データベースシステム　122－25
(2000. 7. 27)

移動体計算環境におけるアクティブデータベースの動的トリガグラフ構築手法

寺田 努† 塚本 昌彦‡ 西尾 章治郎‡

†大阪大学サイバーメディアセンターサイバーコミュニティ研究部門
〒567-0047 大阪府茨木市美穂ヶ丘5-1
tsutomu@cmc.osaka-u.ac.jp

‡大阪大学大学院工学研究科情報システム工学専攻
〒565-0871 大阪府吹田市山田丘2-1
{tuka, nishio}@ise.eng.osaka-u.ac.jp

あらまし　無線通信や計算機ハードウェア技術の急速な発展により，ユーザは無線通信機能を持つ携帯端末を用いて，場所を固定せずにネットワークを介して情報を利用することが可能になった．筆者らは，このような環境において移動体がもつデータを統合利用するために，アクティブデータベースを拡張し，移動体の接続，切断，データ交換などを処理するAMDS(Active Mobile Database System)を提案・実装してきた．AMDSの動作言語であるECAルールは，記述能力が高く，連鎖的に実行させることで複雑な処理が記述できる一方，予期しない異常動作を起こす可能性がある．一般にアクティブデータベースの異常動作検出にはトリガグラフと呼ばれる有向グラフを用いるが，トリガグラフはネットワーク構成に依存するため，ネットワーク構成が動的に変化する移動体計算環境で用いることは困難である．そこで，本研究では動的にトリガグラフを再構築して異常動作を検出する手法を提案する．本機構を用いることで，ECAルールを用いたアプリケーションをより安全に運用できるようになる．

キーワード　移動体計算環境，アクティブデータベース，ECAルール，トリガグラフ

A Method for Constructing Trigger Graph Dynamically on Active Database in Mobile Computing Environments

Tsutomu TERADA† Masahiko TSUKAMOTO‡ Shojiro NISHIO‡

†Cybercommunity Division, Cybermedia Center, Osaka University
5-1, Mihogaoka, Ibaraki, Osaka 567-0047, Japan
tsutomu@cmc.osaka-u.ac.jp

‡Department of Information Systems Engineering, Graduate School of Engineering, Osaka University
2-1, Yamadaoka, Suita 565-0871, Japan
{tuka, nishio}@ise.eng.osaka-u.ac.jp

Abstract　As a result of rapid development in wireless communications and computer hardware technologies, currently, we can access various information from anywhere using handy terminals with wireless communication capabilities. To support the integration and the use of data held by mobile hosts in this environment, we have proposed and implemented AMDS (Active Mobile Database System) as the kernel system for data and mobile host management in mobile environment. The behavior definition language of AMDS, ECA rules, have high description capability that enables users to define complicated behavior. However, the execution of ECA rules may fall into a chain of unexpected behaviors. In general, a directed graph called trigger graph is used for detecting chains. Unfortunately, trigger graph highly depends on network topology, thus it is difficult to employ trigger graph in mobile computing environment. In this paper, we propose a method that could reconstruct trigger graph dynamically to adapt to changes in network topology. By using this mechanism, mobile applications with ECA rules can be used more safely.

key words　Mobile Computing Environment, Active Database, ECA rule, Trigger Graph

1 はじめに

近年，無線通信技術や計算機ハードウェア技術の急速な発展により，無線通信機能をもつ携帯端末を用いることで場所を固定せずにネットワーク上のさまざまな資源を利用することが可能になった．この新しい計算環境を移動体計算環境と呼ぶ．移動体計算環境のモデルは図1に示すように，固定ネットワークに無線通信可能な移動端末を含んだ形態である．移動体計算環境においては，以下のような新しいサービスを提供できるようになる．

- 会社内などにおいて，各社員は一台ずつ各自のスケジュールが入力された携帯端末を持ち歩き，本社でスケジュールデータを統合して全社員のスケジュールを管理し，効率的に仕事を割り当てる．
- 遊園地などのアミューズメント施設で入場者に一台ずつ携帯端末を持たせ，各アトラクションの待ち時間等の情報を提供する．
- 美術館や博物館において，入館者は一台ずつ携帯端末を持ち，展示物に近づくと自動的にその展示物の詳細情報が表示される．

このようなサービスを実現するためには，移動体から，または移動体上でデータを収集する必要がある．しかし，従来の分散データ管理技術では移動するホストを考慮していないため，移動体のネットワークへの接続・切断やデータの収集を自動的に行なう共通の基盤が望まれている．このような要求に対し，筆者らは，イベント駆動型データベースであるアクティブデータベースを拡張することで，移動体計算環境における各種のイベントを容易に扱うことができるアクティブモーバイルデータベース (Active Mobile Database System: AMDS) の研究を行なってきた [3][9]．AMDS では，イベント，コンディション，アクションの3つを一組として記述するECAルールによって動作を記述する．イベントには従来のアクティブデータベースのイベント (更新・挿入・削除など) に加え，移動体の接続・切断などを扱うイベントも提供している．

ECAルールは，アクションの実行によって新たなイベントを引き起こすことができる．ECAルールを連鎖的に実行させることで，複雑な動作が実現できる一方，

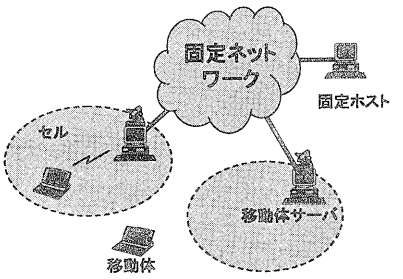

図 1: 移動体計算環境

無限ループなど，予期せぬ動作に陥る可能性がある [4]．そのため，ECAルールの実行を監視してシステムの異常動作を回避する必要がある．そこで，本研究では，移動体計算環境において，ECAルールの連鎖によるシステムの異常動作を検出するための動的トリガグラフ構築手法を提案する．以下，2章ではAMDSの概要について述べ，3章ではアクティブデータベースの異常動作とその一般的な解決手法について述べる．4章で提案する手法について説明し，最後に5章で本研究のまとめと今後の課題について述べる．

2 AMDS

AMDSはアクティブデータベースを拡張して移動体計算環境に適合させたものである．アクティブデータベースは，データベースの内界・外界で起こる事象の発生に対して，規定された処理を行なうデータベースである [2]．その動作は，発生する事象 (イベント)，ルールの発火条件 (コンディション)，実行される操作 (アクション) の3つの組みで表わされるECAルールで記述される．

AMDSには，従来のアクティブデータベースで提供されている，データベースに関するイベント (挿入，削除，更新，検索) に加えて，移動体の動作に関するイベント (移動体のセルへの接続・切断)，AMDS間の通信に関するイベント (データの受信) が用意されている．アクションにはAMDS間のデータ送信関数，データベース対する問合せ関数などが提供されている [5]．AMDSで提供されているイベント，アクションを表1，2に示

表 1: AMDSのイベント

名称	内容
CONNECT	移動体のセルへの接続
DISCONNECT	移動体のセルからの切断
SELECT	テーブルに対するデータ参照
INSERT	テーブルに対するタップルの挿入
DELETE	テーブルのタップル削除
UPDATE	テーブルのタップル更新
RECEIVE	データパケットの受信
TIMER	設定したタイマの発火

表 2: AMDSのアクション

名称	内容
QUERY([クエリー内容])	データベース操作
SEND([宛先], [送信内容])	データの送信
INSERT_ECA([ルール内容])	ECAルール格納
DELETE_ECA([ルール識別子])	ECAルール削除
ENABLE_ECA([ルール抽出条件])	ECAルール有効化
DISABLE_ECA([ルール抽出条件])	ECAルール無効化
SET_TIMER([タイマ条件])	新たなタイマの設定
KILL_TIMER([タイマ識別子])	タイマの削除

表 3: NEWデータとOLDデータの内容

イベント	NEW	OLD
CONNECT	接続移動体情報	–
DISCONNECT	–	切断移動体情報
SELECT	参照タップル	–
INSERT	挿入タップル	–
DELETE	–	削除タップル
UPDATE	更新後タップル	更新前タップル
RECEIVE	到着パケット内容	–
TIMER	タイマ識別子	–

```
create rule 接続 on CONNECT
  then do
   SEND( new.from, "Request_" );

create rule 返信 on RECEIVE
  where new.header = 'Request_'
  then do data = QUERY("select s.*
                        from Schedule s");
   SEND( new.from, "result_", data );
```

図 2: ECAルール例

す．また，到着したデータの内容を用いて処理を行なうルールなどでは，イベント対象となったタップル情報が必要になる場合があるため，NEWデータ，OLDデータと呼ぶシステム変数を用意する．イベント発生時にこれらの変数に必要な情報が自動的に格納され，ルール中で自由に使用することができる．各イベントに対するNEWデータ，OLDデータの内容を表3に示す．

AMDSは一般のアクティブデータベースと同様ECAルールで動作する．AMDSにおけるECAルールの記述例を図2に示す．接続ルールは，移動体が接続してきたときに，その移動体に対してデータ要求を行なう移動体サーバ用のルールである．また，返信ルールは，移動体サーバからの要求パケットを受信したとき，スケジュールデータを送り返す移動体用ルールである．

移動体サーバでは，移動体が接続してきたときに接続ルールが起動し，接続ルールのアクションにより，移動体の返信ルールが起動する．このように，ECAルールでは，アクションの実行が新たなイベントを発生させることができるため，1つのイベントの発生によって複数のECAルールを連鎖的に実行させて複雑な動作を実現できる．

3 異常動作の検出

本章では，ECAルールの連鎖による異常動作の例を示し，アクティブデータベースの異常動作検出手法と，その問題点および解決手法について説明する．

3.1 ECAルールの異常動作

ECAルールは，アクションの実行が新たなイベントを発生させることができるため，複数のECAルールを連鎖的に実行させることができる．そのため，ECAルールを連鎖的に実行させることで複雑な動作を実現できる一方，無限ループなど予期しない連鎖を起こす可能性があり，このような異常動作を検出することが必要となる．

図3，表4に，異常動作の例を示す．R1，R2は移動体サーバに格納されているルール，R3は移動体に格納されているルールである．移動体がセルに接続すると，R1が起動され，移動体サーバは移動体に問合せを行なう．問合せにより移動体のR3が起動され，移動体は移動体サーバに移動体サーバの問合せを行なう．これによりR2が起動されるが，このルールは再びR3を起動

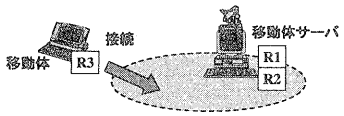

図 3: 異常動作の例

表 4: ルールの内容

ルール	E	C	A
R1	移動体接続	–	データ要求
R2	パケット到着	身元確認	身元要求
R3	パケット到着	–	身元要求

するため，R2とR3の間に無限ループが発生する．

移動体がもつルールと移動体サーバがもつルールを個別に見ても，ループを起こすかは判断し難いが，移動体の移動や，もっているルールの変化などにより，異常動作を起こしてしまう．

このような異常動作を解析的に検出するのは困難であるが，ECAルールを用いたアプリケーションを構築する場合，ECAルールが異常動作を起こさないことを保証する必要がある．したがって，異常動作が起こるかどうかをあらかじめ調べたり，異常が実際に起こったときにそれを検出するような仕組みが必要となる．

3.2 異常動作の検出手法

アクティブデータベースにおける異常動作検出には，次に示す2つの手法が考えられる．

- 実行時検出

  実際にデータベースが動作している状態で，リアルタイムに異常動作の有無を検出する．

- 事前検出

  データベース(ECAルール)が動作していない状態で，トリガグラフと呼ばれる有向グラフを用いて論理的に異常動作の有無を調べる．

実行時検出では，ルールの連鎖カウンタやタイムスタンプを用いて異常動作を検出する[8]．実行時検出は実際に異常が発生してからその検出を行なうので，システムを停止したくない場合や，システムの安全性をチェックしておきたい場合などには用いることができない．したがって，異常が実際に発生する前にその可能性を調べる事前検出が必要となる．

事前検出で用いるトリガグラフとは，あるECAルールから次に引き起こすルールに対して有向のパスをはるという作業をすべてのルールに対して行なったもので，出来上がったグラフ中にループが存在していなければ，そのルール群は安全であるとするものである[6]．トリガグラフは，その信頼性を高めるためのさまざまな拡張がされており[1][7]，アクティブデータベースの無限ループ検出においては信頼性が高い．

しかし，AMDSで想定する移動体計算環境においては，移動体の移動によりネットワーク構成が動的に変化する．また，AMDSでは，ECAルールのサイト間でのやりとりも頻繁に起こるため，複数サイト間にまたがるECAルールの相互関係を考慮する必要がある．したがって，移動体計算環境においてトリガグラフを作成するためには，あらかじめすべての移動体や固定ホスト内にあるECAルールを知っておかなければならない．さらに，ネットワーク構成の変化に応じてトリガグラフの構成も変化するため，各移動体があらゆる移動体サーバに接続した場合についてトリガグラフを構築する必要がある[4]．実環境では，どのようなルールをもった移動体が接続してくるかを知ることは困難であり，また，すべてのトポロジについて調べるのは計算量の大きさから見ても現実的でない．

そこで，本研究では新たに直前検出と呼ぶ検出手法を提案する．直前検出手法では，ネットワーク構成の変化に応じてトリガ情報をホスト間で送受信してトリガグラフを再構築する．本手法は，トリガグラフを用いて異常動作を検出するため，実際に異常が発生する前にその可能性を調べることができる．また，ネットワーク構成の変化に応じて必要な情報だけをやり取りするため，新たな移動体の接続にも対処でき，計算量も削減できる．

4 検出アルゴリズム

本章では，提案する直前検出手法について述べる．まず，トリガグラフの構築アルゴリズムについて述べ，次にトリガグラフの更新処理について説明する．そして，異常動作を検出したときの処理について述べ，最後に本アルゴリズムの適用例を示す．

4.1 トリガグラフの構築

トリガグラフの構築は以下のステップで行なう.

1. 自サイトのトリガグラフ生成と無限ループ検出.
2. RSパスの検出.
3. RSパスのマージ.
4. RSパスの送信.
5. 他サイトでのRSパスの受信処理.

まず, 各ホストがそれぞれ自サイトのトリガグラフを作成し, ローカルな無限ループの検出を行なう. ここで異常が検出されなかった場合, サイト間にまたがった連鎖を引き起こす可能性のあるルールから構成される部分トリガグラフを送信する. この部分グラフを**RS(Receive-Send)パス**と呼ぶ. 他のサイトではこのRSパスを受け取ったら, その情報を含めて再び無限ループの検出処理を行なう.

以下, それぞれのステップについて説明する.

4.1.1 トリガグラフ生成のタイミング

自サイトのトリガグラフ生成が必要となったとき, アルゴリズムが開始される. トリガグラフ生成が必要となるタイミングは以下のうちいずれかとなる.

- **システム開始時.**
- **ネットワーク構成が変化したとき.**

  ネットワーク構成の変化によりトリガグラフの再構成が必要になる. 具体的には移動体サーバがCONNECTイベントまたはDISCONNECTイベントを検出した場合となる.

- **ECAルールの構成が変化したとき.**

  ECAルール構成の変化によりトリガグラフの再構成が必要になる. 具体的にはINSERT_ECA, DELETE_ECAアクションの実行によるECAルールの追加・削除, ENABLE_ECAアクション, DISABLE_ECAアクションを実行によるECAルールの動作停止・解除が行なわれた場合となる.

- **他のサイトからRSパス情報を受け取ったとき.**

  他のサイトのトリガグラフ情報の更新により, RSパス情報が送信される. それを受け取った場合, 自サイトのトリガグラフも再構築する必要がある.

4.1.2 自サイトのトリガグラフ作成

自サイトのトリガグラフを構築する作業は次のような手順で行なう.

1. **パス集合の作成.**

   自サイトのルール集合に属するルールすべてについて, そのアクションが発火させる可能性のあるルールがあればそのパスをパス集合に加える.

2. **パス集合から連鎖パスを生成.**

   パス集合のすべての要素について, パス集合の要素を用いてパスを連結する操作を繰り返し, ループを検出する. 本手法では, あるルールから始まった連鎖が再び同じルールを発火させた場合ループであると判断する. もしループとなるパスが存在しなければトリガパス生成は正常終了する.

3. **コンディションのチェック.**

   ループが存在した場合, それが無限ループになるかどうかを判断するために以下の手順でコンディションのチェックを行なう.

   (a) NEW, OLD変数の置換.

   コンディションに, NEW変数やOLD変数が用いられていた場合, そのルールを引き起こしたルールのアクションを参照し, NEWやOLDを具体的なテーブル名や値に変換する. この操作を行なうことで, 複数のコンディションが同じテーブルにアクセスしているかどうかが判断できる.

   (b) コンディションのマージ.

   連鎖パスに沿ってコンディションをANDで連結する. コンディションを連結する際にはそのルールのアクションもチェックし, そのアクションがアクセスしているテーブルに関するコンディションは, 無効になっている可能性があるのでそれまでの連結コンディションから取り除く.

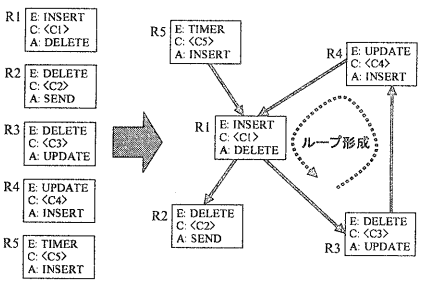

図 4: トリガグラフ構築例

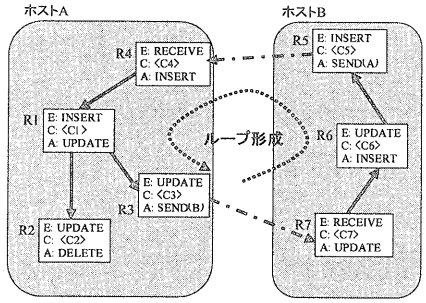

図 5: サイト間にまたがった連鎖

(c) コンディションのチェック.

マージされたコンディションに明らかな矛盾がなければ，無限ループと判断．矛盾があればその連鎖パスは無効となる．

このような手順でトリガグラフを作成し，ホスト内での無限ループを検出する．図4にトリガグラフの構築例を示す．システム内にR1からR5までの5つのルールが存在する．ルールxからyへのパスを(x,y)，パスを組み合わせてできる連鎖を連鎖パスと呼び，ルールx,y,zがその順で発火するとき(x,y,z)と表現すると，図4におけるパス集合は$\{(R1,R2),(R1,R3),(R3,R4),(R4,R1),(R5,R1)\}$となる．このパス集合から連鎖パス$(R1,R3,R4,R1)$が検出され，もしマージされたコンディションが成り立つ可能性があるならループを検出したことになる．

4.1.3 RSパスの作成

AMDSにおいて，図5に示すような，サイト間にまたがった連鎖を引き起こすのは，SENDアクションとRECEIVEイベントの組み合わせだけである．また，サイト間にまたがった連鎖により無限ループが発生するためには，無限ループを構成するサイトにRECEIVEイベントで始まってSENDアクションで終わる連鎖パスが存在する必要がある．そこで，RECEIVEイベント→SENDアクションのパスを**RS(Receive-Send)パス**と呼ぶ．自分のサイトにRSパスが存在した場合，それがサイト間にまたがった無限ループを構成する可能性があるため，関連するサイトにRSパスの情報を送信することでサイト間にまたがった無限ループを検出する．

RSパスの検出方法は次のようになる．まず，自サイトのパス集合の中からRECEIVEイベントで始まっているものを選び，4.1.2節で自サイトのトリガグラフを構築した際と同様の手法を用いて連鎖パスを検出する．もし，連鎖パスの中にSENDアクションで終わっているものがあれば，そこまでをRSパスとして検出する．

4.1.4 RSパスのマージ

RSパスは送信先サイトでそのままルールの連鎖パスとして使われ，さらに他のサイトに伝播することもあるため，出来上がったRSパスをそのまま送信すると，通信量が多くなってしまう．そこで，以下のような手順でRSパスから不要な情報を削除し，データ量を削減する．

1. **連鎖パス内でのコンディションのマージ.**

   RSパスは，連鎖を構成するすべてのルールの情報を持っているが，最終的に利用するのは先頭のルールのRECEIVEイベントの部分と最後のルールのSENDアクションの部分のみとなる．そこで，RSパスを1つのルールの形に縮退する．まず，先頭ルールのイベントと最終ルールのアクションをそれぞれマージ後のルールのイベント，アクションとし，4.1.2節の方法を用いてコンディションをマージした結果をマージ後のコンディションとする．ここで，ローカルに存在するテーブルに関するコンディションは，RSパス送信先のサイトには関係しないのですべて取り除いておく．

2. **複数の縮退RSパスのマージ.**

縮退したRSパスのうち，SENDアクションの宛先が同じものがあれば，コンディションをOR条件で統合することで，1つのルールにマージする．

4.1.5 RSパスの送信

生成したRSパスを送信する．送信先サイトは以下のどちらかとなる．

- **対象のRSパスのSENDアクションに宛先が指定されている場合.**

  宛先が固定されている場合は，そのサイトにしかルールの連鎖が起こらないため，対象サイトのみにRSパスを送信する．

- **SENDアクションに宛先がない場合.**

  宛先を指定しない場合は，無線通信可能な範囲にいるすべてのサイトが対象になるため，RSパスも無線通信可能なすべてのサイトに送信する．

4.2 トリガグラフの更新処理

ネットワーク構成やECAルール構成が変化した場合，その時点までのトリガグラフは無効になる場合がある．また，サイト間にまたがるトリガグラフの場合，ローカルの更新を他のサイトに伝播する必要がある．そのため，状況の変化に応じてトリガグラフの再構築処理を行なう．以下，それぞれの場合のトリガグラフ更新処理について述べる．

- **ECAルールが追加された場合.**

  ECAルールが追加された場合は，そのECAルールがトリガするルールとのパス及びそのECAルールをトリガするルールとのパスをパス集合に追加し，再び連鎖パス検出処理を行なう．

- **移動体が接続した場合.**

  移動体が移動体サーバのセルに入ったとき，移動体サーバはその接続を検出し，接続移動体に移動体サーバのRSパスを送信する．送信するRSパスが存在した場合は，移動体側も移動体サーバへの接続を認識するため，移動体からのRSパスも自動的に受信される．移動体サーバから送信するRSパスがない場合，移動体サーバは移動体にRSパス要求を送る．

- **ECAルールが削除された場合.**

  ECAルールが削除された場合，パス集合から対象となるECAルールを含むパスを削除する．次に，自サイトがもつRSパスの中に対象ECAルールが含まれていれば，そのRSパスはすでに無効になってしまっているため，RSパスの送信先に削除要求を出し，自サイトからも削除する．

- **移動体が切断した場合.**

  移動体が切断したとき，移動体サーバがその切断を検出する．移動体サーバは切断した移動体から送られていたRSパスを削除し，自サイトにそのRSパスを用いて作成された他のRSパスが存在すれば，さらにRSパスの送信先に削除要求を出す．

4.3 異常動作検出後の処理

本研究で提案する手法を用いて異常が検出されなかった場合，そのルール構成が安全であることが保障できる．しかし，トリガグラフによる異常の検出は，異常動作を起こす可能性を示すものであり，必ず異常が起こることを保証するものではない．したがって，トリガグラフにより異常が発生した場合も単純にシステムを停止させるだけではなく，柔軟な処理が行なえることが望ましい．

そこで，本手法では異常動作を検出した場合，以下の3種類の処理を選択的に，または組み合わせて利用できるようにする．

- **ユーザへの通知.**

  この処理手法は，異常動作を検出した場合ユーザにアプリケーションの継続か停止かを選択させる手法である．実行を継続する場合，ループを形成している連鎖パスを危険パスとし，危険パスが実際に発火した場合にはそれ以降のイベント発火とルール連鎖をトレースしてログに記録する．異常が発生した場合，ログを利用することで原因の解決を図ることは一般的であるが，本手法では危険性がある場合のログだけが記録されるため，ログ記録処理のオーバーヘッドは少なくてすむ．

- **異常発生原因の除去.**

  この処理手法は，異常動作の原因となったルールやホストを切り離すことで無限ループを回避する

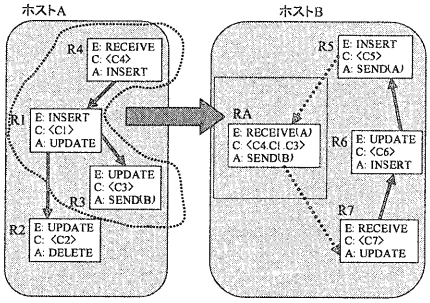

図 6: 適用例

方法である．例えば移動体が接続してきたことにより，トリガグラフがループを検出した場合，その移動体との通信を拒否するといった方法で原因の除去を行なう．

- **エラーイベントを発生させる．**

  異常動作を検出したときの状況をNEWデータにもつERRORイベントを発生させる．したがって，ERRORイベントを処理するECAルールを記述しておくことで柔軟なエラー処理が可能となる．

4.4 適用例

図5の例に提案するアルゴリズムを適用した場合を図6に示す．まず，ホストAにおいてトリガグラフが作成される．ここでは無限ループが発生していない．次にホストAにおいてRSパスを検出する．見つかったパス(R4, R1, R3)の情報をマージする．マージした結果できたノードRAのイベント，コンディション，アクションはそれぞれ，R4のRECEIVEイベント，R4，R1，R3のコンディションをマージした結果できたコンディション，R3のSENDアクションとなる．RAは，SENDアクションの送信先であるホストBに送信され，ホストBではRSパスデータを受信したとき，もともと自分のホストにあるルール(R5, R6, R7)のパスに加えて，受信したパスを加えたトリガグラフを生成する．この時点でループ(R7, R6, R5, RA, R7...)が発見され，異常動作が検出されることになる．

5 おわりに

本研究では，AMDSにおける異常動作検出手法である直前検出手法について述べた．本手法を用いることで，移動体計算環境においてもECAルールの異常動作が検出でき，AMDSをより安全に運用できるようになる．今後は，実際に本手法をAMDS上に実装し，静的検出および動的検出との最適な組み合わせについての評価を行なっていく予定である．

謝辞

本研究は，日本学術振興会未来開拓学術研究推進事業における研究プロジェクト「マルチメディア・コンテンツの高次処理の研究(Project No. JSPS-RFTF97P00501)の研究助成によるものである．ここに記して深謝の意を表す．

参考文献

[1] A. P. Karadimce and S. D. Urban, "Refined Trigger Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database," *ICDE'96*, pp. 384–391 (1996).

[2] 石川 博, "アクティブデータベース," 情報処理, vol. 35, no. 2, pp. 120–129 (1994).

[3] 村瀬 亨, 塚本昌彦, 西尾章治郎, "アクティブデータベースシステムによる移動体計算環境におけるデータ統合," 電子情報通信学会データ工学研究会, vol. 95, no. 287, pp. 41–48 (1995).

[4] 村瀬 亨, 塚本昌彦, 西尾章治郎, "移動体環境におけるアクティブデータベースの安全性について," 情報処理学会研究報告96-DBS-106, vol. 96, no. 11, pp. 33–40 (1996).

[5] 成田藤智, 村瀬 亨, 塚本昌彦, 西尾章治郎, "移動体環境におけるアクティブデータベースの設計と実装," 電子情報通信学会総合大会講演論文集, pp. 315–316 (1996).

[6] S. Ceri and J. Widom, "Deriving Production Rules for Constraint Maintenance," *Proc 16th VLDB Conf*, pp. 566–577 (1990).

[7] S. Y. Lee, T. W. Ling, "A Path Removing Technique for Detecting Trigger Termination," *EDBT*, pp. 341–355 (1998).

[8] 寺田 努, 莫 君, 村瀬 亨, 塚本昌彦, 西尾章治郎, "移動体計算環境におけるアクティブデータベースのECAルール実行監視機構の設計と実装," 情報処理学会研究報告(データベースシステム研究会99-DBS-119) (1999).

[9] T. Murase, M. Tsukamoto, and S. Nishio, "A system Platform for Mobile Computing base on Active Database," *in Proc. International Symposium on Cooperative Database Systems for Adavanced Applications*, vol. 2, pp. 424–427 (1996).