに通信可能なエージェントの組合せが動的に変化するため，LOTOS でモデル化する際には以下の問題が生じる．

(1) エージェント群が (共通の無線範囲に居合わせることなどにより) 物理的に通信可能である時，またその時に限り動的に通信チャネルを割り当てるための手段がない．

(2) 多数のエージェントのうち，幾つのエージェントで，また，どのエージェントの組み合せで通信を行うのか，その組み合わせは無数にあり，LOTOS では基本的にそれらすべての組み合わせをあらかじめ記述しておく必要がある．

3 LOTOS/M の提案

上記の問題点を解決するため，本論文ではモバイルシステムの記述に適した言語 LOTOS/M を提案する．

3.1 LOTOS/M の定義

LOTOS/M で新たに拡張した構文とその意味を表 1に示す．LOTOS/M では，モバイルシステム全体を，$A := A_1 \mid A_2 \mid ... \mid A_n$のように，互いにどのように通信するのかがあらかじめ定まっていない独立なエージェント群の集合として与える．$A_1 \mid A_2 \mid ... \mid A_n$は，$|\{A_1,...,A_n\}$ とも表記され，その操作的意味は以下のように定義される．

$$\frac{A_i \xrightarrow{a} A_i',\quad a \in Act \cup \{sync, disc\}}{|\{A_1,...,A_n\} \xrightarrow{a} |\{A_1,...,A_i',...,A_n\}}$$

各エージェント A_iの動作は，新しい構文 "agent ... endagent" を用いて定義し，その動作式は "sync ... endsync" および標準の LOTOS のオペレータを用いて記述する (オペレータ "|" は用いてはならない)．

エージェント間のチャネルの確立　あるエージェントの組において，それらが物理的に通信可能な状態である時のみ，

[1] 以下では，並列オペレータが指定された動作式 $B1|[G]|B2$ について，ゲートリスト G での同期関係が部分動作式 $B1$ と $B2$ の間に指定されていると呼ぶ．

[2] $R|[G]|(C1|||C2|||...|||Cn)$ のように，「マルチランデブ」でなく 2 つのプロセス間の同期で十分な場合もある．

表 1: 新たに追加された構文とその機能

構文	機能
$A1 \mid A2 \mid ... \mid An$	n 個のエージェントが互いに独立に実行されていることの定義.
agent $A[G](E) := B$ endagent	各エージェント A の動作の宣言.
sync $!G : sid\ IO\ Guard$ in B endsync	ゲートリスト G を用いた同期相手の募集と, 同期相手が見つかった後に実行すべき動作式 Bの指定.
sync $?H : sid\ IO\ Guard$ in B endsync	同期募集への参加表明と, 参加表明が認められた時に実行すべき動作式 Bの指定. ただし, $B[G/H]$ のように, B中の仮ゲートリスト H は募集側エージェントから受け取ったゲートリスト G でリラベリングされる.
$disc(sid)$	$ID = sid$ で確立されたチャネル (同期関係) の切断.
$g!P; ... \ [[g]]_{sid}\ g?Q : process; ...$	エージェント間でのプロセス名の送受信.

チャネルを割り当てできるように, LOTOS/M は次の特別な動作を提供する. (i) 同期相手の募集のための動作 (sync $!G : sid\ IO\ Guard$ in $B1$ endsync), (ii) 同期募集への参加表明のための動作 (sync $?H : sid\ IO\ Guard$ in $B2$ endsync).

上記の動作において, G および Hはゲートリストを表し, ": sid" は確立された同期関係に対する ID を記憶するための変数を表す (後に述べる特別な動作 $disc(sid)$ を用いて既に確立されている同期関係を自発的に解消するために使われる). IOは, チャネル確立時に交換される値の入出力のリスト ($!E1?x1?x2$ など) であり, $Guard$ は, チャネル確立のために成立すべき条件を表す論理式であり $[f(c1, c2, ..., x1, x2, ...)]$ のように表記される (定数 $c1, c2, ...$, および IO中の変数 $x1, x2, ...$がパラメタとして指定可能). なお, ": sid", "IO", "$Guard$" は省略しても良い.

もし, 次の条件があるエージェントの対に対して成立するなら, ゲートリスト G に関する同期関係がそれらのエージェントの間に割り当てられる. 同期関係が割り当てられることを, **結合**とも呼ぶ.

- あるエージェント $\mathcal{A}_1$が $sync$ 動作 "$B1$:=sync $!G$: $sid\ IO1\ Guard1$ in $B1'$ endsync" を実行可能であり (**ホストエージェント**と呼ぶ), 他のエージェント $\mathcal{A}_2$ が "$B2$:=sync $?H$: $sid\ IO2\ Guard2$ in $B2'$ endsync" を実行可能 (**ゲストエージェント**と呼ぶ) である.
- G と Hのゲート数が同じである.
- $IO1$ と $IO2$ のパラメタ数が同じであり, かつ, それぞれのパラメタの対が入力 ($?x$) と出力 ($!E$) から構成され, それらの型が一致する.
- 各出力値を入力変数に代入した後の $Guard1$ と $Guard2$ が共に成立する (対応するパラメタの対が$?x$ と$!E$の場合, 式 Eの値が変数 x に代入される).
- エージェント $\mathcal{A}_1$と $\mathcal{A}_2$が互いに物理的に通信可能な状態にある (互いの無線範囲にいる).

上記の条件が全て成立する時, 2 つのエージェントには同期関係が割り当てられ, 以後, $\mathcal{A}_1[B1'/B1]\ \ |[G]|_{sid}\ \mathcal{A}_2[B2'[G/H]/B2]$ のように振舞う. ここで, $|[G]|_{sid}$は**拡張並列オペレータ**と呼び, $disc(sid)$ アクションによりそのオペランドが離脱可能である以外は LOTOS の同期並列オペレータと同じ意味を持つ. $\mathcal{A}[B'/B]$ は, エージェント $\mathcal{A}$ 全体の動作式に対し, その部分動作式 Bを B'で置き換えたものであり, また, $B2[G/H]$ は, 動作式 $B2$ に出現する H の各ゲートを, 対応する G のゲートに置き換えた動作式である.

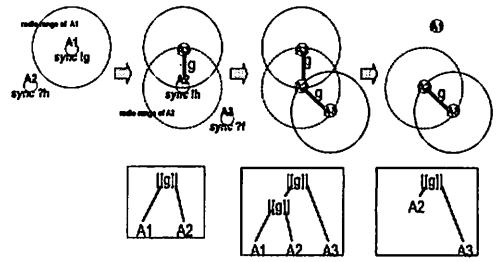

図 2: エージェントの接近, 離脱に伴うチャネルの生成, 消滅

なお, 同期募集 $sync\ !G$ で使われるゲートリスト G は LOTOS の hide オペレータを用いて生成されたものか, $sync\ ?H$ で受け取ったゲートリストに限り, エージェントの環境とのゲート (ユーザとのインタフェースなど) を用いてはならない.

結合したエージェントは新たな一つのエージェントとして扱い, さらなる $sync\ !G$ アクションを実行することで, 他のエージェントと段階的に結合できる. なんらかの同期関係で結合したエージェント群を**エージェントグループ**と呼ぶ. 例として次に示す 3 つのエージェントが段階的に結合する様子を図 2 に示す.

```
A1 | A2 | A3
where
  A1:= sync !{g} in g; stop endsync
  A2:= sync ?{h} in
         sync !{h} in h; stop endsync
       endsync
  A3:= sync ?{f} in f; stop endsync
```

同期関係の解消

各エージェントグループはチャネル切断動作により幾つかのエージェント (エージェントグループ) 群に分離することができる. チャネル切断によるエージェントの動作を以下のように定義する.

表 2: LOTOS/M におけるエージェント結合，離脱のための推論規則

Agent-Join

$$\frac{A_i \xrightarrow{sync!G} A_i',\quad A_j \xrightarrow{sync?X} A_j',\quad G=\{g_1,\ldots,g_k\},\quad X=\{x_1,\ldots,x_k\}}{|\{A_1,\ldots,A_k\} \xrightarrow{sync!G} (A_i' \,|[G]|_{\{sid\}}\, A_j'[G/X] \;\mid\; |\{A_1,\ldots,A_k\}-\{A_i,A_j\})} \quad (1)$$

ここで，A_iは各エージェントあるいはエージェントグループの動作式である．$B[G/X]$ は Bの仮ゲートパラメータ群 $X=\{x_1,\ldots,x_k\}$ の全ての自由な出現を対応する実ゲートパラメータ群 $G=\{g_1,\ldots,g_k\}$ で置き換えることを表す．また，$sync$ 動作にガード式が設定されている場合は省略するが，推論規則の前提条件に指定された条件を付け加えることで対処できる．

Agent-Leave-1

$$\frac{A_1 \xrightarrow{disc(sid)} A_1'}{[A_1 \,|[G]|_{sid}\, A_2 \xrightarrow{disc(sid)} A_2, A_1', matched]} \quad (2)$$

Agent-Leave-2

$$\frac{A_1 \xrightarrow{disc(sid)} A_1',\quad sid <> sid'}{[A_1 \,|[G]|_{sid'}\, A_2 \xrightarrow{disc(sid)} A_2, A_1', unmatched]} \quad (3)$$

Agent-Leave-3

$$\frac{[A_1 \xrightarrow{disc(sid)} A_1', A_3, flag],\quad sid <> sid'}{[A_1 \,|[G]|_{sid'}\, A_2 \xrightarrow{disc(sid)} A_1' \,|[G]|_{sid'}\, A_2, A_3, flag]} \quad (4)$$

Agent-Leave-4

$$\frac{[A_1 \xrightarrow{disc(sid)} A_1', A_3, unmatched]}{[A_1 \,|[G]|_{sid}\, A_2 \xrightarrow{disc(sid)} A_1' \,|[G]|_{sid}\, A_2, A_3, matched]} \quad (5)$$

Agent-Leave-5

$$\frac{[A \xrightarrow{disc(sid)} A', A'', matched]}{A \xrightarrow{disc(sid)} A' \mid A''} \quad (6)$$

ここで，$A, A', A_1, A_2, \ldots$はエージェントあるいはエージェントグループの動作式を表す．補助ターム$[A \xrightarrow{disc(sid)} A', A'', flag]$は，$A''$, $flag$という付加情報がついている以外は遷移関係$A \xrightarrow{disc(sid)} A'$と全く同じである．$A''$は離脱するエージェントの動作式，$flag \in \{matched, unmatched\}$ は sid に対応する離脱対象の並列演算子が見つかったか否かを表すフラグである．

- あるエージェントが，属しているエージェントグループのどのエージェントとも通信不可能な状態になった時，そのエージェントの対応する部分動作式 sync ... in B endsync は $exit$ に変わる (パッシブなチャネル切断と呼ぶ)．
- あるエージェントが 部分動作式sync ... in B endsync, $B = \ldots; disc(sid); B'$中で，$disc(sid)$ 動作を実行した時，その部分動作式は以後 B'のように振舞う (アクティブなチャネル切断と呼ぶ)．ただし，動作式 B'は，$sync$ 動作により他のエージェントから割り当てられたゲートを含んではならない．

パッシブな場合とアクティブな場合のどちらも $disc(sid)$ 動作で扱えるようにするため，$disc$ 動作を含まない sync ... in B ensync については，sync ... in $(B >> disc(sid); exit)[> disc(sid); exit$ endsync と定義されているものと仮定する．

エージェント間でのプロセス名の交換　LOTOS/M では，高階π計算 [8] にならい，LOTOS で扱えるデータタイプに加え，プロセス名をデータとしてマルチランデブにより送受信可能なよう拡張されている．これにより，各モバイルエージェントが必要に応じてプログラムをダウンロードし実行するといった動作を含むシステムを記述できる．LOTOS/M では，プロセス名と付随するゲートおよび値パラメタ群のみをエージェント間で交換することを記述するため，このためのセマンティクスの拡張は不要である (他のデータパラメタの交換と同様のため)．プロセスの動作式をどのようにエージェント間で交換するかは，実装上の問題として扱う (Java 言語では比較的簡単に実現できる)．

3.2 LOTOS/M のセマンティクス

LOTOS/M で拡張された構文 $sync$, $disc$ 動作について，以下のような意味定義を行い，そのための推論規則を定義した (表 2)．

- 各 sync 動作の実行に対して id を発行し，id に対応する disc 動作ができるようにする (図 3 例 1)．
- disc 動作後のエージェントは，動作式のいかなる部分式として動作していても，独立したエージェントに戻す (図 3 例 2)．

エージェントの結合　ゲートリスト G での募集エージェント，ゲートリスト Xでの参加エージェントの組が存在するとき，推論規則 Agent-Join により，これらは結合して G での同期関係を持つことができる．Xはゲートが格納される変数であり，結合後は G と等価になる．この同期関係には id が付随され，次の離脱時の推論規則において，指定 id を持つ同期関係が解消される．

エージェントの離脱　あるエージェントが $disc(id)$ 動作を実行すると，そのエージェントは属しているエージェントグループから分離されなければならない (このことを離脱と呼ぶ)．離脱時には，同期関係が指定されている動作式に対し，

表 3: 結合，離脱時の意味定義

Example 1

$A2 \xrightarrow{sync?F} A2'$, $A3 \xrightarrow{sync!F} A3' \xrightarrow{sync?G} A3''$の時，$A1 \xrightarrow{sync!G} A1'$,

$A1 \mid A2 \mid A3$

$\xrightarrow{sync!F!1} A1 \mid (A2' \;|[F]|_1\; A3')$

$\xrightarrow{sync!G!2} A1' \;|[G]|_2\; (A2' \;|[F]|_1\; A3'')$

Example 2

$A2' \xrightarrow{disc(1)} A2''$, $A1' \xrightarrow{disc(2)} A1''$の時，

$(A1' \;|[G]|_2\; (A2' \;|[F]|_1\; A3''))$

$\xrightarrow{disc(1)} (A1' \;|[G]|_2\; A3'') \mid A2''$

$\xrightarrow{disc(2)} A1'' \mid A3'' \mid A2''$

表 4: 離脱時の推論規則の適用

$A_1' \;|[G]|_2\; (A_2' \;|[F]|_1\; A_3'')$ に対し，

- A_2 が $id = 1$ の拡張並列オペレータに対し離脱する場合．

  $A_2' \xrightarrow{disc(1)} A_2''$の時，

  Agent-Leave-1 を適用すると，

  $[A_2' \;|[F]|_1\; A3'' \xrightarrow{disc(1)} A3'', A_2'', matched]$

  Agent-Leave-3 を適用すると，

  $[A1' \;|[G]|_2\; (A_2' \;|[G]|_1\; A3'') \xrightarrow{disc(1)} (A1' \;|[G]|_2\; A3''), A_2'', matched]$

  Agent-Leave-5 を適用すると，

  $A1' \;|[G]|_2\; (A_2' \;|[G]|_1\; A3'') \xrightarrow{disc(1)} (A1' \;|[G]|_2\; A3'') \mid A_2''$

- A_2 が $id = 2$ の拡張並列オペレータに対し離脱する場合．

  When $A_2' \xrightarrow{disc(2)} A_2''$,

  Agent-Leave-2 を適用すると，

  $[A_2' \;|[F]|_1\; A_3'' \xrightarrow{disc(2)} A_3'', A_2'', unmatched]$

  Agent-Leave-4 を適用すると，

  $[A_1' \;|[G]|_2\; (A_2' \;|[F]|_1\; A_3'') \xrightarrow{disc(2)} A_1' \;|[G]|_2\; A_3'', A_2'', matched]$

  Agent-Leave-5 を適用すると，

  $A_1' \;|[G]|_2\; (A_2' \;|[F]|_1\; A_3'')$

  $\xrightarrow{disc(2)} (A_1' \;|[G]|_2\; A_3'') \mid A_2''$

id に対応する離脱対象の並列演算子が見つかったか否かに応じて推論規則を適用していく．

動作式 $A_1'|[G]|_2(A_2'|[F]|_1 A_3'')$ において，エージェント A_2 が離脱する場合での推論規則の適用を表 4に示す．id が最初から match して見付かった場合は Agent-Leave-1，そうでなければ Agent-Leave-2 を適用し，離脱エージェント動作式を分離 (補助ターム中 B'') する．Agent-Leave-3 により離脱する動作式をオペレータの一つ外側へと抜き出していく (離脱対象の並列演算子が見付かったか否か状態をフラグで保持) が，その際，対応する id を持つオペレータを見付けた時は，Agent-Leave-4 を適用し，その旨をフラグにセットする．最終的に Agent-Leave-5 を適用することで，対象とするエージェントを離脱させる．

表 2の推論規則により，与えられた LOTOS/M 仕様から，対応する LTS(ラベル付き遷移システム) を構成できる．ここで，LTS の各ノードはエージェントの現在の動作式の組 $(A1 \mid (A2' \;|[F]|_1\; A3')$ など) に相当し，各ラベルは $sync$ 動作，$disc$ 動作，イベントのいずれかに相当する．

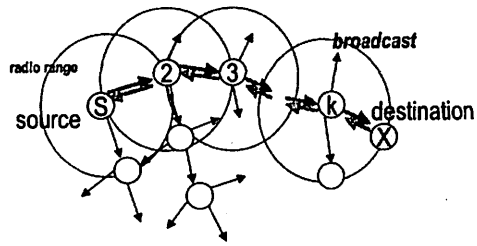

図 3: アドホックネットワークでの経路探索

4 モバイルシステムの記述例

無線アドホックネットワークでは，幾つかの途中エージェントを介した無線範囲の通信を繰り返すことで，無線範囲を超えた任意のエージェント間で互いにデータ交換できる．ここでは，Dynamic Source Routing [2] に基づき，あるエージェントから別のエージェントへの経路を見つけるプロトコルを LOTOS/M で記述する．

このプロトコルでは，図 3に示すように，あるエージェント (ソースエージェント) が他のエージェントへの経路を求めたいとき，その要求を無線範囲内にブロードキャストし，それを受け取ったエージェントは，その要求を再ブロードキャストする．このようなフラッディングを目的エージェントに到達するまで繰り返す．経路探索要求メッセージには，ソースから現在のエージェントに到達するまでの経路を含める (各中間エージェントは要求を転送する際に，自分の ID を変数 route_record の末尾に加える)．目的エージェントに到達した時，そのメッセージには，ソースからの経路がルートレコードに記録されているため，その情報を経路の逆順にソースエージェントに返信する．

エージェントの動作は以下のいずれかである．

(1) ソースノードでの，目的エージェントの ID の入力 ($u?dest\_id$) と経路探索要求のブロードキャスト
(2) 中間ノードでの，経路探索要求の受け取りとその要求の転送 (再ブロードキャスト)
(3) 目的ノードでの，経路探索要求の受け取りと発見経路の返信
(4) 中間ノードでの，発見経路返信の受け取りとソースノードへの転送
(5) ソースノードでの発見経路返信の受け取りとユーザへの提供 ($u!route$)

無線範囲へのデータのブロードキャストは，同期募集 ($sync\ !\{b\}$) を一定時間繰り返すことによって，無線範囲の複数エージェントの同期への参加を募った後，マルチランデブにより一度に送信する．ブロードキャストを行うサブプロセス $Broadcast$ の記述例を以下に示す．

```
process Broadcast[b](msgtype, dest_id, data): exit:=
  sync !{b}:sid!msgtype in
    [not(TimerExpired())]- > Broadcast(dest_id, msgtype, data)
  [] [TimerExpired()]- > b!msgtype!dest_id!data; exit
  endsync
endproc
```

(ここで，$TimerExpired()$ はあらかじめセットしておいた時間が経過したかどうかを表す関数)

エージェントの動作のうち，例えば，(2) は，現在の経路情報に自分が含まれない時のみ ($(not(included(route\_record, my\_id))$)，Search メッ セージを転送するように記述す

表 5: Path finding protocol based on Dynamic Source Routing described in LOTOS/M

```
agent DSR[u](my_id): noexit:=
hide b in
(
  (* (1) 目的ノードの ID の入力と経路探索要求の送信 *)
  u?dest_id:int;
  Broadcast(Search,dest_id,{my_id})
[] (* (2) 経路探索要求の転送 *)
 sync?{a}:sid?msg[msg=Search] in
  a!Search?dest_id?route_record
   [(dest_id<>my_id) and not(included(route_record,my_id))]);
  disc(sid);
  Broadcast(Search, dest_id, route_record+{my_id})
[] (* (3) 経路探索要求が目的ノードに到達した時の動作 *)
  a!Search?dest_id?route_record[dest_id = my_id];
  disc(sid);
  (sync !{b}!Return!last(route_record) in
    b!Return!route_record+{my_id}!route_record; exit
   endsync
  )
 endsync
[] (* (4) 経路情報の転送 *)
 sync?{a}:sid?msg?id[msg=Return and id=my_id] in
    a!Return?route?return_path
     [included(return_path,my_id) and return_path <> my_id];
    disc(sid);
    (sync !b!Return!last(return_path-{my_id}) in
      b!Return!route!return_path-{my_id}; exit
     endsync
    )
  [] (* (5) ソースノードが経路情報を受け取った時の動作 *)
    a!Return?route?return_path[return_path={my_id}];
    u!route;
 endsync
) >> DSR[u](my_id)
endagent
```

included(list,item) と last(list) は抽象データ型関数であり，前者は item が list に含まれるかどうか，後者は list の最後の要素を返す [illegible] に対してのみ転送を行いたいため，*sync* アクションの IO パラメタとしてそのノードの ID を指定し，返送パス上の隣り合ったエージェント間でのみ同期関係が割り当てられるようにする (*sync* !{*b*}!*Return*!*last*(*route_record*) and *sync* ?{*a*}?*msg*?*id*[*msg* = *Return and id* = *my_id*]).

各エージェントの動作全体の記述例を表 5に示す.

5 LOTOS/M コンパイラの性能

我々は，文献 [9] において，与えられた LOTOS/M 仕様をエージェントの動作を実現する Java プログラム群として実装するためのコンパイラを開発している．本コンパイラは，モバイルシステムにおける実行制御部分を LOTOS/M によりモデル化し実装することを目的としているため，仕様内で用いられる抽象データ型関数群 (LOTOS では代数的言語 ACT ONE で記述される) は適当な Java のクラスライブラリに対するメソッド呼出しにて実現できると仮定している.

本コンパイラにより生成された Java プログラム群を用いて，(1) チャネル確立の時間，(2) 確立されたチャネルを通した各イベントの同期実行の時間，(3) 確立されたチャネル上でのデータ転送レートについて計測を行った．実験環境として，Pentium 133MHz から PentiumII 266MHz を搭載した数台のノート PC と IEEE 802.11b の無線 LAN を用いた.

(1) に関しては，2 つのエージェント間のチャネル確立ごとに約 2.6ms かかることがわかった．4 つのエージェント間にマルチランデブチャネルを確立するには，単純にはその 4 倍の 10.4ms 程度かかることになる．(2) については，マルチランデブを行なうエージェントの数が 2〜4 の時，4.9〜7.7ms となった．また，(3) については，2 つのエージェント間のデータ転送レートは約 3.6Mbps となった (詳細は文献 [9] 参照).

6 おわりに

本論文では，無線モバイルアプリケーションの記述・実装に適した言語 LOTOS/M を提案し，その適用可能性について述べた．LOTOS/M では，互いの無線範囲に偶然居合わせたエージェント間で動的にチャネルを確立し，マルチランデブによる通信を行うことができる．また，無線範囲からの離脱に伴いエージェントが離脱しマルチランデブの構成メンバが動的に変わるようなケースをうまくモデル化できる．また，幾つかのアプリケーションの記述，実装実験から，提案手法が無線モバイルアプリケーションの記述・実装に十分適用可能であることを確認した.

参考文献

[1] Hodes, T.D., Katz, R.H., Schreiber, E.S. and Rowe, L.: Composable Ad-hoc Mobile Services for Universal Interaction, *Proc. of Mobile Computing and Networking(MOBICOM'97)* (1997).

[2] Johnson, D. B., Maltz, D. A., Hu., Y. C. and Jetcheva, J. G. : The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks, *IETF Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-dsr04.txt* (2000).

[3] ISO : Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, *ISO 8807* (1989).

[4] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes: Parts I & II, *Information and Computation 100*, pp. 1– 77 (1992).

[5] Najm, E., Stefani, J.B. and Fevrier, A.: Towards a Mobile LOTOS, *Proc. of 8th IFIP Intl. Conf. on Formal Description Techniques (FORTE'95)* (1995).

[6] Yasumoto, K., Higashino, T. and Taniguchi, K.: A comipler to implement LOTOS specifications in distributed environments, *Computer Networks*, Vol. 36, No.2-3, pp. 291–310 (2001).

[7] Vissers, C. A., Scollo, G. and Sinderen, M. v.: Architecture and Specification Style in Formal Descriptions of Distributed Systems, *Proc. 8th Int. Conf. on Protocol Specification, Testing, and Verification (PSTV'88)*, pp. 189 – 204 (1988).

[8] Sangiorgi, D.: From π-calculus to Higher-Order π-calculus — and back, *Proc. of Theory and Practice of Software Development (TAPSOFT'93)*, Lecture Notes in Computer Science Vol. 668, pp. 151 – 166 (1993).

[9] 寺島, 安本, 中田, 東野, 谷口: 並行モバイルエージェント間でのマルチランデブチャネルの動的設定が記述可能な言語とその実装, 情処研報 2001-DPS-102, pp. 37-42 (2001).