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機構の設計と実装

寺田 努[†] 塚本昌彦[††] 西尾章治郎[††]

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持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．

Design and Implementation of a Dynamic Construction Mechanism of a Trigger Graph on Active Databases in Mobile Computing Environments

Tsutomu Terada,[†] Masahiko Tsukamoto[††] and Shojiro Nishio[††]

As a result of rapid development of wireless communications and computer hardware technologies, currently, we can access various information from anywhere using handy terminals with wireless communication capabilities. To support the integrative use of data held by mobile hosts in this environment, we proposed and implemented AMDS (Active Mobile Database System) as the kernel system for data management. The behavior definition language of this system, ECA rules, has a 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. Since trigger graph highly depends on network topology, it is difficult to employ trigger graph in mobile computing environment. In this paper, we propose a method that 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.

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，無線通信機能を持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 1 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 1 台ずつ携帯端末を持たせ，各アトラクションの待ち時間などの情報を提供する．
- 美術館や博物館において，入館者は 1 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．

† 大阪大学サイバーメディアセンター
Cybermedia Center, Osaka University
†† 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Osaka University

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ホストの移動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）の研究を行ってきた[10),11)]．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，無限ループなど，予期しない動作に陥る可能性がある．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直前検出手法を提案する．提案する手法は，必要な情報のみをホスト間でやりとりするため，少ないトラフィック量で異常動作を検出できる．さらに，筆者らが提案する直前検出手法をAMDS上に実装するとともに，シミュレーションによる評価を行い，本手法の有効性とトラフィックに与える影響を評価する．

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，5章でシステムの実装について述べる．6章で提案手法の考察を行い，7章で関連研究について述べる．最後に8章で本論文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うデータベースである[5)]．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つの組みで表されるECAルールで記述される．

AMDSでは，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）を用意している．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[11)]．

以下，AMDSの動作記述言語であるECAルールと，AMDSにおけるECAルールの処理モデルについて詳細に述べる．

2.1 ECAルール

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ルールの記述構文を**図2**に示す．「イベント名」には，ルールが対象とするイベントの名前を記述し，「対象テーブル」には，イベントが対象とするテーブルを指定する．イベントは1つのルールに対して1つのみ記述でき，複数イベントの同時発生といった記述はできない．「変数型宣言」では，ECAルール中で使用するローカル変数を定義する．宣言する変数は，文字列変数，タップル変数またはタップルの組を表す変数のいずれかである．「コンディション」にはルールの発火条件を記述する．記述は，‘< 左辺 > < オペレータ > < 右辺 >’の形の羅列で行う．両辺には，データベース属性や変数を指定する．「アクション」にはルール

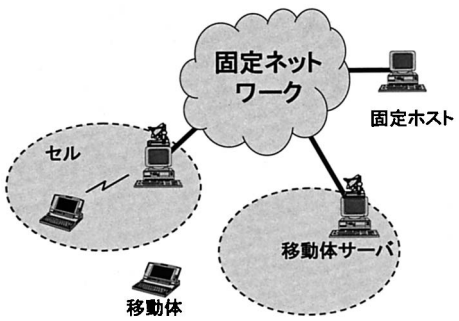

図1 移動体計算環境
Fig. 1 Mobile computing environments.

```
CREATE RULE ルール名 ON イベント名
     [ TO 対象テーブル ]
     [ 変数型宣言 ]
     [ WHERE コンディション ]
     THEN DO アクション
```

図2 ECAルールの記述構文
Fig. 2 Syntax of ECA-Rule.

表 1 AMDS のイベント
Table 1 Events provided by AMDS.

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
Table 2 Actions provided by AMDS.

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
Table 3 Contents of NEW data and OLD data.

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	–

が発火したときに行う動作を 1 つ以上記述する．

AMDS で提供されているイベント，アクションを**表 1**，**表 2** に示す．イベントのうち，CONNECT イベントおよび DISCONNECT イベントに関しては，移動体サーバにおいてのみ発火するイベントである．また，到着したデータの内容を用いて処理を行うルールなどでは，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．

ECA ルールの記述例として，近づいてきた移動体のスケジュールを収集するルール例を**図 3** 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．

```
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 3 ECA ルール例
Fig. 3 An example of ECA-Rules.

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．

2.2 ECA ルールの実行モデル

前節で示した記述構文に沿って生成された ECA ルールは，AMDS 上で次のように処理される．

(1) イベントキューからイベントを 1 つ取り出す．
(2) イベントに適合するルールを検索．
(3) 適合するルールがあればコンディションを評価．
(4) コンディションが真ならアクション実行．

システムはイベントの発生を検出すると，まず発生イベントに関する NEW データ，OLD データを作成し，それらをイベントキューと呼ぶイベント格納用キューに格納する．システムはつねにイベントキューを監視しており，キューにイベントがあれば取り出して発生したイベントに適合するルールを検索する．検索されたルール群はそれぞれコンディション評価され，コンディションが成り立つならアクションが実行される．ルールは実行プライオリティを持たないため，ルール群に属するルールが評価される順番は規定されない．また，アクション実行によって発生する新たなイベントはイベントキューに格納されるため，ルール群の処理がすべて終わるまでは違うイベントに関する処理が行われることはない．コンディション記述においてタップルの参照などイベントとなりうる操作を行う場合，その操作はイベントとして認識されない．

ルールの実行は集合指向で行われる．たとえば，1 つの SQL 文を用いてデータベースに 3 つのタップル

が挿入された場合，3 つの INSERT イベントが起こるのではなく，1 つの INSERT イベントしか起こらない．

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，アクションの実行により新たなイベントを発生させられるため，複数の ECA ルールを連鎖的に実行できる．そのため，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 4，**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 を起動するため，R2 と R3 の間に無限ループが発生する．

移動体が持つルールと移動体サーバが持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，保持しているルールの変化などにより，異常動作が発生する．

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．

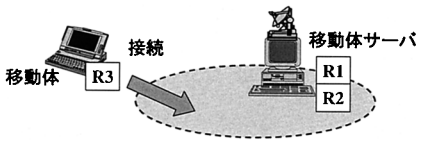

図 4 異常動作の例
Fig. 4 An example of an infinite chain.

表 4 ルールの内容
Table 4 Contents of R1-R3.

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

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[12]．実行時検出は実際に異常が発生してからその検出を行うので，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ったもので，できあ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[1),4)]．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[2),3),6),15)]，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手法として広く用いられている．

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 ルールのホスト間でのやりとりも頻繁に起こるため，複数ホスト間にまたがる 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．実環境では，どのようなルールを持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ことは計算量の大きさから見ても現実的でない．

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．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とりするため，新たな移動体の接続にも対処でき，計算量も削減できる．

以下，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う．

(1) 自ホス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 パスの縮退．
(4) RS パスの送信．
(5) 他ホストでの RS パスの受信処理．

まず，各ホストがそれぞれ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う．ここで異常が検出されなかった場合，ホス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．他のホストではこの 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

自ホス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．一般に，移動体の移動により起こったネットワーク構成の変化を移動体サーバが検出し，移動体サーバ上でアルゴリズムが開始されるが，AMDS では移動できる移動体サーバの存在を許しているため，移動体サーバどうしが接続した場合には両方の移動体サーバでそれぞれアルゴリズムが開始される．
- **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・解除が行われた場合となる．
- **他のホストから RS パス情報を受け取ったとき**
  他のホス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，自ホストのトリガグラフも再構築する必要がある．

4.1.2 自ホストのトリガグラフ作成

自ホス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う．

(1) **パス集合の作成**
自ホス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，アクション実行により満たされなくな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．

(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 5** にトリガグラフ

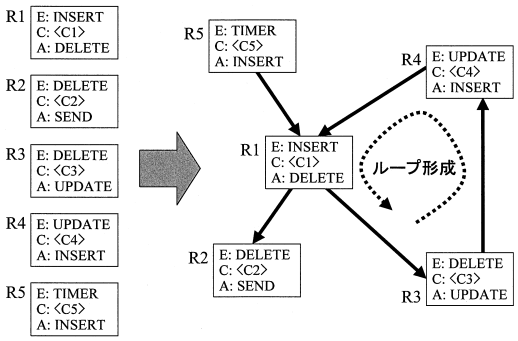

図 5 トリガグラフ構築例
Fig. 5 An example of constructing trigger-graphs.

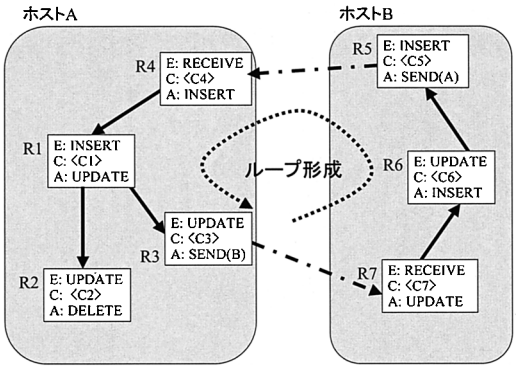

図 6 ホスト間にまたがった連鎖
Fig. 6 An example of chains between different hosts.

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 5 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 6** に示すような，ホス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 イベントの組合せだけである．また，ホス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ホストに 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．自分のホストに RS パスが存在した場合，それがホス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，関連するホストに RS パスの情報を送信することでホスト間にまたがった無限ループを検出する．RS パスを用いることでホスト間でのトリガ情報のやりとりを最小限に抑えられる．

RS パスの検出方法は次のようになる．まず，自ホス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 項で自ホス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 パスの縮退

RS パスは送信先ホストでそのままルールの連鎖パスとして使われ，さらに他のホストに伝播することもあるため，できあがった 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縮退．**
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 項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，ローカルホスト内のテーブルに関するコンディションは，RS パス送信先のホス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．送信先ホス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，そのホストにしかルールの連鎖が起こらないため，対象ホストのみに 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

いるすべてのホストが対象になるため，RS パスも無線通信可能なすべてのホストに送信する．

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．また，ホス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ホス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う．具体的には，ECA ルールが追加・削除された場合，操作後のルール集合に再びアルゴリズムを適用する．移動体が接続したときには，移動体に対して RS パス生成要求を送信し，移動体上でアルゴリズムを開始させる．

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，トリガグラフによって異常を検出した場合，それ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している．

- **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方法である．たと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 データに持つ 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．

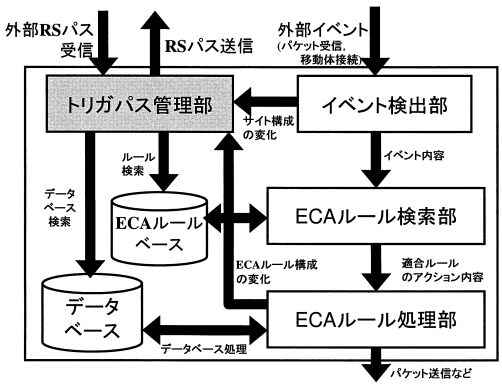

図 7 AMDS のシステム構成
Fig. 7 The system structure of AMDS.

5. 実　　装

3 章，4 章で述べた動的トリガグラフ構築機構を，AMDS 上に実装した．実装は Windows2000 Professional 上で VisualBasic6.0 および VisualC++6.0 を用いて行い，無線通信には赤外線を用いた．動的トリガグラフ構築機構を含む AMDS の構成は**図 7** のようになる．従来の AMDS の各モジュールに加え，新たにトリガパス管理部を実装した．トリガパス管理部では，ECA ルールベース，データベースの内容をもとにホスト内のトリガパスの作成および RS パスの作成を行い，関連ホストに RS パスの送信を行う．イベント検出部で CONNECT/DISCONNECT イベントを検出したとき，および ECA ルール処理部で ECA ルールの追加/削除アクションが実行されたとき，または他のホストから RS パスを受け取ったときにはトリガパス管理部に通知され，トリガパスの再構築が行われる．実装したシステム上でいくつかのルールを実際に動作させ，本手法が正常に動作していることを確認した．

また，多数のホストを想定したシミュレーション実験を行うため，筆者らが開発した AMDS シミュレータ[13] 上にも同様に本機構を実装した．

6. 考　　察

6.1 トラフィック評価

本節では，提案手法を用いることで増加するシステムのトラフィック量の評価を行う．評価は遊園地における待ち時間取得アプリケーションをシミュレータ上に構築して行った．このアプリケーションは，ユーザ

が各施設に近づいたときに，各施設の移動体サーバが自動的に待ち時間情報およびお勧め巡回経路情報を施設情報管理サーバから取得し，結果をユーザに配信するものである．また，ユーザが自発的に施設に関する情報を問い合わせることも可能である．アプリケーションを構成する ECA ルール数は，移動体サーバで 12 個，移動体で 8 個であった．

各施設を表す移動体サーバを環境内に 6 台設置し，同時に存在する移動体の数を 1〜100 まで変化させたときに，提案手法がシステムのトラフィックに与える影響を調べた．評価の比較対象として，RS パスを用いるが，4.1.3 項で述べた RS パスのマージを行わない手法，および RS パスを用いずに，すべてのホストに更新されたトリガ情報を伝播する手法のトラフィックを調べた．また，各ホスト上では，一定の確率で保持するデータベースに対する問合せや更新が発生するとした．

移動体は，500 × 500 マスのフィールド上に配置され，1 ステップにつき 1 マス移動できるとした．本シミュレーションでは，移動体はランダムに選んだ移動体サーバに向かって進み，たどり着いたら一定時間休む，という動作を繰り返すようにした．上記の条件において 10 万ステップのシミュレーションを行いその結果を調べた．結果のグラフを**図 8**，**図 9** に示す．

図 8 は，環境内の移動体数が変化したときの総トラフィック量の変化を示している．総トラフィック量とは，実際にアプリケーションが送受信するデータ量に，異常動作検出に用いるパス情報の通信量を加えたものである．したがって，この総トラフィック量は，実環境において実際に異常動作検出機構を備えたアプリケーションを動作させるときにシステムに発生するトラフィック量を表している．

グラフより，すべてのホストでトリガグラフを持つ従来手法では，ホスト数の増加に対してトラフィック量が指数関数的に増加していることが分かる．これは，従来手法ではトリガの更新情報を接続しているすべてのホストに対して送信しており，ホスト数の増加が，トリガ情報の更新の発生頻度および送信相手の増加の両方に影響するためであると考えられる．提案手法では，RS パスを用いることによるパス情報量の減少だけでなく，関連するホストにしかトリガ情報が送信されないため，トラフィックの増加はほぼ一次関数的増加になっている．ホスト数 100 の場合で見ると，従来手法のトラフィック量は提案手法の 7 倍以上になっており，提案手法の有効性は明らかである．

図 9 は，移動体数を変化させたときに総トラフィックに占めるトリガ情報の割合を示している．従来手法では，かなりホスト数の少ない段階からトラフィック量の大部分をトリガ情報が占めており，異常動作検出のために大量のトラフィックを発生させていることが分かる．RS パスのマージを用いない手法では，従来手法に比べて大幅に効率が改善されているが，ホスト数 100 の場合でトラフィックの 4 割程度をトリガ情報が占めており，さらにホスト数が増大したときに対応できなくなる恐れがある．RS パスのマージを用いる手法では，つねに総トラフィックの 1 割以下となっており，本手法を用いることで発生するトラフィック量がシステム全体にそれほど影響を与えていないことが分かる．

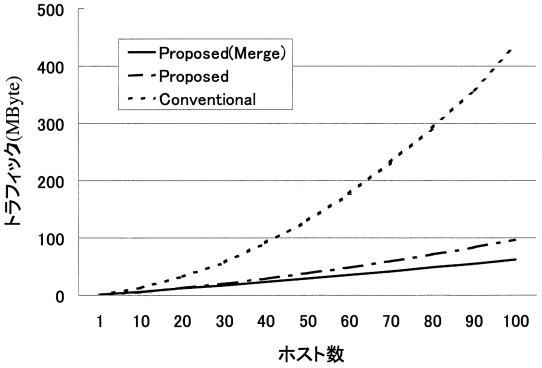

図 8 ホスト数に対する総トラフィックの変化
Fig. 8 Traffic amount in response to the number of mobile hosts.

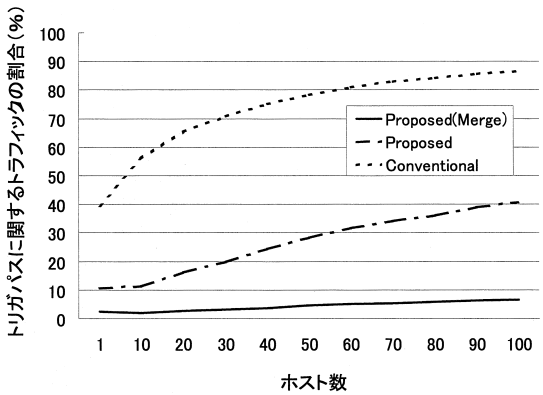

図 9 総トラフィックに占めるパス情報の割合の変化
Fig. 9 Ratio of path traffic in response to total traffic.

6.2 異常動作の検出能力について

本節では，提案手法の無限ループ検出能力について述べる．提案手法はルールの停止性を判定するアルゴリズムであるため，提案手法が正しく停止性を判定できることを証明する必要がある．まず，以下の定理を

示す．証明は文献 9) に示されているものと同様であるため省略する．

［定理 1］任意の ECA ルール集合に対して，なんらかのイベントから到達可能なループがないとき，システムは安全である．ここで，システムが安全であるとは，無限ループによる ECA ルールの無限連鎖が起こらないことを意味する．

定理 1 より，環境内に存在するすべての ECA ルールに対してトリガグラフを構築した場合，そのトリガグラフにループが存在していなければシステムは安全であることが分かる．次に以下の補題が成立する．

［補題 1］ホスト間にまたがるループは必ず 2 つ以上の RS パスの組合せである．

［証明］AMDS のルール言語仕様より，他のホストにイベントを発火させることが可能なアクションは SEND アクションのみであり，他のホストによって発火させられるイベントは RECEIVE イベントのみである．連鎖パスがループを形成していることから，各ホストの連鎖パスは必ず RECEIVE イベントを持つルールで始まり，SEND アクションを持つルールで終わることが示せる． □

［補題 2］ある RS パスを構成するルールのいずれかを基点として，提案する検出アルゴリズムを実行した場合，その RS パスを含むループは必ず検出される．

［証明］RS パスの終端は SEND アクションである．RS パス情報は SEND アクションの宛先に送信されるため，ホスト間の連鎖パスが存在する限り RS パス情報も送信される．補題 1 より，ループは RS パスとホスト間の連鎖パスにより構成されているため，ある RS パスを含むループは存在すれば必ず検出される． □

［補題 3］すべての RS パスは，その RS パスが組織可能になったときに検出アルゴリズムによって評価される．

［証明］連鎖パスが変化するためには，端末内でルール構成が変化するかあるいは端末間の接続関係が変化したときである．4.1.1 項で述べたとおり，どちらの場合においてもそのタイミングを基点として検出アルゴリズムが実行されるため，すべての RS パスが評価される． □

［補題 4］RS パスの縮退が，ループ検出間違いを発生させることはない．

［証明］ループ検出アルゴリズムは，ループがあるかないかを検出するためのもので，途中のパスには関連しない．RS パスの縮退はループの接続状態を変化させないので，RS パスの縮退がループ検出間違いを引き起こすことはない． □

これらの定理・補題より，次の定理が示される．

［定理 2］提案する検出アルゴリズムによってループ検出されなければシステムは安全である．

［証明］定理 1 および補題 1〜4 より，各端末内で完結するループは端末内のトリガグラフにより検出され，端末間にまたがるループは RS パスによってすべて検出されることが分かる． □

定理 2 より，提案アルゴリズムにおいて安全であると判断された場合，システムの安全性が保障されることが分かる．一方，提案アルゴリズムにおいてループが検出された場合でも，実際にはシステムが安全である場合がある．そのような例として次のような場合が考えられる．

- 検出されたループが無限ではなく，一定回数ループすれば必ず停止するとき．
- ルールの無効化や削除アクションによって，ループが実際には成立しないとき．

前者は，たとえばルール r_1 → ルール r_2 → ルール r_1 というループが存在し，r_1 のコンディションが「変数 x の値が一定値以上」とされていて，r_2 のアクションが x の値の減少，かつ x を増加させるアクションが存在しない場合が該当する．この場合，x の値はループが実行されるたびに減少し，いつかは r_1 のコンディションを満たさなくなる．そのため，このルールセットは無限ループではなく安全だと判断できる．後者は，たとえばルール r_3 → ルール r_4 → ルール r_3 というループが存在し，r_4 のアクションが r_3 を無効化する場合が該当する．

前者のパターンの検出に関しては，Lee らによって複数回ループを展開する手法が提案されている[8]．また後者のパターンの検出に関しては，Vaduva らによってアクション–ルール間関係を用いた手法が提案されている[14]．これらの手法を採り入れることで本研究におけるループ検出の精度を高められると考えられるが，これらの手法は ECA ルールの言語仕様に大きな制限を加えており，そのまま本研究に適用できない．たとえば Vaduva らの手法を適用できるのは，有効化・無効化されるルールがアクションの中で明示的に指定されている場合に限られる．そのため，ルールを明示的に指定するだけでなく，データベースから得た内容や NEW データ，OLD データの内容によるルール指定など，静的にルール指定が決まらない本システムには適用できない．

このように，本システムにおいては検出されたループが本当に無限ループであるかどうかは保障できず，さらなる判定精度の向上は今後の課題であるが，上記

の手法を部分的に採用して無限ループの判断精度を向上させるといった方法が考えられる．

6.3 停止性判定中のホスト移動について

これまで，簡単化のために本アルゴリズムにおける停止性判定中にネットワーク構成の変化やルールの追加・削除といった環境の変化が起こった場合については言及していなかった．本節では停止性判定中における環境の変化の影響について述べる．

環境が変化した場合には 4.1.1 項で述べたように，ルールの追加やホストの接続を検出したホストから新たにアルゴリズムが開始されるため，検出能力が落ちることはない．このとき，それまで実行されていたアルゴリズムのインスタンスは，すでに他のホストへ伝播している場合など停止させることが困難であるため，そのまま並行して実行される．継続して実行されるアルゴリズムが環境の変化により影響を受けるのは下記の場合である．

- 変化する前の環境ではループが存在していなかったが，環境が変化した結果ループを検出する場合．
- 変化する前の環境ではループが存在していたが，ホストが退出するなどの結果ループが成立しなくなる場合．

前者の場合，実際にループが存在するため，そのループをアルゴリズムが検出することに問題はない．一方，後者の場合はすでに RS パス中に組み込まれているホストが退出してしまった場合など，実際にはループが成立しなくなった環境においてもループがあると検出する可能性がある．しかし，退出直前の時点ではループが存在していたことから，その状況に対してループ検出をユーザに通知する本アルゴリズムの処理に問題はないといえる．したがって，停止性判定中に環境の変化が起こった場合も提案アルゴリズムは有効に動作する．

6.4 無限ループ以外の原因による異常動作について

本研究で検出する異常動作は ECA ルールの連鎖実行による無限ループであるが，ECA ルールによるその他の異常動作として，次のような例があげられる．

- ECA ルール生成アクションによるルールの無限生成．
- 継続的な外部からのイベント発生によるルールの無限実行．

本システムではルール生成アクションを用いて，データベースに格納されたルールをシステムに追加するといった記述ができる．この機能により，システムが無限にルールを生成し続ける可能性がある．また，頻繁かつ継続的に外部からデータベース操作などが行われることで，ルールの処理速度より発火されるべきルールの数がつねに多い状況となり，ルールの無限実行状態に陥る場合がある．

これらの異常動作は ECA ルールセットが原因ではなく，システムによるルール数の上限設定や，端末の能力を高めることで回避する問題であるため，本研究では取り扱わない．また，これらの異常動作が起こる状況であっても，提案アルゴリズムはルールの連鎖実行による無限ループを検出できる．

7. 関連研究

近年のトリガグラフの研究としては，トリガグラフの検出能力を高める研究がある[2),3),6),8),15)]．従来のトリガグラフでは，イベントとアクションのみによってトリガグラフを構築していたが，このような方法では実際は安全であるルール群も異常動作として判定してしまう．そこで，これらの研究ではルール中のコンディション部を考慮することによって，異常動作検出の精度を高めている．文献 15) ではコンディションとアクションの記述に強く制限を加えることで，コンディションを考慮したトリガグラフを実現し，文献 3) では代数的アプローチによりコンディションを加えたトリガグラフを構築している．文献 2)，文献 6) では一度トリガグラフを構築してから，トリガグラフの各エッジについてのコンディションを評価して，連鎖的に実行されないエッジを取り除いている．文献 8) では，トリガグラフ上でループとなった部分グラフを展開することで，無限ループなのか複数回ループして止まるのかを判定している．また，文献 7) では，トリガグラフの各エッジごとに評価するのではなく，エッジをいくつか連結した意味のあるパスごとにコンディション評価を行うことで検出精度を高めている．

また，従来のトリガグラフ構築アルゴリズムでは取り扱えなかった高度な ECA ルール言語仕様に対応したアルゴリズムとしては，Vaduva らの手法がある[14)]．文献 14) では，複数事象の同時あるいは連続発火をイベントとして記述できる混成イベントや，他のルールを有効化・無効化するアクション，イベントを発火できるコンディションなど，より複雑な ECA ルールが記述できるアクティブデータベースにおけるトリガグラフ構築手法を提案している．

これらの研究は，システムに存在するすべての ECA ルール内容が明らかになった状態でのトリガグラフ構築と無限ループ検出に関する研究であり，ホスト間の連鎖を効率的に検出するための手法を提案している本研究とは共存できる．本研究の提案手法におけるロー

カルホストでのトリガグラフ構築部分にこれらの関連研究で用いられている手法を採用することで誤検出を減らすことが可能になると考えられる．

モバイル環境におけるアクティブデータベースの異常動作検出に関する研究には，マージトリガグラフを用いた異常動作検出に関する研究がある[9]．提案されているマージトリガグラフを用いれば AMDS において静的検出を行えるが，マージトリガグラフの構築には非常に高いコストが必要である．また，マージトリガグラフを構築するためにはあらかじめすべてのホストおよびルールの内容を知っておく必要があるが，3章で述べたように一般に移動体計算環境においてその仮定は現実的ではなく，あらかじめホストやルール内容を知っておく必要のない直前検出手法が有効であるといえる．

8. 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．また，シミュレーション評価により，本手法が従来手法に比べて大幅にトラフィック量を削減でき，本手法がシステム全体のトラフィックに与える影響が小さいことを確認した．今後の課題としては，さまざまなタイプのアプリケーションにおける本手法の効率の評価および実機での実測評価があげられる．

謝辞 本研究は，日本学術振興会若手研究（B）（13780331）および文部科学省振興調整費「モバイル環境向 P2P 型情報共有基盤の確立」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) Aiken, A., Widom, J. and Hellerstein, J.M.: Behavior of database production rules: Termination confluence and observable determinism, *ACM SIGMOD International Conf. on the Management of Data*, pp.59–68 (1992).
2) Baralis, E., Ceri, S. and Paraboschi, S.: Run-Time Detection of Non-Terminating Active Rule Systems, *DOOD*, pp.59–68 (1992).
3) Baralis, E. and Widom, J.: An Algebraic Approach to Rule Analysis in Expert Database Systems, *20th VLDB Conf.*, pp.475–486 (1994).
4) Ceri, S. and Widom, J.: Deriving Production Rules for Constraint Maintenance, *16th VLDB Conf.*, pp.566–577 (1990).
5) 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).
6) Karadimce, A.P. and Urban, S.D.: Refined Trigger Graphs: A Logic-Based Approach to Termination Analysis in an Active Object-Oriented Database, *ICDE '96*, pp.384–391 (1996).
7) Lee, S.Y. and Ling, T.W.: A Path Removing Technique for Detecting Trigger Termination, *EDBT*, pp.341–355 (1998).
8) Lee, S.Y. and Ling, T.W.: Unrolling Cycle to Decide Trigger Termination, *25th VLDB Conf.*, pp.483–493 (1999).
9) 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).
10) Murase, T., Tsukamoto, M. and Nishio, S.: A system Platform for Mobile Computing base on Active Database, *International Symposium on Cooperative Database Systems*, Vol.2, pp.424–427 (1996).
11) Murase, T., Tsukamoto, M. and Nishio, S.: Active Mobile Database Systems for Mobile Computing Environments, *IEICE Trans. Info. and Syst.*, Vol.E81-D, No.5, pp.427–433 (1998).
12) 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 99-DBS-119, Vol.99, No.7, pp.369–374 (1999).
13) 寺田 努，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シミュレーション環境について，情報処理学会研究報告 2001-DBS-125 (1), Vol.2001, No.70, pp.351–358 (2001).
14) Vaduva, A., Gatziu, S. and Dittrich, K.: Investigating Termination in Active Database Systems with Expressive Rule Languages, *Rules in Database System '97* (*LNCS 1312*), pp.149–164 (1997).
15) van der Voort, L. and Siebes, A.: Termination and confluence of rule execution, *2nd International Conf. on Information and Knowledge Management* (1993).

(平成 14 年 6 月 19 日受付)
(平成 14 年 9 月 30 日採録)

（担当編集委員 石川 博）

寺田　努（正会員）

平成 9 年大阪大学工学部情報システム工学科卒業．平成 11 年同大学大学院工学研究科修士課程修了．平成 12 年同大学院退学．同年より大阪大学サイバーメディアセンター助手．平成 14 年より大阪大学大学院情報科学研究科マルチメディア工学専攻助手を併任．現在に至る．アクティブデータベース，モバイルコンピューティング，データ放送の研究に従事．

塚本　昌彦（正会員）

昭和 62 年京都大学工学部数理工学科卒業．平成元年同大学大学院工学研究科修士課程修了．同年シャープ（株）に入社，同社研究員．平成 7 年大阪大学工学部情報システム工学専攻講師．平成 8 年より同専攻助教授，平成 14 年より大阪大学大学院情報科学研究科マルチメディア工学専攻助教授．現在に至る．工学博士．モバイルコンピューティング，分散知識ベースシステムの研究開発に従事．ACM，IEEE 等 8 学会各会員．

西尾章治郎（正会員）

昭和 50 年京都大学工学部数理工学科卒業．昭和 55 年同大学大学院工学研究科博士課程修了．工学博士．京都大学工学部助手，大阪大学基礎工学部および情報処理教育センター助教授を経て，平成 4 年より大阪大学大学院工学研究科教授，平成 14 年より大阪大学大学院情報科学研究科マルチメディア工学専攻教授となり，現在に至る．平成 12 年より大阪大学サイバーメディアセンターを併任．この間，カナダ・ウォータールー大学，ビクトリア大学客員．データベース，知識ベース，分散システムの研究に従事．現在，Data & Knowledge Engineering，Data Mining and Knowledge Discovery，VLDB Journal 等の論文誌編集委員．本学会フェロー含め ACM，IEEE 等 8 学会各会員．