Vol.2014-DPS-158 No.38
Vol.2014-CSEC-64 No.38
2014/3/7

無線センサネットワークにおける遠隔反応可能なモバイルエージェントモデル

周 力陽[1,a)]　横田 裕介[2]　大久保 英嗣[2]

概要：本稿では，無線センサノード向けのモバイルエージェント（Mobile Agent, MA）型ミドルウェアにおける遠隔反応（Remote Reaction, RR）機能の導入を提案する．無線センサネットワーク（Wireless Sensor Network, WSN）では，ロバスト性や，通信の信頼性が強く要求される場合がある．この要求に対し，タプルスペース（Tuple Space, TS）を備えた MA 型ミドルウェアの Agilla が開発されたが，ノードの通信量・メモリバッファの制限のため，MA は，ローカルイベントのみに反応するように設計されている．そのため，ユーザのシナリオによっては，通信レイテンシ・計算負荷・通信量などの増大の問題が発生する可能性がある．これらの問題を解決するための手段として，MA 型ミドルウェアにおける遠隔反応モデルを提案する．MA が Remote Reaction(RR) 情報を隣接ノードに登録し，隣接ノードはイベントが発生した場合，RR 登録ノードを通知する．通知を受けたノードは，RR を発火させ，RR 登録 MA に対し，適切な割込み処理を実行させる．これにより，ミドルウェアの遠隔反応レイテンシの問題と，物体追跡アプリケーションに対する不適応性やフォールトトレランス性の低さなどの問題を解決する．

1. はじめに

無線センサネットワーク（WSN）は，複数の無線通信機能を持つセンサノードを空間に散在させ，それらが協調して動作することで，環境や物理的状況に関するデータを収集することを可能とするシステムである．通常の無線ネットワークと比べ，WSN は更に厳しい制約が課せられている．広域にわたって実世界の情報を収集するネットワークであるため，その要求により，センサノードのサイズは，一般的な無線ホストより，更に小さくなり，また生産コストは最小限まで削減されなければならない．従って，センサノードの性能と資源が大きく制限され，計算性能と通信能力は従来の無線ホストより低いという制約の下で，無線通信の信頼性を保持しなければならない．一方，センサノードは常に厳しい自然環境にさらされるため，故障や，通信障害は頻繁に発生する．このようなノード上の障害が複数発生しても，全体としてのセンサデータ収集作業自体は可能な範囲で正常に継続することも，WSN における重要な課題の一つである．

これらの要求に答えるため，センサノード上において，Tuple Space（TS）を情報共有手段として利用する Mobile Agent（MA）型ミドルウェアの導入が提案されている．MA は，あるノード上で実行されるプロセスだが，実行を一時停止し，他のノードへ移動して実行を継続する能力を持っている．ノードに多数の障害が発生したり，ホストとの通信が不可能な状態になっても，MA が適切なノードに移動し，任務を継続することができる．更に，ノードの物理的移動の代わりに，MA の論理的移動を用いて，物体追跡アプリケーションを開発すれば，ハードウェアの開発・保守費用も低く抑えることが期待できる．

TS は，イエール大学で開発された Linda という協調言語において提案された，並列処理に利用されるデータ共有の仕組みである．この仕組みは，無線ネットワークの通信不安定性に対する有効な対応手段だと気付き，セントルイスワシントン大学の研究者たちが TS をモバイルネットワークに導入し，LIME というミドルウェアが誕生した [1], [2]. 同大学の研究グループは，更に LIME の WSN 向けバージョンである TeenyLIME を開発したが，TeenyLIME は，TS を保有していながら，LIME の重要な特性である MA の機能を放棄した [3].

同グループは，他に Agilla というセンサノードミドルウェアも開発した．Agilla は LIME と同じく，TS・MA を共に備えているが，ノードのトランザクション量・メモリバッファの制限のため，MA は，異なるノードに位置する MA が出した Tuple に一切反応しない [4]．このシステムにおいて，異なるノードに位置する MA を協調動作させた

[1] 立命館大学大学院情報理工学研究科
[2] 立命館大学情報理工学部
[a)] zhou@sol.cs.ritsumei.ac.jp

場合，実現手法により，通信レイテンシ・計算負荷増大・通信量増大などの問題が発生する可能性がある．

本稿は，これらの既存のセンサノード向けミドルウェアの持つ課題に対し，無線センサノードのモバイルエージェント（MA）型ミドルウェアに遠隔反応（Remote Reaction, RR）機能を導入することを提案する．導入された RR モデルは，LIME・TeenyLIME と同様に，MA が RR 情報を隣接ノードに登録し，隣接ノードはイベントが発生した場合，RR 登録ノードを通知する．通知を受けたノードは，RR を発火させ，RR 登録 MA に対し，適切な割込み処理を実行させる．

以下，本稿では，第 2 章において関連研究である LIME・TeenyLIME と，Agilla を紹介し，Agilla における問題を簡単に分析する．第 3 章では，モデルの設計について述べ，第 4 章でシミュレータ上の実装について述べる．第 5 章では評価を行い，第 6 章で本研究のまとめと今後の課題について述べる．

2. 関連研究と課題

2.1 LIME・TeenyLIME

LIME（Linda In Mobile Environments）は，セントルイスワシントン大学が開発した，Linda モデルに基づくモバイルアプリケーション向けミドルウェアである．その最大の特徴は，MA と TS を共に備えていることである．ユーザが開発したアプリケーションは，コンパイラにより MA に変換され，モバイルホスト（Mobile Host，MH）上で実行される．MA であるアプリケーションは，自身の状態とデータ，及び実行コードを完全に他の MH に移動し，適切に実行を継続することが可能である．

この特徴により，LIME で実現したアプリケーションは以下の点で優れていると考えられる．

- 並列処理能力
  多数の MA が，様々な種類の MH 上で非同期実行し，協調動作することが可能である．この並列処理能力は，WSN ノードの計算性能の低さに対する，良い解決策にもなる．
- 動的な適応性
  MA は，実行環境の変化を感知することが可能であり，その変化に対し，適切に対応することも可能である．
- 高いフォールトトレランス性
  障害が発生したり，通信が不安定な MH を避けて，MA は他の MH に移動し，実行を継続することが可能である．
- 物体追跡の容易さ
  MH の物理的な移動の代わりに，MA の論理的な移動で，実在する目標を追跡することによって，自動追跡機能を備える MH の開発が不要となり，開発コストを下げることができる．

一方，LIME は，MA 間通信手段として，Transiently Shared Tuple Space（TSTS）を提供する．MA は，転送したい情報を，Tuple と呼ぶ配列に保存し，MA から MH のローカル保存空間（ローカル TS）に保存する．複数の MH が相互通信可能になった場合，MH は互いのローカル TS を共有し，TSTS を構成する．MA は，その TSTS に保存するすべての Tuple を受け取ることが可能であり，MH が一時的に他の MH と通信できなくなった場合，通信不能となった MH が保有する Tuple は共有できなくなる．

MA は，他の MA と通信する場合，送信情報を Tuple に変換し，ローカル TS に保存する．もし受信 MA が存在する MH（受信 MH）が，送信 MH と通信できる場合，LIME は Tuple を受信 MH に配布する．受信 MA は，ユーザが指定するときに TS 読込指令を実行すれば，ローカル TS から Tuple を受け取ることができる．この仕組みによって，MA の物理アドレスは不要となり，無効なアドレスへの通信を回避することが可能になる．

更に，LIME は，TS に基づく反応モデルを提供する．MA は，他の MA から発生するイベント（イベント Tuple を TS に出す形式）に反応することが可能である．MA は，反応したいイベント形式 (例えば Tuple t)・イベント発生 MA の id などの情報を Remote Reaction（RR）にまとめ，MA が現在位置する TSTS の構成する全ての MH に登録する．TSTS に位置するある MA が，t に合致する Tuple を TSTS に出す場合，RR を発火させ，RR 登録 MA に通知する．

TeenyLIME は, 同大学の研究グループが開発した, LIME の WSN 向け応用を意図したミドルウェアである．通常のモバイル端末に比べ，センサノードの資源と性能は極めて制限されるため，TeenyLIME は LIME より簡単な仕組みを持っている．

LIME の根本的な特徴である TSTS を，TeenyLIME は引き継いでいるが，1 ホップの距離を持つセンサノード同士でのみ構成可能としている．反応モデルについては，LIME の二種類の TS で反応精度を高める仕組みを放棄し，一種類の TS で一般の通信と reaction 通信を兼用する．

もう一つの差異は，MA 管理による計算量が大き過ぎるとの判断により，TeenyLIME は，MA の仕組みを放棄したことである．したがって，MA の特性による，アプリケーションの論理的な移動が不可能となった．また，MA によって実現されるフォールトトレランス性も無くなった．

2.2 Agilla

Agilla[4] は，同グループが開発した，MA 型 WSN ミドルウェアである．具体的なモデルは，図 1 に示すように，TeenyLIME より，LIME のモデルに近い．

LIME と同様，並列処理・移動物体追跡問題などを解決するため，MA を導入した．しかし，MA はあまりにも複

雑な仕組みで，MA の管理・プロセス実行・移動により計算量が大きくなるため，資源と性能が極めて制限されたセンサノードに対しては，より簡単な仕組みが必要である．そのため，ノード上で実行する MA の数を限定し，MA 個数の超過による計算量増大を避ける．一方，MA の開発は LIME と異なり，Java ではなく，独自のアセンブリ類似言語で実現する．この言語によってコンパイルされたコードは，メモリ指定・分配がより簡単になり，指令もシンプルになるため，コンパイルされたコードの実行が容易になる．

Agilla は，LIME の中核的な仕組みである TS も引き継いだが，TSTS を維持すると，頻繁にノードを多数スキャンする必要があり，TS 指令の不可分性と TS の一致性を保証するため，大量のトランザクションが発生する．これが WSN に対して過大な負荷を与えるため，TSTS を放棄し，代わりに，隣接ノードのアドレスを記載する Neighbor List を通し，ユニキャスト遠隔 TS 指令を提供する．一方，イベント反応については，RR も放棄し，MA は，位置しているノードのローカル TS に存在する Tuple のみに反応する．

2.3 Agilla の持つ課題

前節で述べた Agilla の仕組みには若干の問題が存在すると考えられる．ここでは，その問題について簡単に述べる．

2.3.1 ループ動作 MA の反応レイテンシ

前述の通り，Agilla には遠隔反応メカニズムが備わっていない．したがって，仮にユーザが，遠隔反応による MA 協調動作が可能なアプリケーションを開発しようとすれば，MA は，定期的な遠隔 TS 読込操作（rrdpg）が必要である．一番分かりやすい方法は，MA 自体がループを実行し，ループの中で必ず rrdpg を実行することである．rrdpg 指令の実行結果の Trigger Tuple(TT) の存在状況により，登録したリアクションを実行するかしないかを決定する．

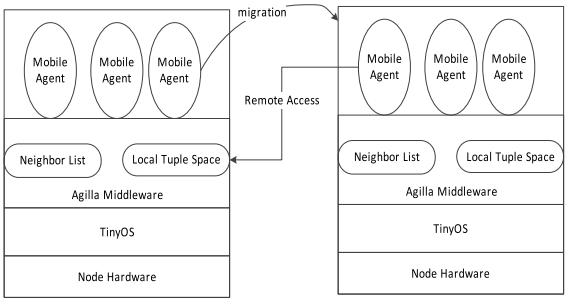

図 1 Agilla モデル ([4] より)

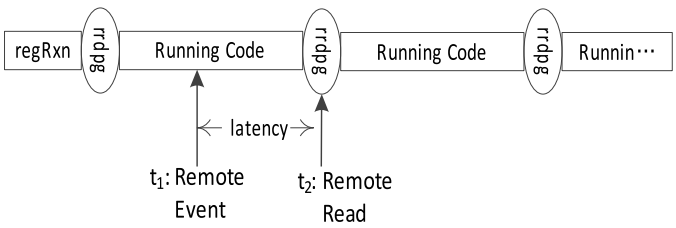

図 2 ループ動作によるレイテンシ

図 2 に示す動作例において，MA は rrdpg 操作の実行と通常のアプリケーションコードの実行を交互に繰り返している．ある TT が rrdpg の実行後，アプリケーションコードの実行中である時刻 t_1 に隣接ノードで生成された場合，この MA は，次の rrdpg 実行時刻 t_2 になってから，この TT に反応することになるため，$t_2 - t_1$ のレイテンシが発生する．

2.3.2 Tuple 獲得専用 MA による MA 管理量増大

2.3.1 の方法を改良し，リアルタイムに反応できるようにするためには，ノード上に TT 獲得用 MA と，アプリケーション実行用 MA の 2 種類の MA を用意する方法が考えられる．これによって反応レイテンシを削減することができるが，同一ノード上の MA 数を増やすことは大きな計算負荷となり，また資源の占有も問題となる．

2.3.3 エージェント移動式遠隔反応による通信量増大

最後の方法として，MA の移動能力を活用することが考えられる．MA 自体が他のノードへ移動することで，Remote Reaction の代わりに Local Reaction を実行して対応する．ただし，サイズが 64 ビット（Tuple Field が 32 ビット，Tuple 属性が 32 ビット）の TT の伝搬の代わりに，必須属性だけでも 106 ビット (プログラムコードや変数ヒープ・スタックは除いている) となる MA を伝搬するとこは，通信量の大幅な増大となる．

3. モデル設計

本稿で提案するモデルは，図 3 に示すように，3 つの層に分かれる．ユーザアプリケーションは，アプリケーション層に，MA としてまとめられる．MA は，ノードの間で移動・複製することが可能である．ミドルウェア層は，主に MA の管理・TS の管理を行う．最下層はノード OS 層である．現時点では，TinyOS を利用する予定である．

TS を管理するミドルウェア層は，Local Tuple Space (LTS) と，Neighbor List (NL) と，Reaction List(RL) の，三種類のストレージを保有する．LTS は，Tuple を保存する．MA は，ローカル TS 指令を実行し，現在位置するノードの LTS から Tuple を読み込むこと，および LTS に Tuple を書込むことができる．本モデルはリモート TS 指令も提供するため，MA は NL から隣接ノードのアドレスを獲得し，隣接ノードが保有する LTS を操作することが可能である．RL は，MA が登録した Reaction 情報を保存し，適切な Tuple が LTS に入るとき，Reaction を発火し，Reaction 登録 MA を通知する．Remote Reaction も提供するため，隣接ノードで登録された Reaction も，RR とし

て RL に登録する．

3.1 Mobile Agent

前述のとおり，MA は，自動的にノード間で移動・複製できるプロセスである．本モデルでは，Agilla のように，MA は二種類の移動が実行できる．すなわち，実行コードのみ移動する弱移動（Weak Move, wmove）と，コードと MA 状態をまとめて移動する強移動（Strong Move, smove）である．MA の状態を移動させる必要が無い場合，MA は wmove を実行することによって，MA 移動によるオーバーヘッドを削減する．

MA は，指定したノードへ移動（move）する，あるいは複製（clone）することができる．MA が移動する場合，MA が新たなノードに到着した後，元ノードに残した MA レプリカを削除する．複製する場合，元ノードにある MA レプリカは，そのまま残される．

3.2 Tuple Space

TS は，一種のデータ共有機構であり，共有する情報を，Tuple と呼ぶ構造体に記述し，TS に保存する．アプリケーションは，TS から，求めるパターンに合致する Tuple を読込み，獲得したい情報を入手することができる．本モデルは，Agilla のように，通信のオーバヘッドを回避するため，同じノード上で実行される MA が共有する LTS のみを提供し，複数のノードが保有する LTS で構成する Federated TSTS は提供しない．代わりに，Neighbor List によって，隣接ノードにアクセスし，隣接ノードの LTS を直接操作する．

本モデルが提供する TS 指令は，ローカル TS 指令とリモート指令の二種類がある．

ローカル TS 指令は，MA が，自身が位置するノードのローカル TS に対して行う操作である．Linda や LIME のように，主に out・in・rd 三種類の指令を提供する．out は，MA が通信情報を Tuple にまとめ，LTS に保存する操作である．in は，MA が求めるパターンに合致する Tuple を読み込む操作で，受け取った Tuple を，LTS から削除する．rd は in と似ている操作だが，受け取った Tuple をそのまま LTS に残す．もし LTS に，求めるパターンに合致する Tuple が存在しない場合，rd・in は，Tuple が見つかるまで MA をブロックする．本モデルはこれに加え，rdp と inp の 2 つの指令も提供する．p はプローブの意味で，この 2 つの指令は，合致する Tuple が見つからない場合，MA をブロックせず，null を MA に返信する．

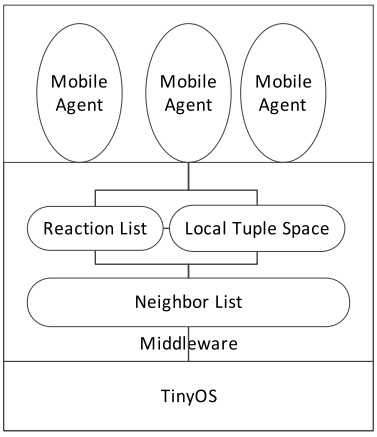

図 3 モデル設計

リモート TS 指令は，MA が，隣接ノードのローカル TS に対して行う操作である．ブロック操作によって頻繁なリモートアクセスが必要になり，ネットワーク通信に対し大きな負荷を与えるため，リモートブロック操作は提供しない．その結果，リモート指令としては，まず rout・rinp・rrdp 三種類を提供する．いずれも指定した一つのノードに対する操作である．本モデルはこれに加え，全ての隣接ノードに対する操作の，rrdpg と routg を提供する．ここで g は，グループを意味する．

3.3 Reaction

Reaction は，MA が，他の MA のイベントに基づいて，協調動作を実現する仕組みである．イベントは，MA が Tuple を TS に出す形で表現し，この Tuple を，便宜上 Trigger Tuple(TT) と呼ぶ．MA は，他の MA が出した TT に反応し，協調動作を実行したい場合，反応すべき TT のテンプレートと，自身の id をまとめて，Reaction 情報としてミドルウェアの Reaction 管理コンポーネントに登録する．ミドルウェアは，TS に，Reaction 情報にある TT テンプレートと合致する Tuple の存在を確定すると，Reaction 登録 MA に Reaction 発火通知を行う．発火通知を受けた MA は，MA の実行コードに記述された Reaction 反応コードを，割り込んで実行する．

本モデルは，Reaction 情報を，Local Reaction（LR）と Remote Reaction（RR）の二種類に分ける．LR は，MA が直接に位置するノードに登録する Reaction 情報である．RR は，LR が登録された際，ミドルウェアが隣接ノードに配布し登録する Reaction 情報で，親 LR と常に対応している．RR が発火したら，ミドルウェアは，RR と対応している親 LR ノードを通知し，親 LR を発火させる．RR と LR は常に対応付けられているため，Reaction 情報の同期も必要である．

3.4 動作性能に関する考察

本節では，提案モデルが Agilla と比較して，想定する動作における性能が改善されるかどうかに関する検討を行う．

主に次の 3 種類の項目について検討する．

(1) Agilla における rrdp ループ式 Reaction と，本モデル

における RR 式 Reaction の，反応レイテンシの比較
(2) Agilla における Tuple 獲得専用 MA 式 Reaction と，本モデルにおける RR 式 Reaction の，使用資源量の比較
(3) Agilla における MA 移動式 Reaction と，本モデルにおける RR 式 Reaction の，通信量の比較

3.4.1 反応レイテンシ

ここでは，rrdp ループ式 Reaction と，RR 式 Reaction による，反応レイテンシを比較する．

まず，必要な仮定について述べる．

(1) ノード N_a の上で，MA MA_a が動作する．N_a と常に通信できるノード N_b の上で，MA MA_b が動作する．
(2) MA_a は，動作開始時，Tuple t に反応する Reaction R_t を登録する．登録した後，ループ動作を繰り返し，ループの最初に遠隔リード rrdp を実行する．ループの実行時間は t_{loop}.
(3) MA_a は， MA_b との通信開始時から，リクエストが MA_b に到着するまで，t_{rq} の時間がかかる． MA_b は，リクエストを受けたら，直ちにレスポンスを生成し送信する．リクエストを受けた時点から，レスポンスが MA_a に到着するまで，t_{rp} の時間がかかる．
(4) MA_a は，t_a であるループ動作を開始する．その後，MA_b は，ある時刻 t_t に Tuple t を LTS に出す．このとき，$t_a + t_{rq} < t_t < t_t + t_{rq}$ となる．

従って，rrdp ループ式 Reaction と RR 式 Reaction による反応レイテンシは**表 1** のようになる．RR 式は，rrdp ループ式により，レイテンシが明らかに安定し，短くなる．

表 1 レイテンシ試算

反応方式	最小 latency	最大 latency	平均値
rrdp loop	t_{rp}	t_{loop}	$\frac{t_{rp}+t_{loop}}{2}$
Remote Reaction	t_{rp}	t_{rp}	t_{rp}

3.4.2 使用資源量の比較

ここで，Tuple 獲得専用 MA 式 Reaction と RR 式 Reaction による，使用資源量を比較する．

まず，必要な仮定について述べる．

(1) 各ノードは，6 つのノードと隣接し，通信ができる．
(2) 各ノードに，4 つの一般 MA が動作し，各 MA は Reaction1 つを登録する．
(3) MA コンテキスト・Reaction 共にデフォルトでメモリ資源を占有する．Agilla のコードによると，MA が 160 バイト，Reaction が 24 バイトのメモリを占有する．なお，MA 変数スタックは 50 バイトである．
(4) 各 MA は，5 ノードのキュー・リンクリストを持ち，各キューノード・リストノードに 2 バイトのデータを保存し，ノードポインタを含めて，総計 200 バイトのメモリを必要とする．
(5) MA のコードは計算に入っていない．

この時，Tuple 獲得専用 MA 式 Reaction と RR 式 Reaction による，各ノードのメモリ資源の使用量は**表 2** のようになる．RR 式は，Tuple 獲得専用 MA 式により，使用資源量が増える．

表 2 メモリ使用量試算

反応方式	MA 数	MA メモリ	Rxn 数	Rxn メモリ	総メモリ
Tuple Catch MA	5	2.05KB	4	96B	2.15KB
Remote Reaction	4	1.64KB	28	672B	2.31KB

3.4.3 通信量

最後に，MA 移動式 Reaction と，RR 式 Reaction による，通信量を比較する．

まず，必要な仮定について述べる．

(1) MA コンテキスト・変数スタック・キュー・リンクリストは，メモリ管理量の試算の場合と同じ．
(2) MA コードは，MA の移動と共に移動する．コードの大きさは 1KB と仮定する．
(3) RR 式で，RR が発火させたメッセージは，20 バイトの Tuple として返信する．

この時，MA 移動式 Reaction と RR 式 Reaction による，ネットワーク通信量は**表 3** のようになる．通信量は激減することが分かる．

表 3 ネットワーク通信量試算（バイト）

反応方式	送信量	返信量	総計
MA Migrate	1434	—	1434
Remote Reaction	24	20	44

4. シミュレータにおける実装

本稿が提案するモデルは，現在 NS 2 上でシミュレート用のプロトタイプの実装を進めている．本章では，その実装の概要について述べる．

4.1 システムアーキテクチャ

図 4 に示すように，NS2 上の実装は，3 つの層に分かれ，最上層はユーザアプリケーションを実行する MA 層，中間層は MA・TS 及び Reaction を管理するミドルウェア層となっている．この 2 つの層は，NS2 のアプリケーションとして実装する．最下層の OS・ノード層は，本稿では通信機能のシミュレートのみを目的としているため，NS 2 の通信エージェント・ノード層として実装する．無線ブロードキャストを用いてシミュレートするため，通信エージェントは，UDP エージェントを継承した，自作のブロードキャストエージェントを利用する．なお，最終的に実機のノー

ド上に実装する予定のため，すべてのプログラムは C++ ライブラリ関数は用いない．

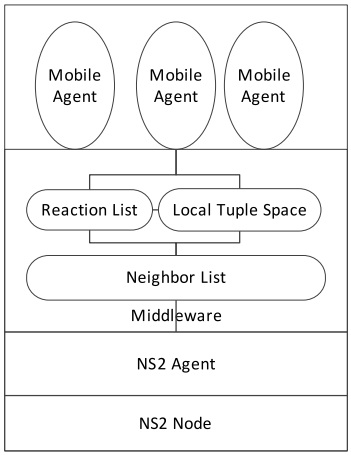

図 4 NS2 上のモデル実装

4.2 Mobile Agent

MA は，主シナリオ関数と，Reaction 関数 2 つの関数を保有する．システムが初期化されたときから，MA は主シナリオ関数を永続的に実行するが，ミドルウェア層から Reaction 発火通知を受けると，主シナリオ関数を一時停止し，Reaction 関数を実行する．Reaction 関数が実行完了後，主シナリオ関数を再開する．便宜上，主シナリオ関数は永続ルーチンを実行し，Reaction 関数の割込実行は，フラグ制御で実現する．

MA は，TS に対する TS 操作，及び Reaction 登録・登録消し操作ができるため，MA が位置するミドルウェアのポインタを持つ．

4.3 ミドルウェア

ミドルウェアは，MA リスト・LTS・RL・NL 四種類のストレージを持つ．

MA リストは，このミドルウェア上で実行する MA のポインタのリストで，最大で 4 個の MA を保存する．

LTS は，ミドルウェアに保存する Tuple ポインタのアクティブリストで，ミドルウェアに実行する MA に対し，rdp・inp・out の関数を提供する．ブロック実行はまだテストしないため，今回は実装しない．なお，遠隔 TS 操作の rout・rinp・rrdp・rrdpg は実装する．

LTS に保存する Tuple は，name と contents の二種類の変数を持つ．name は，Tuple の種類の名前で，contents は，Tuple に保存するデータである．

RL は，ノードに登録された Reaction 情報を保存するリストである．Reaction 情報は，三つの部分から構成される．一つはこの Reaction の種類である type で，type は Reaction が LR か RR かを示す．次は反応の対象となる Tuple のテンプレートである temp で，LTS に temp と合致する Tuple が存在した場合，Reaction 登録者情報である registrar を通知する．registrar は，LR 通知対象 MA の id，または RR 通知対象ノードのアドレスを持つ．

NL は，隣接ノードアドレスリストで，定期的に更新する．

4.3.1 Reaction 発火

LR の発火は，LTS に新たな Tuple が出された，あるいは RL に新たな Reaction が登録された場合，RL に記載された全ての Reaction 情報をスキャンし，各 Reaction 情報に保存する temp に合致する Tuple を，LTS から探す．全ての Reaction 情報をスキャンし終わるまで，探索を続ける．合致する Tuple が見つかったら，Reaction 登録 MA の Reaction フラグを立たせ，Reaction 関数を実行させる．

RR が発火すると，図 5 に示すように，下記の順序で協調動作が実行される．

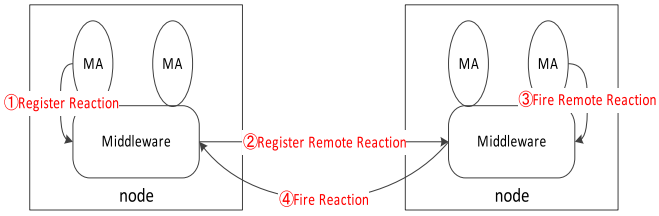

図 5 RR モデル

(1) MA は，Reaction 情報をローカルに登録する
(2) ミドルウェアは，Reaction 情報を RR とし，Neighbor List(NL) に記載されたすべての隣接ノードにも配布し登録する
(3) 隣接ノードに，ある MA が RR を発火させる Tuple を TS に出す
(4) 隣接ノードは，RR 登録ノードに通知し，RR 登録ノードに登録された Reaction を発火させ，RR 登録 MA を協調動作を実行させる

4.3.2 Neighbor・Reaction 同期

NL は，現時点では周期で更新するものとしている．更新する場合，ミドルウェアは，ブロードキャストで ping を送信し，10 秒以内に返信を受け取った送信ノードを，新たな Neighbor ノードとして更新する．

NL の更新，または MA の移動によって，下記の Reaction 同期動作を行う．

図 6 のように，もしある隣接ノードが通信できない状態となった場合，通信できなくなったノードは，NL から削除される．

図 7 のように，ノードを NL から削除するとき，ミドルウェアが，この削除されたノードに登録された RR を削除する．一方，NL が更新するとき，新たな隣接ノードを見

Vol.2014-DPS-158 No.38
Vol.2014-CSEC-64 No.38
2014/3/7

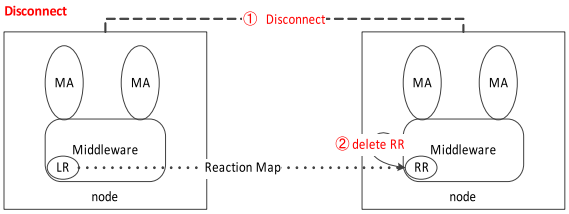

図 6 Reaction 同期—Disconnect

つけると，ノードは，この新たな隣接ノードに，現在管理している Local Reaction 情報を，RR として登録する．

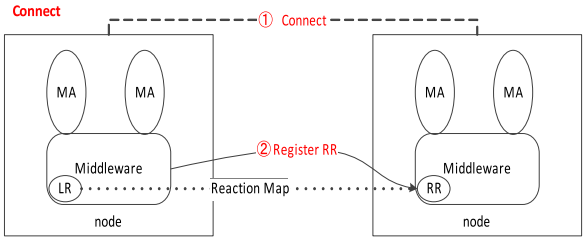

図 7 Reaction 同期—Connect

なお，Reaction の登録者は MA であるため，MA の移動・登録取り消し操作によって，ノードが保存する Local Reaction 情報も変動する．もし MA が他のノードへ複製ではなく移動する場合 (図 8)，あるいは Reaction 登録削除操作 (図 9) を行う場合，ノードは，該当する Local Reaction を削除すると同時に，隣接ノードへ RR 削除通知を配布する．RR 削除通知を受信したノードは，該当する RR を削除する．

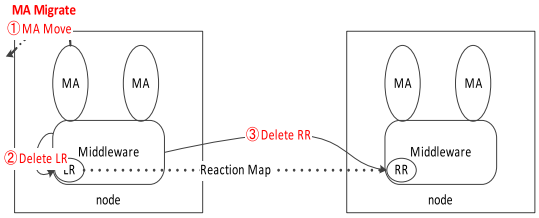

図 8 Reaction 同期—MA Migrate

現時点では，MA の移動はまだ実装されていないため，NS2 上での MA の生成・削除によって，MA の移動をシミュレートする．

5. シミュレーションによる評価

シミュレート環境は，50 × 50 のフィールドに，(5, 5) に無線ノード N_a，(25, 25) に無線ノード N_b を設置する．ZigBee を利用する予定だったが，現時点では実装の都合により，IEEE 802.11 と ADOV を利用している．

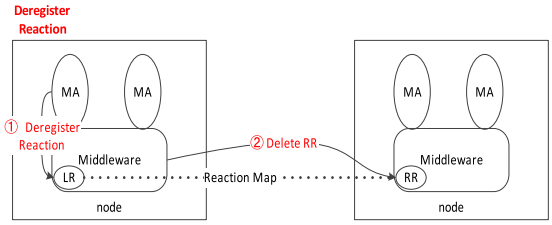

図 9 Reaction 同期—Deregister Reaction

シミュレーションシナリオは，3.4.1 に述べたように，2 つのノードの間のシナリオである．MA MA_a が N_a 上で動作し，MA MA_b が N_b 上で動作する．MA_b は，ループ動作中に，ランダムに，Tuple t を LTS に出す．MA_a が，t を獲得したら Reaction を発火し，獲得しない場合，ループを継続する．MA_a は MA_b が出した tuple に反応した時点で 1 回の実験が終了する．この実験を 1000 回実施する．

ループ時間 50ms のシミュレーション結果を**表 4** に示す (単位は ms)．この表の結果によると，遠隔反応機能を持つ提案モデルは，レイテンシが Agilla における rrdp ループにより，明らかに短くなる．

表 4 レイテンシのシミュレーション結果 (ms)

反応方式	最小レイテンシ	最大レイテンシ	平均値
rrdp loop	8.26	57.26	50.58
Remote Reaction	7.63	15.26	8.21

次に，ループの時間とレイテンシの関係の検証を行う．図 10 は，二種類の手法における，平均反応レイテンシと MA 実行ループ時間の関係を表す．横軸はループの長さであり，縦軸は平均レイテンシの長さである．そして青い線は rrdp ループ式反応モデルの性能曲線で，赤い線は遠隔反応機能を持つ提案モデルの性能曲線である．図より，ループの実行時間が長くなると，rrdp ループ式反応の平均レイテンシは長くなるが，提案モデルの反応レイテンシは，rrdp ループ式と比べ，ループ時間による大きな影響はなく，レイテンシがわずかに降下する．

図 11 は，二種類の手法における，平均レイテンシとループ時間の比率と，MA 実行ループ時間の関係を表す．横軸はループの長さで，縦軸は平均レイテンシとループ時間の比率である．青い線は rrdp ループ式反応モデルの性能曲線を示し，赤い線は遠隔反応機能を持つ提案モデルの性能曲線を示す．図より，ループの実行時間と関係なく，rrdp ループ式反応の平均レイテンシ比率はあるレベルを維持するが，ループ時間が長くなると，提案モデルの平均レイテンシ比率が下がることが確認できる．

結論として，提案モデルは，従来の Agilla における rrdp

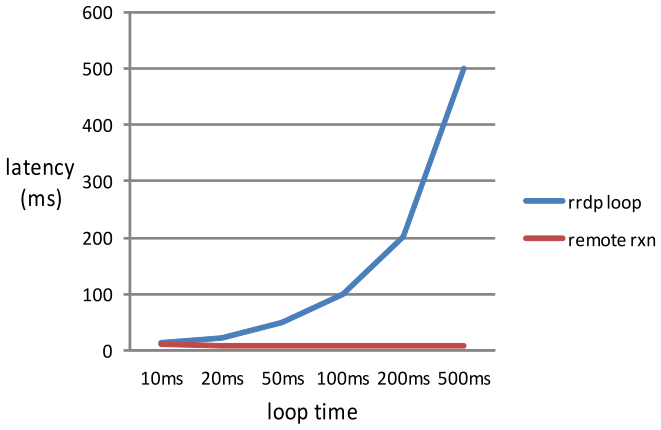

図 10 レイテンシとループの長さの関係

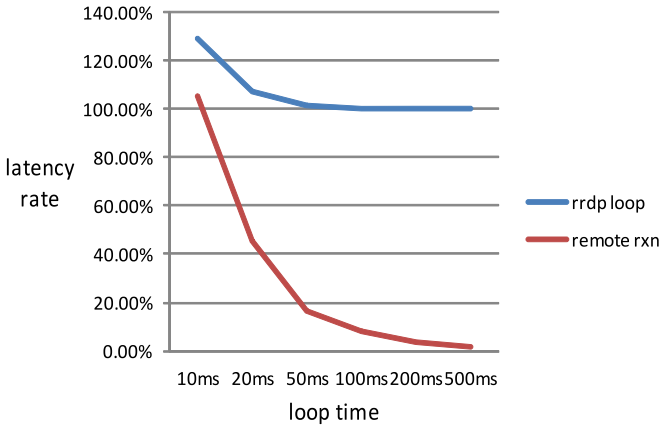

図 11 レイテンシ比率とループの長さの関係

ループ式反応による実装と比較して，レイテンシが安定し，ループの長さが大きい場合は，レイテンシをより有効に削減することが可能になることが分かる．

6. おわりに

本稿では，MA および TS を用いる WSN 向けミドルウェアについて述べ，これらのミドルウェアに課せられた課題について議論し，RR 機能を導入する MA 型ミドルウェアモデルを提案した．さらに，モデルの設計や，NS2 上でのシミュレート用プロトタイプシステムの実装とその評価について述べた．今後，最終的なシミュレーション実装が完成し次第，詳細な実験と評価を行う予定である．

今後の予定として，実機でのテストを検討している．実機でのテストは，Agilla をベースとして，Reaction 管理コンポーネントに，RR を追加し，提案モデルが実機上でどの程度計算量・データ管理量に変化を与えるかを検証する．

なお，RR の導入によって，メモリ管理量が増大することは試算により分かるが，その影響を減らすことも重要な課題である．現段階では，MA 数の制限を暫定的な解決策とするが，今後は MA を RR モデルに最適化することを検討している．

参考文献

[1] G. P. Picco, A. L. Murphy, G. Roman: “LIME: Linda Meets Mobility, ” ICSE ’99 Proceedings of the 21st international conference on Software engineering, Pages 368-377.
[2] A. L. Murphy, G. P. Picco, G. Roman: “LIME: A Coordination Model and Middleware Supporting Mobility of Hosts and Agents, ” ACM Transactions on Software Engineering and Methodology, Vol. 15, No.3, July 2006, Pages 279-328.
[3] P. Costa, L. Mottola, A. L. Murphy, G. P. Picco: “TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks, ” MidSens ’06 Proceedings of the international workshop on Middleware for sensor networks, Pages 43 - 48.
[4] C. Fok, G. Roman, C. Lu: “Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks, ” ACM Transactions on Autonomous and Adaptive Systems (TAAS), Volume 4 Issue 3, July 2009, Article No. 16.
[5] C. Fok, G. Roman, C. Lu: “Mobile Agent Middleware for Sensor Network — An Application Case Study, ” IPSN ’05 Proceedings of the 4th international symposium on Information processing in sensor networks, Article No. 51.
[6] U. Sharif, A. khan, M. Ahmed, W. Anwar, A. N. Khan: “CLUSMA: a Mobile Agent Based Clustering Middleware for Wireless Sensor Networks, ” FIT ’09, December 2009.
[7] L. Mottola, G. P. Picco: “Programming Wireless Sensor Networks: Fundamental Concepts and State of Art, ” ACM Computing Surveys (CSUR), Volume 43 Issue 3, April 2011, Article No. 19.