社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE

複数名技術者による同時編集作業における競合の抑制と並列化の両立

梶原 直人† 杉山 安洋‡

† 日本大学大学院工学研究科 〒963-8642 福島県郡山市田村町徳定字中河原 1
‡ 日本大学工学部 〒963-8642 福島県郡山市田村町徳定字中河原 1
E-mail: † kajiwara@ssl.ce.nihon-u.ac.jp, ‡ sugiyama@ce.nihon-u.ac.jp

あらまし ソフトウェア開発においては，複数のソフトウェア技術者による作業の競合を抑えることが重要であるが，一方で作業の並列性を確保することも重要である．我々の研究室では，これまで作業の競合を抑えるメカニズムとして細粒度ロックを研究してきたが，開発作業の並列性という意味では不完全な部分が多かった．そこで，細粒度ロックに複数名の技術者によるリアルタイムな同時編集機能を付加することにより，作業の並列性を高めることを狙った．

キーワード ソフトウェア開発環境，細粒度ロック，ネットワーク，並列性

Parallelization and Synchronization in Coding Activities by Multiple Software Engineers

Naoto KAJIWARA† Yasuhiro SUGIYAMA‡

† Graduate School of Engineering, Nihon University, Koriyama, Fukushima, 963-8642 Japan
‡ Department of Computer Science, College of Engineering, Nihon University, Koriyama, Fukushima, 963-8642 Japan
E-mail: † kajiwara@ssl.ce.nihon-u.ac.jp, ‡ sugiyama@ce.nihon-u.ac.jp

Abstract In software development, it is important to avoid conflicts of software development activities by multiple software engineers. On the other hand, it is also important to ensure the high parallelism of their activities. So far, we have proposed the Fine Grain Locking mechanism to avoid conflicts of their works. However, the degree of parallelism of activities was not high enough. To remedy this problem, we have added a real-time editing capability for multiple software engineers to the locking mechanism.

Keyword Software Development, Network, Multi-Users, Fine Grain Locking Mechanisms

1. 研究の背景

最近は，インターネットの普及に伴い，国境を越えての大規模なソフトウェアプロジェクトも珍しくない．そのため，ネットワークを経由してソースコードを共有することができ，共同で開発作業ができるようなソフトウェア開発環境の構築が求められている．現在，CVS(Concurrent Versions System)[1]のようなネットワークで１つのソースコードを共有し開発を進められるシステムが存在する．オープンソースソフトウェアでは CVS を用いて編集されているものが数多い．しかし，複数名での編集作業には必ず競合の問題が発生する．

競合を回避する方法にはソースファイルをロックするものと，ロックはせずに更新時にマージするものがある．SCCS[2]や RCS[3]は前者のロックによる方式を採用し，CVS では後者の方式を採用している．マージによる競合の解消では，複数名のプログラマによる編集作業の並列性は高い．しかし，プログラムの意味まで考えたマージを自動化することは難しい．また複数名が同じ場所を編集した場合，全ての編集作業を矛盾なくマージすることは難しい．このことは結果的にプロジェクトの開発効率の低下を招いてしまう．

そこで我々の研究室では，マージではなくロックによる競合解決方法を改善し，ソフトウェアの開発効率を向上することを研究している．この方法では，編集したいソースコードをロックすることで編集作業の競合を未然に回避することができる．しかし，これまでの SCCS や RCS による方式では，ファイルを粒度とするために，複数のプログラマによる編集作業の並列性が低くなってしまうという問題がある．そこで，ロックの対象をファイルではなく構文単位にまで細かくすることで，編集作業の並列性向上に繋がるのではないかと考え，これを細粒度ロック[4]と名づけ，開発を行った．我々の研究室では，Java 言語でプログラミングを行うためのソフトウェア開発環境として ClassFactory[5]を開発している．細粒度ロックはその開発環境の機能の一部である．ロックの際にはプログ

ラムの内容を解析し，ロックする部分に関連するソースも併せてロックすることで，競合による矛盾も防止する.これにより,ファイルを粒度とした場合に比べ，大幅な並列性の向上に成功した.

しかしながら，いかに粒度を細かくしてロックをしようとも，ロックをする必要の無いマージによる競合の解消に比べると，並列性の面では劣ることは否定できない.

そこで，新たな観点で，細粒度ロックシステムの並列性を向上することを検討している.細粒度ロックは，ロックを行う対象の粒度を可変とするアプローチであるが，今回検討している方式は，ロックの意味自身を柔軟に考えるというアプローチである．これまでのロックでは，ロックすることは，ロックを行った部分に対する排他的な変更権を得るという意味合いが強かったが，これを見直し，状況に応じたロックが行えるようにするというアプローチである．例えば，ロック済みの部分であっても閲覧のみは許可するというような方式はこれまでのシステムでも実装されている．これをさらに発展させ，より細かくロックの機能を整理していくことが本研究の目的である.

本稿の構成は以下の通りである．まず 2 章で ClassFactory と細粒度ロックの概要とその問題点について述べる．３章でその問題点を解決する方式について述べ，４章でその実装について述べる．最後に評価と今後の課題で論文を締めくくる.

2. ClassFactory と細粒度ロックの概要

ClassFactory には細粒度ロックを含み，大きく分けて以下の３つの機能が実装されている.

- クラスブラウザ機能
- ネットワークを介した共同開発機能
- 細粒度ロック機能

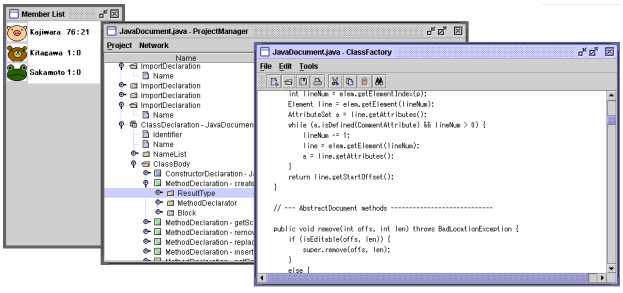

図 1 ClassFactory の概観

2.1. クラスブラウザ

ClassFactory は，ソースファイルをまとめて 1 つのプロジェクトとして管理し，編集中のクラスの内容を解析して一覧で表示して編集作業を許すクラスブラウザの機能を持つ．図 1 は ClassFactory の実行中の様子を示している.Java で記述されたプログラムを解析し，クラスやインタフェースの関連，クラスで定義されているフィールドやメソッドを視覚的に理解することを補助し，プログラムの構造に即した編集作業を支援する．クラスの要素がツリー構造で表示され，そのためクラスのメンバ単位での編集が可能となる.

2.2. ネットワークを介した共同開発機能

ClassFactory では，ネットワークを介して複数のプログラマが共同でソフトウェアの開発作業をするための環境を CVS などの外部のツールに依存しないで実施できる機能を提供する．ClassFactory は，単体でも使用できるが，クライアントやサーバーとしても使用することができる．ひとつの ClassFactory が備えるデータベースをサーバーとして使用し,他の ClassFactory をクライアントとすることにより，クライアントサーバーシステムを構築している.

2.3. 細粒度ロックにおける排他制御

細粒度ロックは，ロックの粒度を細かくすることにより，複数のプログラマによる編集作業の並列性を高め，編集作業の競合を未然に防ぐ方式である.

図 2 では細粒度ロックの概念図を示す．図中の木構造は,ソースプログラムの構文木を表したものである.たとえば図 2 の左側で，A がプログラムの構文要素の一部分を編集するとする．ファイルロックの場合は，ここでファイル全体をロックしたわけであるが，細粒度ロックでは，A が編集する構文のみをロックする.すると図 2 の右側で示すように，B が編集したい構文要素が，A が編集している構文要素ではないためロックされていないので，B も編集作業に加わることが可能になる.

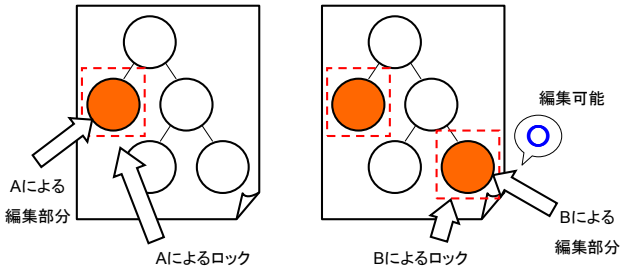

図 2 細粒度ロックシステムの概念図

粒度を細かくすることにより，プログラマが構文要素をロックするという作業が煩わしくなるということが考えられる．しかし，ClassFactory では，この煩雑さを無くすための支援を行っている．例えば図 2 の右側で,B がロックをする際に,A がロックをしている，していないに関わらず，B が編集したい構文を含む Class 全体をロックしたとする．もちろん A がロックしている構文要素を含むために,Class 全体をロックすることは出来ない．しかし ClassFactory が，自動的に

A がロックしている構文要素を除く，最も広い範囲でロックできるロックの粒度と組み合わせで構文要素を選択して提示してくれる．このように ClassFactory における細粒度ロックは，システムを煩雑にするのではなく，プログラマに対して，ロックの選択肢の幅を広げることが目的である．

2.4. 細粒度ロックの問題点

細粒度ロックは，ロックの粒度を細かくして並列性を向上させたが，以下のような未解決の問題が存在する．

まず第 1 は，Check-out 操作の問題点である．これまでのバージョン管理システムは，ベースライニング機能により，編集を行う際には，Check-out 操作により編集対象のファイルのコピーを作成し，それを編集した後，もともとのファイルを更新することを基本としていた．細粒度ロックもこの原則に従っているが，Check-out 操作により，各プログラマは自分専用のソースプログラムのコピーを獲得することになる．このコピーは一般的には他のプログラマには非公開である．他のプログラマが Check-out した構文要素を，他のプログラマが編集目的で Check-out できないようにするためにロックを行う．従って，あるプログラマが Check-out している部分の現状を，他のプログラマが把握できない．

これを解決する方法のひとつとして，Check-out 即ちロックが行われている部分であっても，閲覧だけは許可する方式が RCS などでも採用されている．しかし，読み出し専用のロックは，本質的な問題の解決にはなっていない．

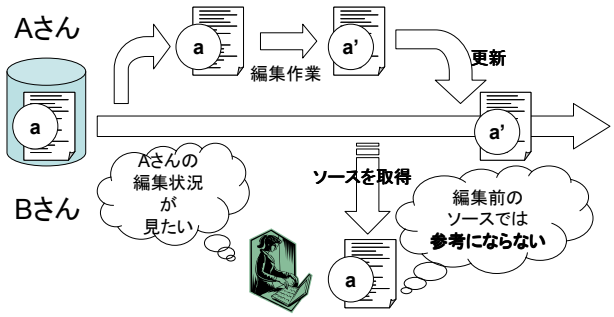

図 3 参考可能なのはロック前のソースコード

例えば図 3 で示すように，A がある構文要素をロックし編集をしている最中に，B が他のある部分を編集する際，A がロックしている部分のソースを参考にする必要があり，読み出し専用でチェックアウトしたとする．A はソース a を編集し変更されたソース a'をチェックイン（すなわちもともとのファイルを更新）する．しかし，A がソース a'をサーバーに更新するのは，編集が終わってからなので，B が参考にできるソースは A がロックをした時点の，古いソース a になってしまう．その結果，B がある部分の編集を終了し，もとのファイルを更新する頃には，B が参考にしたソースは A によりすでに上書きされて別のものになってしまい，プログラムの意味的矛盾が発生してしまう．これは，マージの問題と基本的に同じである．

第 2 の問題点は，どんなにロックの粒度を細かくしたとしても，複数のプログラマが同時期に他のプログラマのロック部分に対して，編集する必要が出てくる可能性は 0 ではなく，ロックシステムを利用する以上，100%の並列性は確保できないということになる．もちろん文字単位までロックの粒度を上げれば，少なくとも構文要素が粒度である場合よりは並列性が増すであろう．しかし，Java 言語のプログラミングで基本となるものは構文要素であり，ロックをする側としても，文字単位でのロックは非常に困難なものである．よってロックの粒度は，構文単位で留めておくのが無難と考えられる．従って，ある程度の大きさの粒度を保っていながら，並列性を向上させる別の方式を実現する必要がある．

3. 並列性向上に向けた機能の概要

前章で述べた細粒度ロックにおける二つの問題点を解決するための我々の方針は，まず，ファイルをリアルタイムに共有することによって，Check-in/out の操作を行わなくても細粒度ロック作業が行えるようにすることである．

また，ロックについての認識を改め，ロックをその構文要素を独占的に編集する権利と理解せずに，その構文要素への他のプログラマからのアクセスを管理する権利を得るものと理解する．言い換えれば，ロック済みの構文要素に関しては，ロックを行ったプログラマの裁量によって，他のプログラマに編集や閲覧あるいは部分的な入れ子のロックを許すなどの細かなロックの制御を行えるようにすることである．

3.1. リアルタイムなファイル共有機能

これまでのベースライニングの原則を見直し，基本的に，ファイルはすべてのプログラマによって共有されていることを原則とする．ClassFactory を使用することにより，複数のプログラマは，同一のソースファイルを共有し同時に見ることができる．

ただし，共有されたファイルの編集作業を無条件に許すと，編集作業が競合することは容易に想像がつく．そこで編集作業は，編集対象の構文要素をロックした後のみに可能とする．従って，ロックを行ったプログラマのみが編集できるが，編集中のファイルの状況は，全てのプログラマにより共有されることとなる．

また，もちろんロックを行ったプログラマは，編集内容に満足しなかったなどの場合，共有するソースファイルに変更が加わることなく，編集を破棄できる．

3.2. ロックの制御

構文要素に対するロックを獲得すると，排他的な編集を行う権利を得るだけではなく，他のプログラマが同一の構文要素あるいはその一部に対してアクセスをしたい場合などには，そのアクセスを許可したりする権利を得ることとなる．

例えば，編集作業の性質上，編集中のファイルの状況は他のプログラマには公開しない方がよい場合も多い．他のプログラマへの公開が望ましくないと判断したプログラマは，自分のロックした構文要素に対し，他のプログラマへの公開の程度などを指定することができる．

公開の可否の指定を含め，最初にロックを獲得したプログラマは，ロック部分に対し，他のプログラマが可能な作業に以下のような様々な制限を付加できる．

(1) ロックの共有

プログラマ A がロック済みの構文要素に対して，プログラマ B がロックを要求し，その要求を A が認めると，「A によるロック」が「A と B によるロック」になる．A と B は全く同時に編集作業を行えるようになる．ロックを共有させない方式では，A がロックを解放するまで待たなければならなかった B が，待ち時間無しで編集に参加できるようになる．

(2) ロックの貸し出し

共有のみではなく，一時的なロックの貸し出しも有効である．A がロック中の構文要素を B が編集したい場合などは，A のロックを解除することなく B にロックの権利を一時的に貸し出すことにより，B に編集作業を許す．貸し出している間は，A は編集作業を行えなくなるが，B の作業が終わった段階で，ロックはそのまま A に戻される．これにより，A は作業を継続できる．

(3) ロックの細分化

ロックの共有や貸し出しは，既にロックを行っている構文要素全体が対象となるとは限らない．ロック済みの構文要所の一部分を他のプログラマが編集したい場合も多い．その場合には，一部に限ってロックの共有を許可したり，貸し出すこともできる．たとえば，プログラマ A がクラス全体を編集する予定でクラス全体をロックしていたと仮定する．メソッドがいくつかあり，そのうちのひとつのメソッドを編集中に別のプログラマ B が別の，メソッドを編集する必要が発生した場合などは，A は B にそのメソッドのみのロックの共有を許可したり，そのメソッドのロック権を貸し出すことにより排他的にロックすることを許可できる．B がロックを解除すると，A のみがロックを保持している状態に戻り，A はそのまま作業を継続できる．

3.3. ロックの状況管理機能

ロックされている部分にアクセスしたいプログラマにとっては，既にロックされている部分に関するロックの状況を把握できる手段が提供されていることが重要である．例えば，誰が既にロックしているのかとか，いつからロックしているのか等の情報は最低限必要であろう．また，ロックを保持しているプログラマに対して，何か問い合わせをしたりする機能も必要と考える．

また，これに関連した機能として，チャット機能やヒストリ機能も重要であると考えている．チャット機能は，ロックを共有しているプログラマ同士でのコミュニケーションを行う際に使用される．ロックの共有中はチャットウィンドウが表示され，そこに他のプログラマのメッセージが表示される．

ヒストリ機能を用いると，誰がどの構文要素をいつからいつまでロックしたというようなログが残されるだけでなく，ロックをしたプログラマは，ロック部分に対して行った自分の作業内容をメッセージログとして残すことができる．このヒストリは他のプログラマが自由に見ることが可能である．

4. 機能の実装方式

本研究で ClassFactory に対し実装した機能の実装方式について説明する．図 3 で示す ClassFactory[2]は Java 言語で書かれており，そのため，本機能も Java で開発を行った．それに伴い，プログラマ同士でソースコードや編集作業内容を通信する部分の実装には，RMI(Remote Method Invocation)技術を使用している．RMI とは，ある JVM（Java Virtual Machine）から別の JVM にあるオブジェクトのメソッドを呼ぶための仕組みである．

本研究では，3 章で述べた細粒度ロックにおける問題の解決策を，以下の二つの機能として細粒度ロックシステムに対し実装した．

- ロック部分のリアルタイムな閲覧機能
- ロックの共有とリアルタイムな同時編集機能

ただし，ロックの貸し出しや，ロックの細分化についてはまだ実装段階に至ってはいない．

これらの機能は，一つのソースファイルを複数のプログラマが共有できる環境があって実現できる．そこでまず 4.1 では，その環境で必要なデータベースサーバーの仕組みについて説明する．4.2 では，共有したソースファイルの一部を，あるプログラマがロックした後，他のプログラマが閲覧したり，ロックを共有したりする際の処理について説明する．そして 4.3 では，閲覧時や，ロックの共有時に行われる，編集作業をリアルタイムに他のプログラマに対し送信する機能について，例を用いて説明する．

4.1. C/S におけるデータベースの役割

細粒度ロックや，今回開発した機能を利用するためには，第一段階として，一つのソースファイルを複数のプログラマがネットワーク上で共有できる環境を構築しなければならない．そのため，ClassFactory ではクライアントサーバーシステムを採用し，サーバーには共有するソースファイルを置き，クライアントにはそのソースファイルをダウンロードし編集するためのエディタ機能を設けた．では，このようにクライアント(プログラマ)が一つのソースコードを共有するまでの流れを，図 4 を用いて説明する．

まず共有したいソースファイルを所持したクライアントが，ClassFactory の機能を用いて，そのソースファイルを ClassFactory のローカルデータベースに読み込む．そしてクライアントがサーバーとなる ClassFactory を指定すると，そのサーバーに対しローカルで読み込んだソースファイルをアップロードする．

アップロードしたソースファイルはサーバー上のデータベースに読み込まれる．そのソースを共有したいクライアントが，ClassFactory からサーバーにアクセスすると，サーバのデータベース上のソースをローカルデータベースに読み込む．クライアントで読み込まれたソースファイルは，構文解析され GUI により，構文要素単位でツリー構造化され表示される．

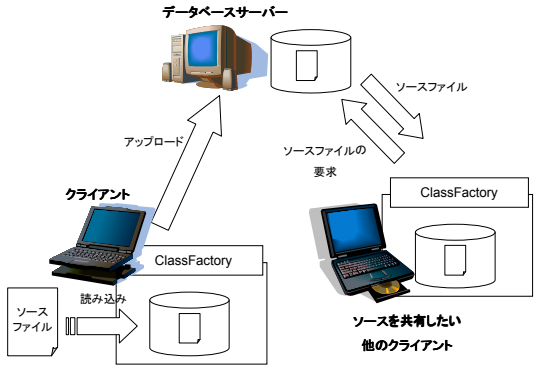

図 4 データベースサーバーとソースファイルの共有

4.2. 閲覧とロックの共有の開始時における処理

ソースファイルが複数のプログラマによって共有されたら，次に，各プログラマは編集作業に移るため，編集部分をロックすることになる．

ロックをする際に，各プログラマはロック部分に対し，以下のような設定をすることを可能とした．

1. 閲覧のみ可（ロックの共有は不可）
2. 閲覧，ロックの共有，両方とも可
3. 閲覧，ロックの共有，両方とも不可

また，ロックの共有を可とした場合でも，無条件で共有を許可せずに，特定のプログラマ以外に対しては共有させないなどや，共有を要求した際確認メッセージを表示させ，了承を得なければ共有を許可させないなどの制限を，ロックをする際に付加できる．これらの設定は，データベースサーバーにロック部分ごとに保持される．

では，プログラマが編集部分をロックし，その後他のプログラマが，そのロック部分に対し，閲覧やロックの共有を行う際の処理の流れについて説明をする．

まず，あるプログラマが編集部分をロックすると，サーバーのデータベース上に，その部分をロックしたということが書き込まれる．その際，もし編集するプログラマがロック部分を閲覧可能に設定したならば，他のプログラマはその部分をリアルタイムで閲覧可能となる．また，編集するプログラマがロックの共有を許可すれば，他のプログラマが編集作業に参加できるようになる．

では，ロックの共有を行った場合のクライアントサーバー間の処理について図 5 を用いて説明する．ここではクライアント A のロック部分を，B がロックの共有をする場合を例に述べる．

まず，A がソースコードの編集する部分に対しロックをかける．その際 A はロックの共有機能を有効にしたとする．サーバーは A からロックの要求を受け取るとともに，A の IP アドレスを取得する．要求を受けたサーバーは，ロックする部分がロックされていないことが確認できたら，その部分をロックし，ロックできたことを A に伝え，A は編集作業に移る．

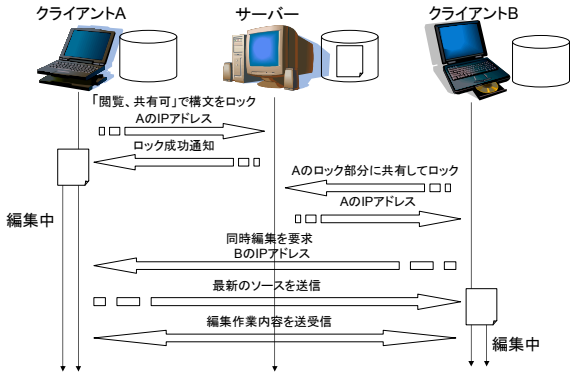

図 5 ロック部分の共有機能

そして B は，A がロックしている部分が共有可能であることを確認すると，サーバーに対し，その部分のロックの共有を要求する．要求を受けたサーバーは，共有の確認を A に取る必要がある場合はその確認をとった後，ロックを共有させると同時に，B に対し A の IP アドレスを返す．B はその IP アドレスを元に A にアクセスし，その際 B の IP アドレスも通知される．ロックの共有の要求を受けた A は，編集中のソースコードを B に返す．そして，その後，お互いが手元にあるソースコードに編集を加えるごとに，自動的に共有している相手のリモートメソッドを呼び出し，編集結果がお互いのソースコードに反映されることになる．

閲覧時の場合は，A が B に対して最新のソースを送信するまでは，上述したロックの共有時の処理と同様にして行われる．ただし，B からデータを送信する必要は無いため，リモートメソッドの呼び出しは A から B へ一方的に行われる．

また，これらの例はクライアントが 2 名の場合だが，3 名以上になった場合は，最初にロックをしたクライアントが，編集作業に参加するその他のクライアントの IP アドレスを全て所持する．そして，最初にロックをしたクライアントが，ソースコードに編集を加える，または他のクライアントから編集作業結果を受け取ると，その編集作業結果をリアルタイムで全てのクライアントに送信する．

逆に，後からロックを共有したクライアントは，最初にロックをしたクライアントの IP アドレスのみを所持していて，編集作業を送信するのは，その IP アドレスのみに対して行われ，全てのクライアントに対し編集作業結果を送信することはない．

4.3. 編集作業内容のリアルタイム同期

ここでは，ロックの共有をした後，複数のクライアント間で編集作業結果を，それぞれのクライアントのソースコードに反映させる際のメソッド呼び出し時の処理について，詳しく説明する．

図 6 で示すように，この場合では，最初にロックをしたクライアントを「親」とし，その後ロックをしてきたクライアントをそれぞれ「子 A」「子 B」「子 C」としている．

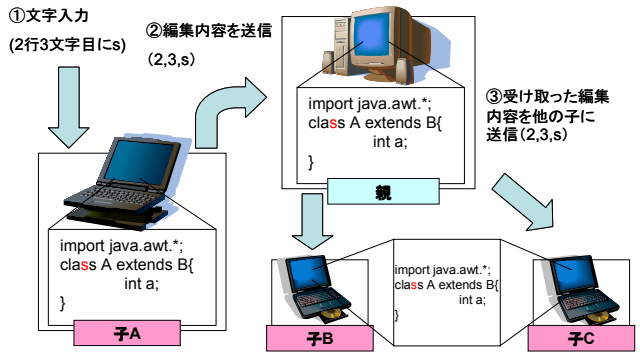

図 6 複数名プログラマ間での編集作業結果の同期

例えば，まず子 A がソースコードの間違いに気づき，この場合では 2 行目の 3 文字目に s という文字を入力したとしよう．すると，編集した場所（2,3）と文字(s)が親に対して自動的に送信される．親は子 A から編集作業結果を受け取ると，すぐに子 B，子 C に対し，受け取ったデータをマルチキャストする．そして，子 A からの編集作業結果を受け取った親と子 B，子 C は，s という文字を 2 行目の 3 列目に挿入する．このような処理は，親，子とも，キーボード入力やコピー＆ペーストなどの全ての種類の編集作業に対し行われている．

5. 評価と今後の課題

本論文では細粒度ロックにおける作業並列性の低さを指摘し，その解決策としてロックを複数のプログラマで共有させ，ロックを共有したプログラマ間で編集作業の内容をリアルタイムで同期させる方式を提案し実装した結果を述べた．その結果，ロックの粒度を適度な大きさに留めた上で，複数のプログラマによる作業の競合を抑制しつつ，並列性を向上させることができることが確認できた．

リアルタイムで複数のプログラマ間で編集作業を同期させる機能は，MacOS 上で動作する SubEthaEdit[6]という現存するエディタにも実装されている．そこで作業並列性やエディタの機能性という面において，ClassFactory との比較検証が必要であると考える．

また，編集作業結果をリアルタイムですぐさま他のプログラマに送信する際，ネットワーク上のタイムラグなどによって，入力された文字の順序が逆になってネットワーク越しのソースコードに反映されたりするなどの不具合が見つかっており，タイムラグを考慮した設計が必要である．

なお，本論文中で述べたロックの貸し出し機能や，ロックの細分化機能は実装が済んだではない．また，他にも，いくつかの機能が未実装であったり，安定した動作をしていなかったりするため，今後，引き続き実装を続けていかなければならない．

文　献

[1] CVS, https://www.cvshome.org/

[2] M. J. Rochkind, The source code control system. *IEEE Transactions on Software Engineering*, Vol. 1, No. 4, pp. 364-370, 1975.

[3] Walter F. Tichy, RCS – A System for Version Control. *Software – Practice and Experience*, Vol. 15, No. 7, pp. 637-654, 1985.

[4] 佐藤友章，杉山安洋，ソフトウェアの共同開発作業における細粒度ロックの有効性の検証，日本ソフトウェア科学会第 19 回大会（2002 年度）論文集

[5] 佐藤友章，杉山安洋，ClassFactory を用いたグループ開発作業における排他制御方式の検討，FOSE2001

[6] SubEthaEdit, http://www.codingmonkeys.de/subethaedit/