Vol.2009-ARC-183 No.28
Vol.2009-OS-111 No.28
2009/4/24

Remote Editing Protocol の実装と検証

与 儀 健 人[†1] 宮城健太[†1] 河 野 真 治[†2]

本研究室が提案する RemoteEditingProtocol(REP) は、異なるホストにあるアプリケーション同士による協調したデータ編集を可能にするプロトコルである。しかし REP は複雑なプロトコルを用いているため、その検証による動作の保証が不可欠となる。本研究では REP の主要なプロトコル部を抜き出し、JavaPathFinder を用いてその検証を行う。

Implementation and verification of Remote Editing Protocol

Kento YOGI,[†1] Kenta MIYAGI[†1]
and Shinji KONO [†2]

Remote Editing Protocol what we have suggested makes applications possible to concertedly edit any data with one another. Because this protocol is very difficult, verification of the progoram is important. So in this paper, we extracted the core program of the protocol and verified it using JavaPathFinder.

1. はじめに

本研究室では、vim、Emacs、Ecplise を相互接続するプロトコルを提案して来た。今回は、Session Manager を導入することにより、より単純なユーザインタフェースを実現するとともに、複雑なプロトコルを Session Manager 側に閉じ込めて、Editor 側の実装の手間を軽くすることを提案する。

一方で、プロトコル自体がかなり複雑になったので、プロトコルの正しさ及び、プロトコル実装の正しさを検証する必要が出て来た。プロトコル検証では、Java PathFinder[3)] の有効性が知られているが、それを用いるために、ソケット通信を Thread 間の同期で実現するライブラリを作成した。また、Editor 側の実装の正しさの検証及びデバッグのために、テスト用の Editor を作成した。

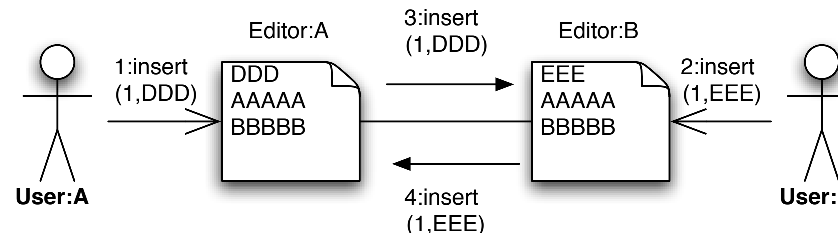

図 1 REP での相互接続

2. Remote Editing Protocol の設計方針

複数人が同じテキストを共有して編集するプロトコルは、さまざまなものが提案されているが、汎用エディタに実装する前提のプロトコルはほとんどない。Remote Editing Protocol では、複数の Session Manager と、リング状の Session の上に編集コマンドを循環させる方法を取っている 。

この方法を採用した理由はいくつかがある。集中サーバを用いない 分散実装 が一つの前提になっている。Session Manager 自体が分散していて、Session Manager は、(分離された Merger を除けば) 編集コマンドを中継するだけである。また、既存のエディタを用いるために、local な編集 を妨げない点を重視している。遠隔/共有編集を実現することによって、本来の編集機能が速度低下などにより損なわれることはない。一度に大量の通信をすることなどを避け Network 負荷が軽い こと。複雑なコマンド入力などのない Simple なユーザ Interface。これらを実現するために Conflict を非同期に解決 し、変更の伝播の遅延は容認する。また、小人数向け の共有とする。遅延を容認するために、遠距離でも使用可能 となる。また、オープンソースとして実装し、教育用途 に向いている。特に、XP (eXtreme Programming)[7)] に

†1 琉球大学理工学研究科情報工学専攻
Interdisciplinary Infomation Engineering, Graduate School of Engineering and Science, University of the Ryukyus.
†2 琉球大学工学部情報工学科
Infomation Engineering, University of the Ryukyus.

© 2009 Information Processing Society of Japan

おける Pair Programming での使用を意識しているので、Emacs/vim/Eclipse の相互接続を重視する (**図 1**)。将来的には、動的な変更を可能とする Inter-Application Protocol として使えるものを目指している。プロトコル自体の信頼性を増すために、プロトコル自体の正しさ、及び、実装の正しさを調べることを可能にする。

3. Protocol の構成

ここでは、REP を Session manager(SM), Session manager 接続プロトコル、Session 接続プロトコル、Editor Command、Merge プロトコル、Merge の Session Manager への移動の順に説明する。

3.1 Session manager の導入

従来の REP はエディタ間を直接結んでいたが、その場合は相手のエディタのホスト名やファイル名を直接入力する必要があった。これは、ユーザに取って繁雑なだけでなく、個々のエディタでの実装に複雑な UI を含める必要がある。

Session manager(SM) はエディタの動作するホストに一つあり、エディタは自動的に決まったポートを通して SM に接続 (join/put) する。このようにすれば、エディタ上でホスト名を入力する必要はない。一つのホスト上では、単一の SM に複数のエディタが接続する。離れたホスト同士のエディタを接続する場合は、まず、それぞれのホスト同士の SM を接続する。そして、それぞれエディタが SM に接続した後で、ホスト間の接続を選択 (select) する (**図 2**)。

3.2 Session manager の接続 protocol

SM 同士の接続は、`sm_join` コマンドを SM に送ることによる (**図 3**)。接続により、接続した SM 間で unique な session manager id が決められる。SM 同士の接続は木構造 (SM 木) になるようになっており、唯一の master SM が存在する。

同時に相互に `sm_join` が発行される場合もあるので、リングを避けるために、`sm_join` は master SM まで転送される。自分自身に `sm_join` が戻って来た場合は、その `sm_join` は廃棄される。現在は、既に session/editor を持つ SM は、他の SM に接続することは出来ない。

3.3 Session 接続 protocol

SM に接続したエディタは、自分の既にオープンしたファイルを持って接続する `put` と、他のエディタへ空のバッファを接続する `join` の二種類の接続を行なう。

接続が行なわれると、SM から editor id を ACK として受け取る。editor id は、session

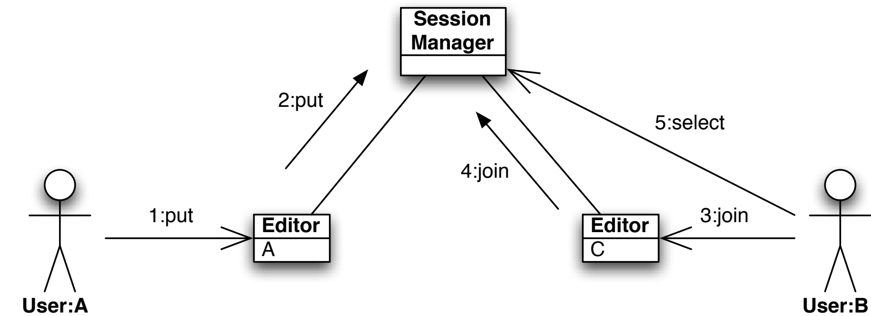

図 2 Session Manager の導入

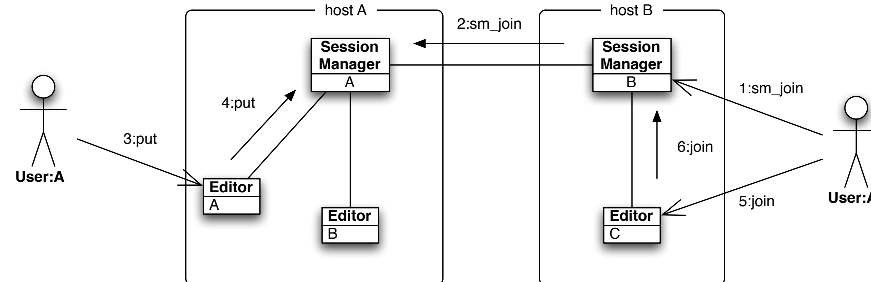

図 3 Session Manager 同士の接続

manager id (SM id) を含んでおり、全ての SM 上でユニークとなる。

`put` したファイルを持つエディタは、その session の master となる。ファイルを共有する editor 群を session と呼ぶ。session には、SM id を含む session id が割り振られ、全ての SM 上でユニークとなる。

ユニークな SM id を使うので、editor id/sesison id は master SM に問い合わせることなく生成が可能となる。

`put` されたファイルは SM 木を `put}` コマンドで遡り、`put_ack` によって、すべての `SM` に通知される。このファイルの編集に参加したい場合は、まず、`Editor` を空のバッファの状態で `SM` に `join` コマンドで接続する。すると、`put` と同様に `join` した `Editor` がすべて

© 2009 Information Processing Society of Japan

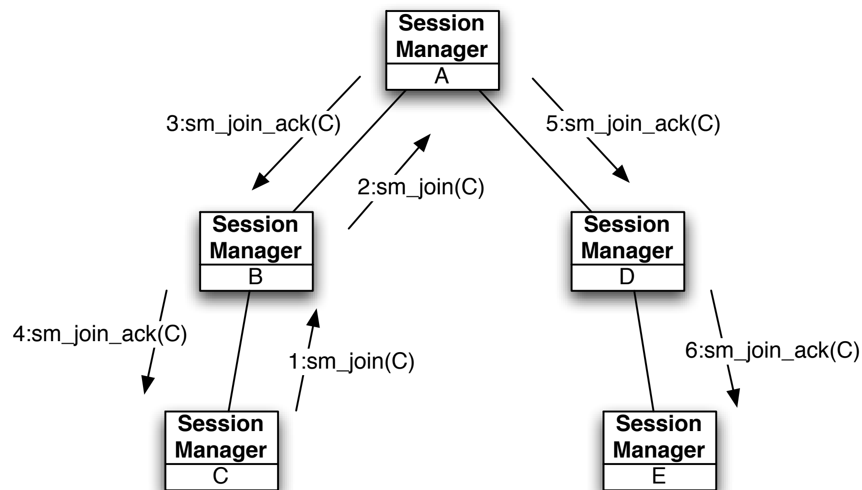

図 4 join コマンド

の SM 上に通知される。SM の GUI 上の操作 select により、put されたファイルと join した Editor が結びつけられる（**図 4**）。

select 操作では、join した Editor と put したエディタを探し出す必要がある。そのために、SM 木上に SM 同士に到達するための routing table を構築している。これは、sm_join 時に作成される。まず put した Editor を探し、見つかったら select_ack を、session ring を構築しながら join した Editor を探す。見つかったら join_ack が Editor に返される。この時に、必要があれば、join 側、put 側の認証を行なう。

join したエディタは空のバッファを持っているので、Session master（put した Editor）に、必要な編集行を要求する sync コマンドを session ring に送る。Session master は、次の Editor Command を使って必要な行を送信する（**図 5**）。

3.4 Editor Command

Editor のコマンドは、すべて、insert,delete に分解される。SM 上での混乱を避けるために、Editor が直接 SM に送ったユーザが生成した Editor Command user_inert, user_delete と SM 経由で送られた他の Editor Command は異なるコマンドとして扱っている。

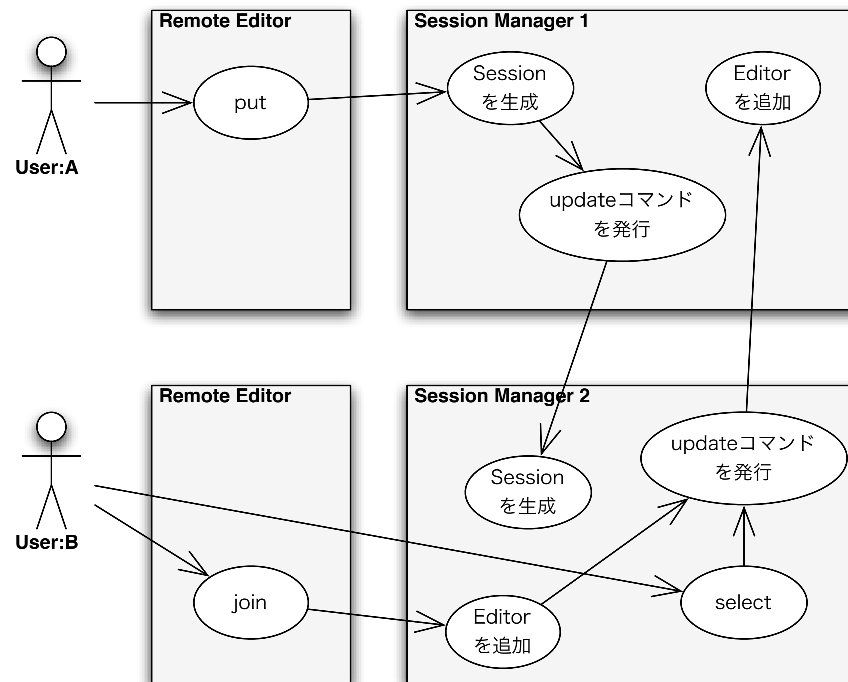

図 5 Session Manager を介したエディタの接続

Editor は複数の session を持つことも可能であるが、一つの Editor が同じ Session に複数回 join すると、Editor の通信経路と Editor id が対応しなくなる。問題はないが、実装はより複雑となる。

次の Merge Protocol では、SM 上で Editor のコマンドの undo を計算する必要があるので、user_delete には、削除した行の内容が付加されて SM に送られる。したがって、user_delete と user_insert と見掛け上対称となる。

全文置換なども user_inert, user_delete に分解する必要があり、その分解は Editor によって行なわれる。REP は歴史的理由で行指向のプロトコルであり、行指向でない Editor でも行番号を付加する必要がある。

sync に対しては、要求された行に対して、delete, insert を順に送ることで、join した Editor に行を転送する。特別なバッファ転送コマンドはない。

© 2009 Information Processing Society of Japan

置換を特別扱いすることによるコマンド短縮の利点があるように思えるが、SM では undo を生成する必要があるので、変更前の行と変更後の行を送る必要があり、delete, insert を順に送る場合との差は無視できる。

3.5 Merge Protocol

一つの Session の上で、複数の Editor が同時に編集を行なった場合には、その結果は、最終的に、Session 上で同じになる必要がある。

REP では、二つの Editor の場合の編集の衝突の解決を行なう手法を提案して来た。この方法 (Merge Protocol (A)) では自分の Editor Command を相手に送り、戻って来るまでの Editor Command をキューに入れておく。他の Editor の Command を受け取った時には、そのキューと、その Command の可換性を調べて、キューを変更する[6)]。しかし、この方法は、三つ以上の Editor の場合はうまく動作しない。

そこで、以下のような Merge Protocol (B) を導入する。(1) Editor Command を Session Ring 上に流し、それが戻って来るまでに、他の Editor から受け取った Editor Command をキューに入れておく。(2) 戻って来たタイミングで、キュー上の Editor Command を、eid と Command の順序を基にソートする。(3) Editor Command がなくても、他の Editor から Command を受け取ったら、NOP Command を生成して、それが戻って来た時にソートを行なう。

この手法では、Editor が N 個ある Session の場合、一つの Editor Command に対して、N-1 個の NOP Command が生成される。

そこで、以下のような Merge Protocol (C) を導入する。(1) Editor Command を Session Ring 上に流し、それが戻って来るまでに、他の Editor から受け取った Editor Command をキューに入れておく。(2) 戻って来たタイミングで、キュー上の Editor Command を、eid と Command の順序を基にソートする。(3) 他の Editor にソートのタイミングを与えるために、Editor Command の ack を、もう一周させる。(4) 他の Editor の Command を受け取ってから、ack が来るまでの Command をキューに入れておき、ack が来たら、eid と Command の順序を基にソートする。

Editor には、ソートした編集結果になるように、それまで行なった編集を Undo して、ソートした編集結果を適用する。

3.6 Merge の Session Manager への移動

Merge Protocol は、かなり複雑であり、すべての Editor 実装に対して実装する必要がある。我々のターゲット (Vim, Emacs, Eclipse) は、すべて異なる言語 (C,Emacs Lisp,Java) で実装されており、そのすべてで、複雑なプロトコルを実装するのは不可能ではないが、コストがかかる。

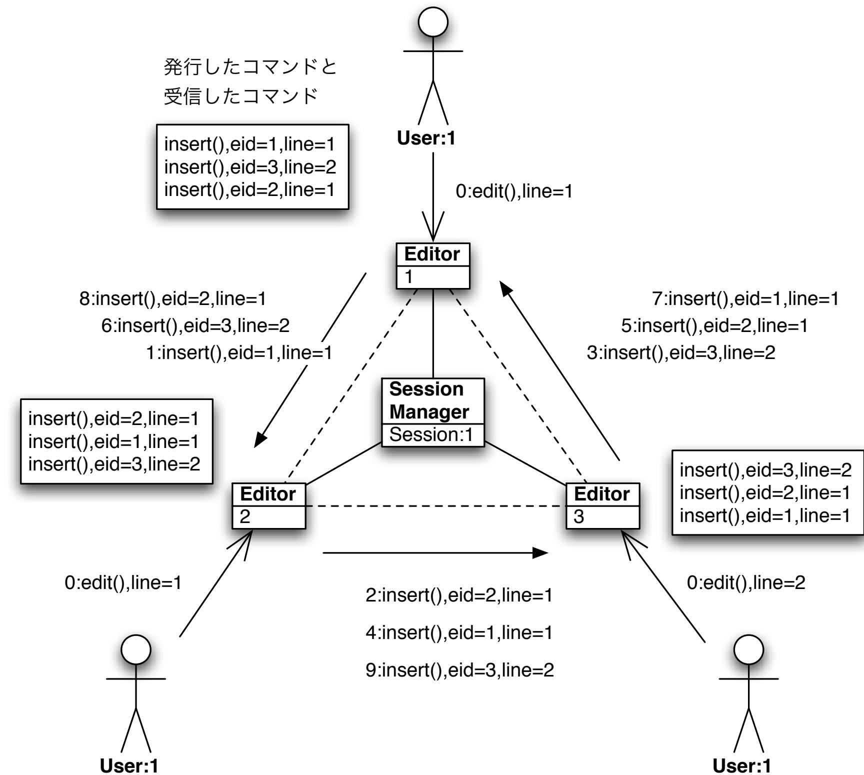

図 6 Session Ring 上の REP コマンドの送信

今回は、SM が一つの Editor に対して必ず存在するので、Merge Procotol を SM に実装すると、SM の実装言語 (Java) に実装するので十分になる。しかし、Merge Protocol は編集バッファに対して複雑な操作をするので、それを Editor Command を通して実装する必要がある。

まず、Editor Command が Undo(取消し) 可能でなければならない。このために、

ⓒ 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.28
Vol.2009-OS-111 No.28
2009/4/24

user_delete Command に削除した行の内容を付加することにした。

次に、SM が Merge Protocol でソートした編集結果を適用した結果は、(可能な最適化をした)Editor Command 列で Editor に反映する必要がある。この時に、ユーザが編集コマンドを割り込ませる可能性がある。

これを防ぐ一つの方法は、Merge 作業が始まった段階で、ユーザ入力を block してやれば良い (a)。もう一つの方法は、ユーザ入力の割り込みがあった場合は、その入力込みで、もう一度、ソートを実行すれば良い (b)。これはリマージと呼ばれる (**図 7**)。

Merge 作業中には、他の SM/Editor からの入力を block することは問題ない。それは、もともと非同期で動作しており、遅延は許容されるようになっている。

ユーザ入力の lock(a) は、エディタの実装に依存していて、実装はさまざまである。また、REP 設計の一つの目標である local な編集を妨げないという点では問題が残る。(b) は、Merge Protocol の実装が繁雑になるが、ユーザの入力を妨げることはない。また、エディタの lock を実装しなくてすむ。(a),(b) はお互いに干渉しないので、エディタの lock の実装は REP を実装する時の選択肢の一つとなる。lock がある方が大量の変更 (コピーペースト) がある場合にスムースな動作が期待できる。

4. Protocol の正しさ

Merge Protocol の正しさの証明は、Protocol 自体の正しさと、Protocol の実装を含めた正しさの二種類の正しさを示す必要がある。

ここでは、(A) の Protocol の正しさを示す。Editor $0..n$ が、それぞれ、編集コマンド C_{ij} (Editor i の j 番目のコマンド,j は 0 から) を入力したとする。このコマンドは、Session ring を巡回する。巡回するたびに、各 Editor k が NOP Command N_{kx} を、そのコマンドの前に付加する。x は、コマンドの順序である。

Editor m では、

$$C_{m0}C_{x0}N_{00}....N_{yz}$$

などのコマンド列が実行されることになる。これを C/N の区別のないコマンド記号 (E_{ij}) で置き換えよう。

$$E_{m0}E_{x0}E_{00}....E_{yz}$$

NOP の付加手順から、他の Editor が送った Command には、その前の他の Editor からの Command を受け取った後の、自分が送った Command(0 以上の複数個) または、NOP が必ず対応している。対応する Command とは、Session ring 上で同時に実行されたと考えられる Command の集合と考えて良い。Command は Session ring を一周するので、すべての Editor が同じ Command の集合を受け取っている。

ここで、対応した Command の集まり毎に列を分割し、Editor i のその集まりを集合とみなし S_{ij} とする。この集合の列を Z_i とする。

$$Z_i = S_{i0}S_{i1}....S_{in}$$

定義から隣同士の S_{ij} には、対応した Command が含まれることはない。

この集合列 Z は、すべての Editor で同一となる。[証明] Editor i と Editor j で、S_{ik} と、j の S_{jk} が異なっているとしよう。ある Command E_s があって、$E_s \in S_{ik}$ であって $E_s \in S_{jk}$ でない。しかし、E_s は session ring を一周しているので、S_{ik} と S_{jk} の両方に含まれているか、隣同士にあるはずである。両方に含まれているとすると仮定と矛盾するので、隣同士になるはずである。しかし、隣同士であれば、Command の分割の方法に矛盾する。[証明終り]

従って、S_i を Editor id と Command 順序によってソートしてやれば、すべての Editor で、同一な Command 列を得ることが出来る。ソートのタイミングは、対応するコマンドがすべて自分に到着した時点である。自分の送った新しいコマンド、または、新しい NOP が来たことによって、その一つ前までが対応しているものだということがわかるので、その時点でソートしてやれば良い。従って、Merge Protocol により、すべての Editor で同一の編集結果が得られることがわかった。

(B) では、NOP の挿入の代わりに ack を、もう一周させている。ack が来た時点で、対応する Command の集合が確定する。あるいは、仮想的に NOP を挿入したと考えると、その NOP は、ack の前に挿入されていると考えて良い。従って、(A) と同じように集合列 Z を、すべての Editor で同一となるように決めることが出来る。

プロトコルの実装の正しさは、実装言語である Java に深く依存するので、このように簡単に証明することは出来ない。そこで、モデル検査器である Java PathFinder[3)] を用いる。

5. Protocol の実装

Session Manager は Java 1.5 で実装されており、 Ecplise は Java による Plug-in となっている。 Emacs は、従来は C で書いたクライアントを接続していたが、今は、すべて Emacs Lisp で書かれている。 Vim は、C で記述されており、Merger も C で記述されていたが、今回の実装で取り外された。

今回の実装では、Editor 側の実装のコストが削減されており、Merge Protocol 部分

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.28
Vol.2009-OS-111 No.28
2009/4/24

で C で 150 行程度が削減されている。Editor 側の実装は、Editor Command を insert, delete に分解する部分の実装が大半である。

Editor 側で実装する必要があるのは、表 1 の機能である。

表 1 Editor 側での実装

1	編集機能の user_insert,user_detele への分解と、分解した Editor Command の送信
2	join,put Command の UI 部分と、Command の送信
3	join,put Command の UI 部分と、Command の送信
4	join_ack,put_ack の受け取りと sid,eid の設定
5	外部からの Editor Command の非同期受け取りと実行
6	sync Command を受け取った場合の user_insert,user_detele の生成
7	Merge 時の lock (optional)
7	quit Command

ファイルのオープン/セーブなどの機能は REP には含まれていない。Master Editor も、それ以外の Editor も任意の時点でのローカルなセーブが可能である。版管理なども、REP 以外の部分で対応する。

外部からの Editor Command を受け取った場合のカーソルなどの制御は、規定されていない。移動した場合が便利な場合と、そうでない場合があると思われる。

6. Socket Simulator

Java Pathfinder による検証を行なうために、直接、Socket を経由せずに、Thread を通して通信を行なうライブラリを作成した。

このライブラリは、最初、java.nio 互換ではなく作成し、Merge Protocol (A) 及び (B) の動作を Java PathFinder で確認した。編集結果の同一性を調べるために、Session 内の Editor の quit プロトコルを実装している。

まず、 quit Command が Sessin ring に送られ、各 Editor は、quit を受け取ると、自分のユーザ編集コマンドを停止する。その編集を終了した後、quit を返す。quit を受け取った後も、他の Editor からの Editor Command は来る可能性がある。quit を最初に送った Editor に quit が戻ると、今度は quit2 を Session Ring に流す。これより後に、ユーザ編集コマンドが来ることはない。Editor は自分の待っている Editor Command の ack がすべて来るのを確認してから、quit2 を次へ渡す。quit2 を渡した後は、Editor Command は来ないので、終了して良い。最初の Editor へ quit2 が戻った時点で、すべての編集が終了する。

この時に、Editor のバッファを比較して、すべての Editor のバッファが同じならば、正しくプロトコルが動いたことになる。これを Java PathFinder で検証を行なう。

(C) の実装を、実際の Session Manager 上で検証するために、java.nio と Thread による通信のシミュレーションを切替えることが可能なライブラリを作成した。実際の Session Manager に対する Java PathFinder での実行確認は、計算時間の制約により、まだ、可能とはなっていない。

7. 検証とデバッグ

(A) のプロトコルが Editor 二つで動作すること、及び、(B) のプロトコルが複数の Editor で動作することを Java PathFinder で確認することが出来た。

(C) は、実際の Session Manager の実装を含む検証となるので、よりハードルが高い。現状では、Java PathFinder での動作確認は出来ていない。

Java PathFinder で vim や Emacs を含む検証は可能とはなっていないが、Sample Editor を Java で実装することにより、Java PathFinder での Merge Protocol の検証が可能となっている。

実際には、vim や Emacs などの Editor 側の実装が正しいかどうかを調べることも重要である。それは、Merge Protocol を切った状態で、Java の Sample Editor に対する動作を確認することで調べることが出来る。

8. 比　　較

類似の Project としては、GroupKit[4], Soba Project[1] がある。vim や Emacs などの Open source editor の実装を含むのが、REP の特徴である。

また、Java で実装されていて、Session Manager 部分、Editor の改変部分、Eclipse plugin のすべてが、LGPL で公開されているのも独自な特徴の一つである。

GroupKit は tcl/tk で記述されており、検証などが困難だが、REP では、Java の部分を Java PathFinder で検証することが可能だと思われる。しかし、現状では、まだ、検証までには至っていない。

GroupKit などで使われているマルチメディア編集の同期は、Master が一つ存在し、それに対する Command の発行と、Master からの Command のマルチキャストで実現されている[2]。REP では、マルチキャストではなく、Session ring によって同期を実現している。

© 2009 Information Processing Society of Japan

Vol.2009-ARC-183 No.28
Vol.2009-OS-111 No.28
2009/4/24

Ring は、遅く信頼性に欠ける部分があるが、ネットワークに対する負荷が軽いと言う特徴がある。(C) の Merge Protocol を使うことにより、o(n) のパケットで同期を行なうことが出来る。また、マルチキャストを避けているので、WAN などの遅延が大きい部分に複数のストリームを張る必要がないという特徴がある。

また、Session Manager 上には、Editor Buffer が存在しないので、大きなファイルを編集する場合でも、Session Manager のメモリを消費することはない。

9. 最後に

このプロジェクトは、sourceforge を通じて公開[5] されており、まだ、開発途上となっている。

残念ながら、実際の Session Manager 上での Java Pathfinder での検証はまだ、出来てはないが、通信ライブラリ上での処理を atomic にするなどの工夫で可能になると期待している。

参考文献

1) . SOBA Project, March 2004.
2) C. A. Ellis and S. J. Gibbs . Concurrency control in groupware systems. 1989.
3) K.Havelund and T.Pressburger. Model checking java programs using java pathfinder, 1998.
4) Mark Roseman and Saul Greenberg. *Building Real Time Groupware with GroupKit, A Groupware Toolkit*. 1996.
5) Shinji KONO. rep, Aug 2006.
6) 安村 恭一 and 河野 真治 (琉球大). 巡回トークンを用いた複数人テキスト編集とセッション管理. 情報処理学会システムソフトウェアとオペレーティング・システム研究会, June 2004.
7) 長瀬嘉秀. *eXtreme Programming*. 日本 XP ユーザグループ関西支部, 2002.

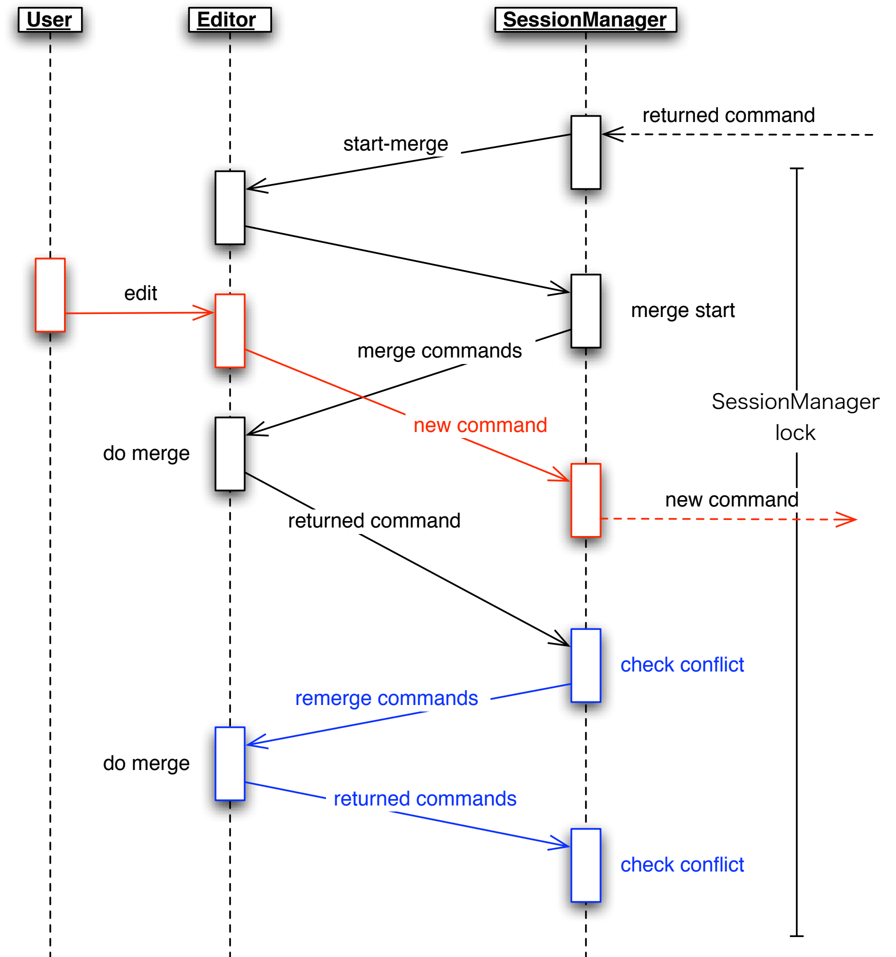

図 7 リマージ

ⓒ 2009 Information Processing Society of Japan