平成20年度 修士論文

既存のプログラミングツールを活用する共同開発環境の設計と実装

電気通信大学 大学院情報システム学研究科

情報システム設計学専攻

0650038　山下 剛

指導教員　多田 好克　教授
田野 俊一　教授
星 守　　教授

提出日　平成21年1月29日

目 次

第 1 章　序論 **1**

1.1 背景 1

1.2 目的 2

第 2 章　共同開発と Awareness **4**

2.1 共同開発 4

2.2 共同開発の問題点 6

2.3 Awareness と Awareness 情報 6

2.4 共同開発支援システム 7

2.5 既存システムを少人数の共同開発に導入する際の問題点 12

2.6 エディタ、バージョン管理システムを自由に選べる少人数の共同開発でも重要な Awareness 情報 13

2.7 少人数の共同開発で導入が容易な共同開発支援システム 15

第 3 章　既存のエディタ、バージョン管理システムからの Awareness 情報収集 **16**

3.1 各開発者のファイルの編集状況について 17

3.2 既存のエディタからの情報収集 18

3.2.1 ファイルシステムからの情報収集 18

3.2.2 エディタの機能を利用した情報収集 19

3.2.3 本研究で使用する既存のエディタからの情報収集手法 20

3.3 既存のバージョン管理システムからの情報収集 23

3.3.1 サーバからの情報収集 24

3.3.2 クライアントからの情報収集 26

3.3.3 本研究で利用する既存のバージョン管理システムからの情報収集手法 26
3.4 エディタ連携プログラム、バージョン管理システム連携プログラムについて 27
3.4.1 エディタ連携プログラム、バージョン管理システム連携プログラムでの通信 27

第 4 章 CAEDE:プログラミング環境と連携する協調開発支援ミドルウェア 29

4.1 共同開発支援システム:CAEDE 29
4.2 CAEDE が提供する Awareness 情報 30
4.3 CAEDE を利用した共同開発の方法 32
4.4 CAEDE の構造 33
4.5 CAEDE クライアント 35
4.5.1 メッセージディスパッチャ、メッセージトランスミッタ . . 38
4.5.2 ユーザ管理モジュール、タスク管理モジュール 38
4.5.3 メッセージングモジュール 39
4.5.4 通知モジュール 39
4.5.5 Awareness 情報表示部 40
4.6 CAEDE サーバ 42
4.7 CAEDE プロトコル 45
4.7.1 CAEDE プロトコルのメッセージ 46
4.7.2 CAEDE のメッセージの種類 48
4.7.3 CAEDE のメッセージの詳細 49
4.8 CAEDE とエディタの連携 53
4.8.1 CAEDE で利用できるエディタ 53
4.8.2 Emacs と連携するコマンド:transcaede 55
4.9 CAEDE とバージョン管理システムの連携 56

第 5 章　システムの評価 **58**

5.1　ネットワーク性能の評価 58

5.2　実験結果 60

5.2.1　実験環境 60

5.2.2　実験手順 61

5.2.3　実験結果 62

5.3　考察 62

第 6 章　関連研究 **65**

6.1　Awareness 65

6.2　共同開発支援システム 66

第 7 章　結論 **67**

7.1　まとめ 67

7.2　今後の課題 68

図 目 次

2.1 バージョン管理システムを利用した共同開発 5
2.2 Awareness 情報が無い共同開発 8
2.3 Awareness 情報がある共同開発 9
2.4 Awareness 情報を提供する共同開発支援システム 10

3.1 エディタからの情報収集 21
3.2 異なる OS での外部プログラムの流用 22
3.3 バージョン管理システムからの情報収集 25

4.1 CAEDE を利用した共同開発 32
4.2 CAEDE の概観 34
4.3 CAEDE クライアントの概観 37
4.4 CAEDE の通知 39
4.5 CAEDE の Awareness 情報表示部 41
4.6 他のユーザの編集状況の確認 42
4.7 CAEDE の IM 機能 43
4.8 CAEDE サーバ内部のメッセージの流れ 44
4.9 CAEDE プロトコルのメッセージ 46

5.1 実験で送信したメッセージの例 62
5.2 ネットワークの遅延のテスト結果 64

表 目 次

2.1 共同開発に必要な Awareness 情報と依存先 14

4.1 開発者の状態の一覧表 31

4.2 在席状態と色の対応表 41

ソースコード目次

4.1 Emacs に行なった設定 54
5.1 ネットワークに頻繁にデータ送信を行なうプログラム 62

第 1 章

序論

1.1 背景

ネットワークを通じて複数の開発者が共同で1つのソフトウェアを開発する共同開発が一般的となった。共同開発の問題として、作業を行なう者同士が空間的に近くに居ないことによるコミュニケーションのとりづらさ、作業の進捗の把握しづらさがある。コミュニケーションや互いの作業の進捗把握がうまく行なわれていない場合、開発者同士が意識的に連絡をとる必要があり開発者への負担が増加し、結果として作業の進行に影響を及ぼす。このため、開発者同士の作業に関する情報の共有が共同開発を円滑に進めるには重要となる。

上記の共同開発のような複数人による作業を、計算機を使って支援を行なうCSCW(Computer-supported cooperative work) という研究分野がある。CSCW の分野では、共同作業を行なっている他のメンバの作業の状況把握のことを Awareness と呼ぶ。Awareness を促すことで、メンバは協調作業において個人の作業のグループ全体における役割、目標や進捗に対する個人の作業のグループ全体における意義の理解が可能になる。つまり、Awareness によって進捗の管理が容易となる。

Awareness に利用できる情報のことを Awareness 情報と呼ぶ。Awareness 情報は、他の人の作業の状況を示す情報である。具体的には、あるファイルを編集しているのが誰であるか、どういう編集を行なっているか、他の人の在席情報といったものが挙げられる。例えば、ファイルの編集状況を共有することで、誰かが編集中

のファイルを開くことを避けることが可能となり、競合の発生を回避することができる。また他人の作業がどの程度進行しているのかという事も把握可能となる。

共同開発の研究分野では、共同開発の支援を行なうためのシステムも提案されている。共同開発支援システムは開発者への Awareness 情報の提供を目指している。これらのシステムでは、Awareness 情報の提供のために、ファイルを誰が編集したのか、マウスの位置やエディタのカーソルの位置といった情報をエディタやバージョン管理システムから収集している。しかし、これらの情報は一般のエディタ、バージョン管理システムでは取得方法がない。そのため、これまでのシステムでは、情報取得が可能な独自のエディタ、バージョン管理システムを設計しそれを利用している場合や情報取得が容易な特定のエディタ、バージョン管理システムを指定しているものが多い。

そのような共同開発支援システムを導入した場合、開発者はシステム導入のメリットとして Awareness 情報を得ることができる。しかし、デメリットとして、開発者は指定されたエディタ、指定されたバージョン管理システムという、制限された環境下での開発を余儀なくされる。

特に少人数の共同開発では、個人の能力を最大限に生かした方が効率よく開発できる。つまり、今まで利用してきた開発者の好みのエディタやバージョン管理システムを利用して開発を行なう方が開発の効率が良い。このため、既存のエディタやバージョン管理システムが利用できないこれまでの共同開発支援システムは、少人数の共同開発では導入するメリットが薄い。

1.2 目的

既存のエディタ、既存のバージョン管理システムから Awareness 情報を収集し、Awareness 情報を提供する共同開発支援システムを構築する。このような構成であればシステム利用者は既存のエディタ、バージョン管理システムを利用したまま共

同開発支援システムの導入が可能となり Awareness 情報を利用できる。

本研究では、既存のエディタ、バージョン管理システムから収集できる可能性があり少人数の共同開発にも利用できる Awareness 情報を定義した。既存のエディタ、バージョン管理システムを利用して Awareness 情報を取得し、Awareness 情報の配信を行なう共同開発支援システム CAEDE の設計および実装を行なった。CAEDE は、既存のエディタ Emacs と、既存のバージョン管理システムである CVS を利用して動作する共同開発支援システムである。

以下の章では、第 2 章で少人数の共同開発に必要な Awareness 情報、第 3 章では少人数の共同開発に必要な Awareness 情報を既存のエディタと既存のバージョン管理システムから収集する方法、第 4 章では、既存のエディタ、既存のバージョン管理システムを利用して動作する少人数の向けの共同開発支援システム CAEDE の実装、第 5 章では CAEDE のネットワーク性能の評価、第 6 章で関連研究、第 7 章で本研究の結論を述べる。

第 2 章

共同開発と Awareness

2.1 共同開発

ネットワークを通じて複数人で 1 つのソフトウェアを開発する共同開発というものがある。

本研究では、この共同開発の中でも少人数で行なうものに注目する。例えば、大学の研究室、大学のサークル活動といった場所で行なわれる数人から十数名程度の少人数でのソフトウェア開発である。

こういった共同開発では、開発者が各々好き勝手にファイルの管理を行なっていくと一貫性が取れなくなる。そのために、図 2.1 で示すようなバージョン管理システムというファイルの一貫性を管理するシステムが利用されている。共同開発では、開発者はこのバージョン管理システムが保持している最新のファイルの保管庫であるリポジトリからダウンロード (チェックアウト) する。そして、そのファイルにエディタで変更を加え、バージョン管理システムのリポジトリに変更を反映 (コミット) することで開発を進めていく。このように進めることで全員が最新のファイルを保ちつつ開発を進めていくことができる。

バージョン管理システムを利用した共同開発のスタイルでは、二人以上が同時に同じファイルの変更をコミットすることも可能になっている。しかし、二人以上で同じ箇所に変更を行なった場合、競合という問題が発生する。競合が発生してしまうと、バージョン管理システムはどのファイルを最新のファイルとして採用す

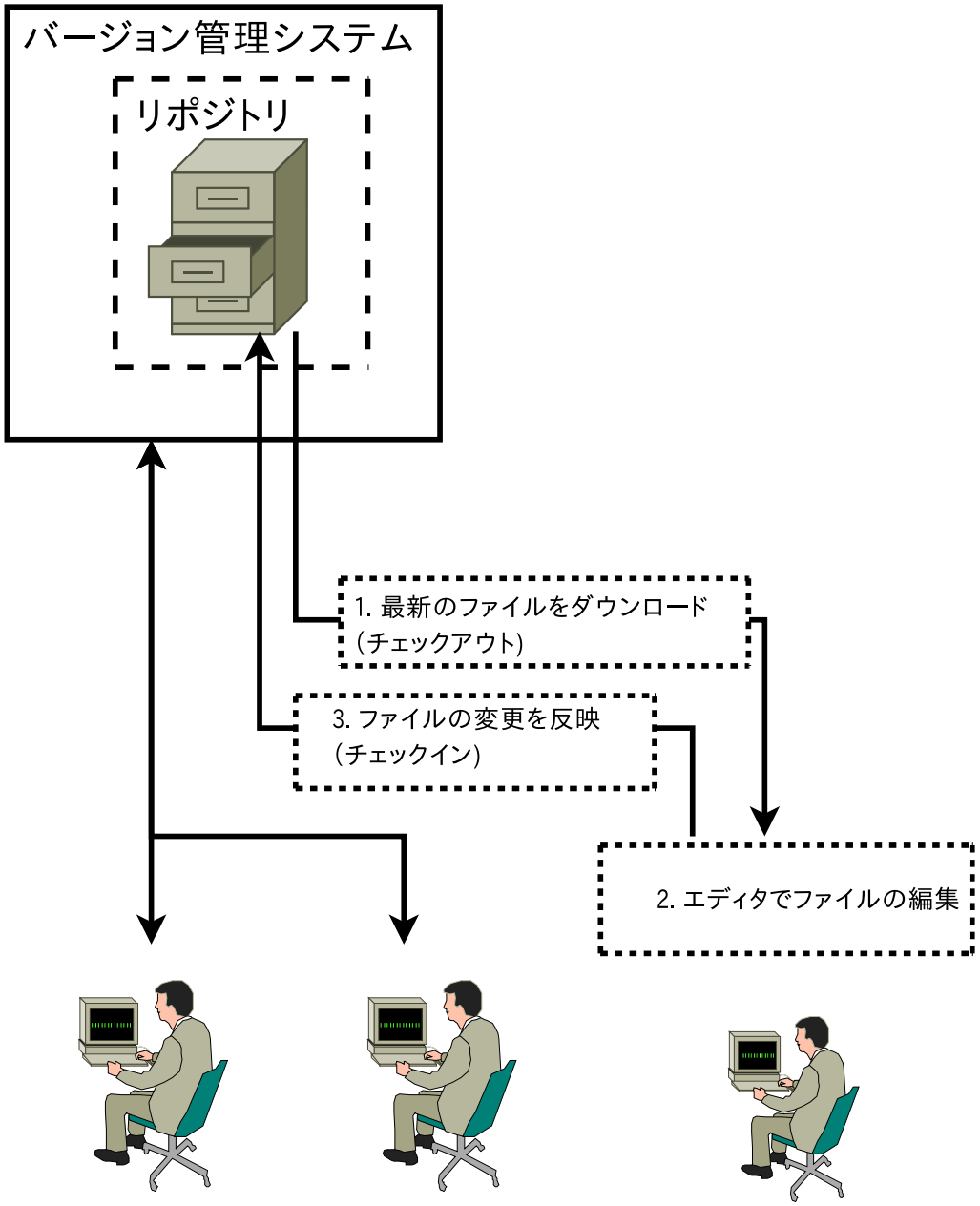

図 2.1: バージョン管理システムを利用した共同開発

るかを決めることが困難となる。多くのバージョン管理システムでは開発者にファイルの修正を促すことで解決を図っている。このため、競合が発生すると開発者同士が連絡をとりあって解決しなければならない。

2.2 共同開発の問題点

複数人でのソフトウェア開発では互いの設計方針や作業の予定といった互いの作業状況の把握が重要である。互いの設計方針を理解せず作成したプログラムの動作が予定と異なってしまったり、作業の予定が分からないため同じような内容のファイルを作成してしまったり、同じファイルを編集してしまい競合が発生してしまうためである。

同じ部屋で作業を行なっている場合には、互いの作業状況の把握は比較的容易に行なえる。例えば、計算機に向かっている場合にはその開発者は何か作業を行なっているということが分かる。作業状況の確認をする場合には、作業状況を確認したい開発者が休憩をしている時に確認するといった具合である。

しかし、共同開発では、同じ部屋に開発者同士が居ない場面が多い。同じ部屋に開発者同士が居ない場合には、互いの作業の進捗や作業の進め方を把握するために互いが意識的に情報を交換しあう必要がある。しかし同じ部屋に居ないため連絡を取りたい相手が現在どういう状態であるか不明である、このためタイミングによっては、連絡を取りたい相手が作業に没頭している場合など連絡に応じられない場合も存在する。そのため、共同開発では連絡が円滑に行なえるかどうかも不明となる。

2.3 Awareness と Awareness 情報

共同開発において共同作業者の作業の状況の把握のことを Awareness と呼ぶ。Awareness している状態を保つことが共同開発では重要である。この Awareness して

いる状態にするために、利用できる情報のことをAwareness情報と呼ぶ。Awareness情報は、他の開発者の状況を表す情報や他の開発者の在席の状態を表す情報などである。

このAwareness情報を各開発者に提供することで、共同開発における状況把握を容易に行なうことができる。Awareness情報を確認して作業状況の把握を行なうことで競合の回避が可能となり、互いの作業状況の確認のための連絡を減らすことができる。また、直接連絡を取る必要がある場合でも、在席状況を確認して、開発者が忙しくない時に連絡を取るということが可能となる。

図2.2のようにAwareness情報が無い場合では互いの作業状況が分からず競合が発生していることはバージョン管理システムにコミットして初めて把握できる。

図2.3のようにAwareness情報があれば、各開発者がどのファイルを編集しているか把握できるため、競合を事前に回避することが可能となる。

2.4 共同開発支援システム

Awareness情報は開発者が自分で配信したりするようなものではうまく機能しない。開発者が更新を忘れてしまい、実際の状況を表さない情報になってしまうと情報の意味がなくなってしまうためである。

このため、Awareness情報を開発者の作業環境から自動的に収集して配信する必要がある。Awareness情報を自動的に配信することで共同開発の支援を行なう共同開発支援システムが提案されている。共同開発支援システムを導入し、図2.4のように専用のエディタやバージョン管理システムを利用することで、開発者は自動的にAwareness情報を受けとりながら開発を行なうことが可能となる。つまり、Awarenessしている状態を常に保つような開発を行なうことが可能となる。

共同開発支援システムでは、提供する機能によって分類できる4種類のAwareness情報を提供している。

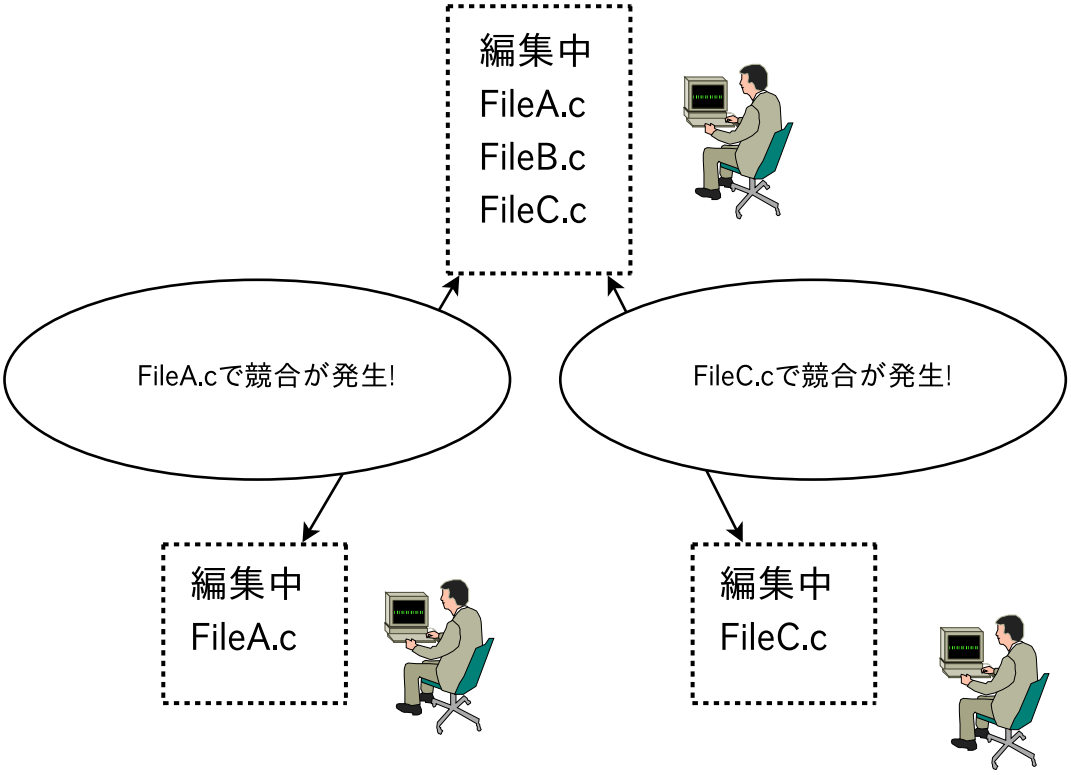

図 2.2: Awareness 情報が無い共同開発

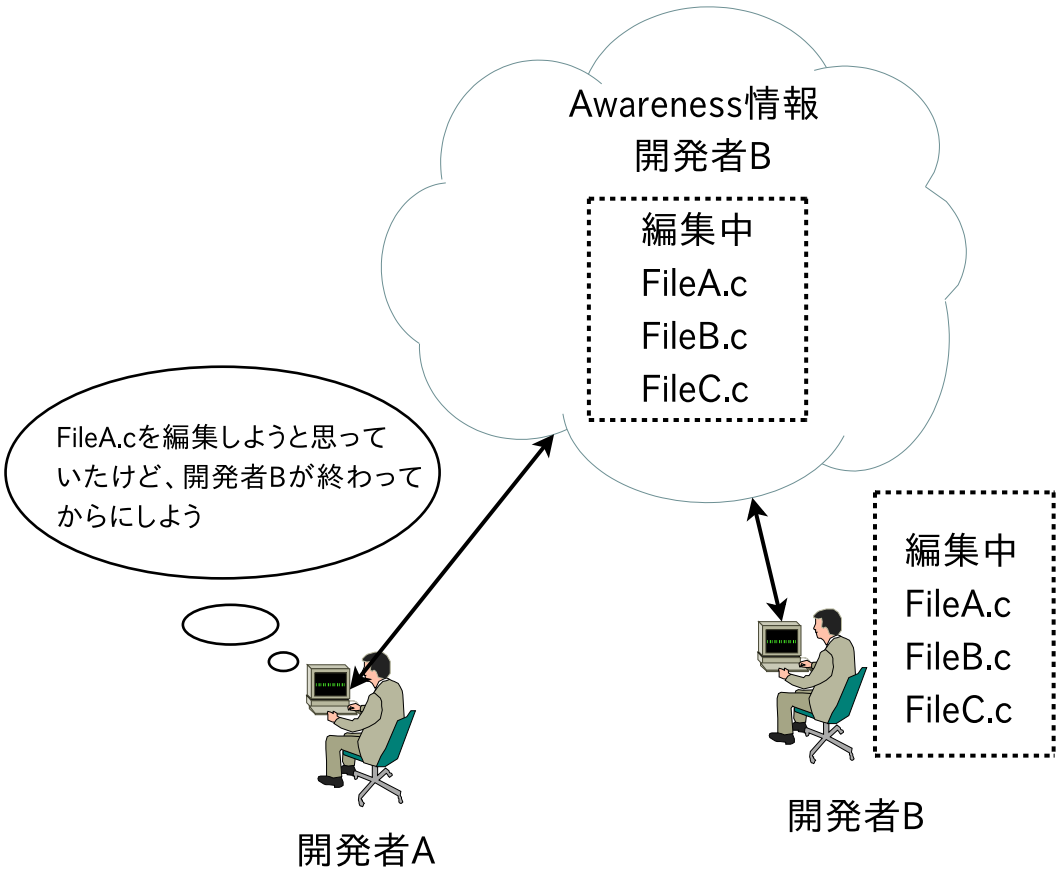

図 2.3: Awareness 情報がある共同開発

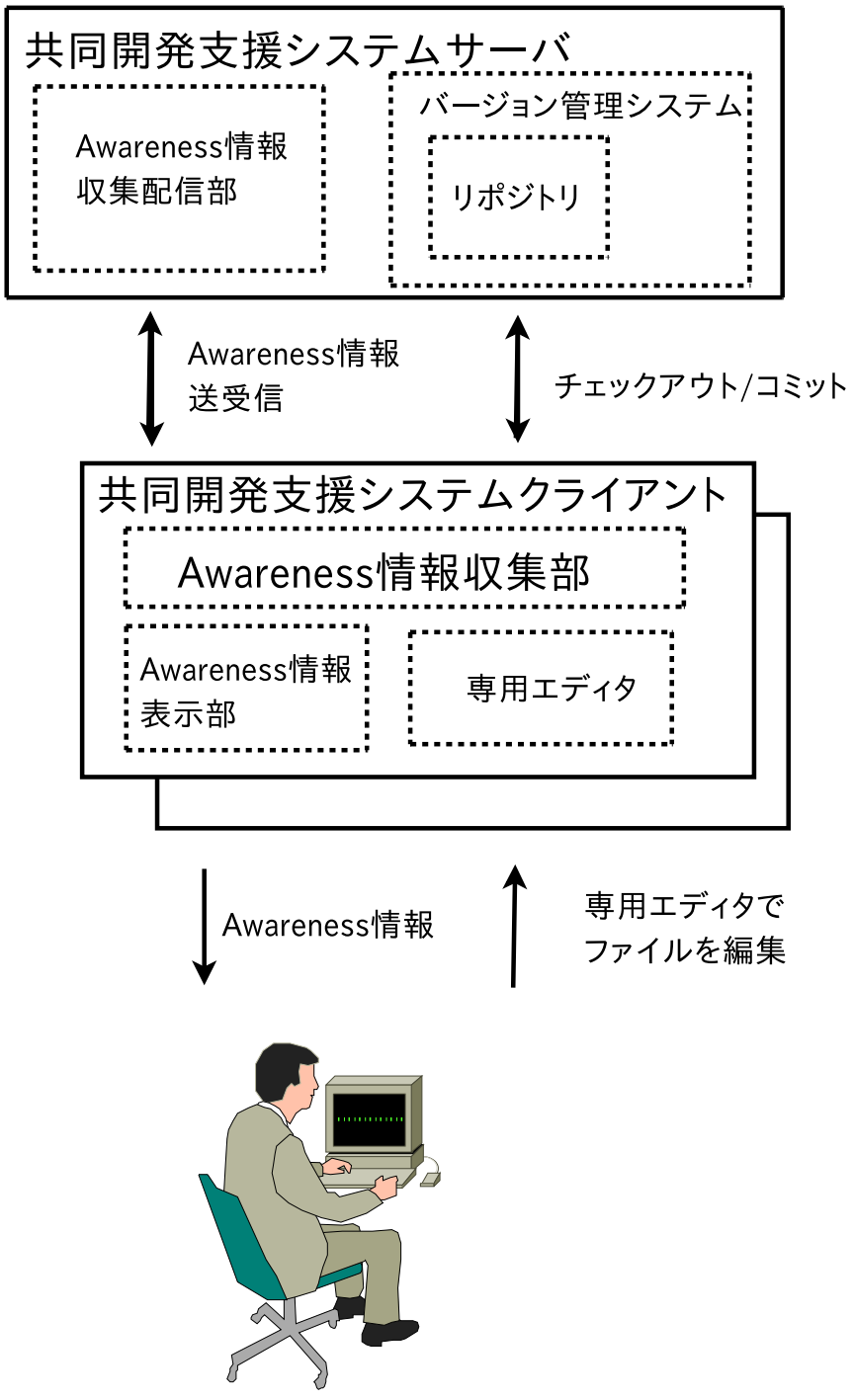

図 2.4: Awareness 情報を提供する共同開発支援システム

- 各開発者の在席情報
- 各開発者のファイルの編集状況
- 各開発者の編集画面
- ファイルの過去の情報

各開発者の在席情報は、各開発者が忙しい、離席しているのか、在席しているのかといった情報を把握するための情報である。この情報を利用することで各開発者は連絡を行なうことができるのか、作業中なのか、離席しているのかという各開発者の作業状態の情報を把握することができる。

各開発者のファイルの編集状況の情報は、各開発者がどのファイルを現在編集しているのか、編集の内容はどういうものなのか、どのファイルがリポジトリで変更されたかといったファイルの編集に関する情報である。ファイルの編集に関する情報を把握することで、競合の回避や設計の方針の情報を把握できる。

編集画面の情報というものは、エディタ上のマウスの位置、エディタのカーソルの位置といった実際の編集状況の情報を提供する。この情報を利用することで、各開発者のファイルの編集状況では不明だった、各開発者の作業の状況をさらに細かく把握することが可能である。例えば、カーソルの位置を把握することで、開発者が注目している箇所の把握が可能となる。編集画面の情報を把握することで、他の開発者の編集の情報を正確に把握することができる。

ファイルの過去の情報は、バージョン管理システムのログの解析を行ない、リポジトリの更新情報に基づく Awareness 情報の提供を行なっている。この情報を利用することで、ファイルにどのような変更が行なわれたのか、最後に編集したのは誰なのかという情報を把握することができ、ファイルの変更の状況を把握することができる。もし編集に疑問がある場合にも最後に編集した開発者が分かれば連絡を取り質問を行なうことができる。

共同開発支援システムはこれらの Awareness 情報を提供し、開発者はこれらの Awareness 情報を利用することで互いの状況把握を容易にしている。このため連絡を取る回数を減らし、共同開発を円滑に進めることを可能にしている。

2.5 既存システムを少人数の共同開発に導入する際の問題点

既存の共同開発支援システムは Awareness 情報を提供することで共同開発を円滑に進めることを可能にしている。しかし、既存の共同開発支援システムには問題もある。Awarenss 情報を収集するためにバージョン管理システム、エディタの内部の情報を解析している。このため開発者が共同開発支援システムを利用する場合は、共同開発支援システムが指定したエディタ、バージョン管理システムを利用することになる。

このため、開発者は普段使用しているエディタ、バージョン管理システムを利用することができないという問題が発生する。特に少人数で行なう共同開発では、個人の能力が全体の能力に大きな影響を及ぼす。このため、個人の能力を最大限に発揮できる開発者の好みのエディタを利用できず、共同開発支援システム導入前から利用しているバージョン管理システムが利用できないということは大きなデメリットとなる。このため、開発者が普段利用しているエディタ、バージョン管理システムを利用することができる共同開発支援システムが小人数の共同開発支援には必要である。

2.6 エディタ、バージョン管理システムを自由に選べる少人数の共同開発でも重要な Awareness 情報

ここでは、既存のエディタ、ソースコード管理システムから取得可能な少人数の共同開発にも利用可能な Awareness 情報で重要なものついて述べる。

表 2.1 に既存の共同開発環境が提供している Awareness 情報とその情報の収集元を示している。

表 2.1 の情報のうち、ファイルの過去の情報は少人数の開発では必須ではない。多人数で共同開発している状況では誰が何をしたのかという情報の把握が困難なためこの Awareness 情報は重要である。しかし、少人数の共同開発では開発者も少ないため既存のバージョン管理システムに存在するログ管理の機能で十分に互いの過去の作業を把握できる。

少人数の共同開発でも重要となるのは、互いが今行なっている作業の把握に利用できる Awareness 情報である。これは表 2.1 にある、各開発者の在席情報、各開発者の編集画面、各開発者のファイルの編集状況という Awareness 情報が該当する。

この 3 種類の Awareness 情報のうち在席情報は共同開発支援システムがウィンドウシステムを利用して提供する情報である。このためエディタやバージョン管理システムには依存せずに実現できる。

各開発者の編集画面、各開発者のファイルの編集状況の Awareness 情報はエディタ、バージョン管理システムに依存している。

各開発者の編集画面の情報はウィンドウシステムとエディタに依存している。画面上のマウスの位置、エディタ上のカーソル位置といった編集画面の情報は開発者が今興味がある位置を知るための重要な情報である。しかし、これらの情報は画面を共有してファイルを編集するような特殊なエディタを開発者全員が利用していることを想定している。このため、各開発者が異なるエディタを利用する場合は、各開発者同士のウィンドウシステム、エディタが異なるため利用できない。

各開発者のファイルの編集状況の情報では、開いているファイルの一覧、ファイルの変更箇所といった情報をエディタから収集し、ファイルをコミットしたといった情報をバージョン管理システムから取得している。ファイルの更新箇所といった情報はエディタに依存しない。このため、これらの情報は開発者がエディタを自由に選択した場合でも意味がある。つまり、エディタ、バージョン管理システムを自由に選択できる環境でも、この情報は今現在の各開発者の編集状況の把握に利用できる。

各開発者のファイルの編集状況の情報と各開発者の在席情報を既存のエディタ、バージョン管理システムを利用して取得可能にすることで、開発者がエディタ、バージョン管理システムを自由に選択可能な共同開発支援システムの作成が可能となる。

表 2.1: 共同開発に必要な Awareness 情報と依存先

情報	情報の取得元
各開発者の在席情報	ウィンドウシステム
各開発者の編集画面	ウィンドウシステム エディタ
各開発者のファイルの編集状況	エディタ バージョン管理システム
ファイルの過去の情報	バージョン管理システム

2.7 少人数の共同開発で導入が容易な共同開発支援システム

少人数の共同開発において、開発者が自由にエディタ、バージョン管理システムを選択する場合に、各開発者のファイルの編集状況、各開発者の在席情報の Awareness 情報が必要であることを 2.6 節で示した。これらの Awareness 情報を既存のエディタ、バージョン管理システムから収集可能にすることで、少人数の共同開発において自由にエディタ、バージョン管理システムを選択可能な共同開発支援システムの実装が可能となる。

また、既存のエディタ、バージョン管理システムを利用する共同開発支援システムでは導入の容易性、移植性といったことも考える必要がある。

共同開発支援システムの導入が容易でなければ、既存エディタ、バージョン管理システムを利用でき既存の環境を保ったまま導入できる利点が薄れてしまう。例えば、既存のエディタで利用ができる場合でも、そのエディタの設定やプログラミングスタイルを大幅に変えるようなものでは、導入しても新しい環境で開発を行なう場合と変わらなくなってしまう。また、各開発者が自分の好みの環境を利用する場合には、多くの環境が想定される。このため、システムが容易にその環境に対応できる必要がある。

本研究では、少人数の共同開発においても重要な Awareness 情報を既存のエディタ、バージョン管理システムから収集する、導入、移植が容易な共同開発支援システムの設計と実装を行なう。

第 3 章

既存のエディタ、バージョン管理システムからの Awareness 情報収集

少人数の共同開発においても開発を円滑に進めるには Awareness が重要である。Awareness している状態を保つことで、競合の回避や解決、他の開発者の作業の状況の確認に掛ける時間を減らすことができ、共同開発を円滑に進めることが可能となる。特に現在編集されているファイルの編集に関する Awareness 情報を提供することで、競合の回避や互いの編集状況の把握が容易となる。

この、Awareness 情報を自動的に収集し、配信することを可能にするため共同開発支援システムが利用されている。しかし、既存の共同開発支援システムでは、現在編集されているファイルの編集状況のに関する Awareness 情報の提供のためにエディタの情報を利用し、特定のエディタに依存した実装を行なっている。このため既存の共同開発支援システムでは開発者は好みのエディタ、バージョン管理システムを利用することが出来ない。

少人数の共同開発においても各開発者のファイルの編集状況の Awareness 情報は重要である。この Awareness 情報を既存のエディタ、バージョン管理システムから収集できれば、共同開発支援システムから特定のエディタ、バージョン管理システムへの依存を解消できる。その情報を利用する共同開発支援システムを構築することで自由に既存のエディタを組みあわせることが可能な共同開発支援システムを構築できる。

本章では、ある各開発者のファイルの編集状況の情報を既存のエディタ、バージョン管理システムから収集する方法について議論する。

3.1 各開発者のファイルの編集状況について

少人数の共同開発においても重要である各開発者のファイル編集状況の Awareness 情報を実現するために必要な情報をここでは述べる。各開発者のファイル編集状況というものは、各開発者がどのファイルを開いているのか、どういう編集を行なっているのかという情報、リポジトリの変更の通知という情報である。

この Awareness 情報を利用して、どの開発者が、どのファイルを開いているのかを把握することで競合の回避を開発者が行なうことが可能となる。またファイルに行なわれている編集について把握することで、互いの編集状況の把握が可能となり、他の開発者との連携を上手く行なうことが可能になる。

この情報を整理すると、以下の 5 種類の情報を収集しその情報を配信することで、ファイル編集状況の Awareness 情報が実現できる。

- 開いているファイルは何か
- ファイルに行なわれている編集は何か
- バージョン管理システムのリポジトリにファイルが追加された
- バージョン管理システムのリポジトリのファイルが変更された
- バージョン管理システムのリポジトリのファイルが削除された

この情報を既存のエディタ、バージョン管理システムから自動的に情報収集可能にすることで既存のエディタ、バージョン管理システムを利用した少人数向けの共同開発支援システムを作成することができる。

3.2 既存のエディタからの情報収集

既存のエディタを利用する少人数向け共同開発支援システムを実現するには既存のエディタから Awareness 情報を収集する必要がある。

特に少人数の共同開発においても重要である各開発者のファイル編集状況の Awareness 情報を提供するためには以下の情報をエディタから取得可能にする必要がある。

- 開いているファイルは何か
- ファイルに行なわれている編集は何か

既存のエディタから上記の情報の収集を可能にすることで、特定のエディタに依存せずに Awareness 情報を提供することが可能となる。ここでは、既存のエディタで利用できる情報の収集方法について議論する。

既存のエディタには種類が多く、動作するオペレーティングシステム (OS) も様々である。このため、多くの環境で実現が可能な情報収集の方法を考える必要がある。また、情報収集の方法自体も、収集のために必要な設定を容易に行なえるようなものが良い。収集のために必要な設定が複雑では、開発者が今まで利用していたエディタを利用できても共同開発支援システムの導入が容易ではなくなってしまい、既存のエディタをそのまま利用できる利点が薄れてしまう。

ここでは、可能な限り多くの既存のエディタで利用が可能で開発者にとって導入やの手間が少ない情報収集の手段について議論する。

3.2.1 ファイルシステムからの情報収集

ファイルの情報を定期的に集める手段としては、オペレーティングシステム (OS) のシステムコールにフックを掛けてファイルを開閉、保存する際に情報を収集するプログラムを実行するという手法がある。この手法ではほとんどのエディタで利用

することができる。また、エディタなどに設定を行なう必要もなく、システムコールの実行時にプログラムを実行するように設定するだけで動作し既存のエディタの設定を保ったまま導入を行なうことが可能である。

しかし、この手法はファイルに変更が起きたときに情報収集を行なう。このためファイルに変更が起こらないと情報収集が不可能である。つまり、エディタ上でファイルを保存して初めて情報収集を行なうこととなる。エディタ上で編集を行なっている間は情報収集を行なうことはできない。開発者が保存を怠ってしまうと情報の収集は行なわれない。競合の回避を行なおうとした場合に、競合の存在に気がつくのが遅くなってしまう可能性がある。また同様の理由で他の開発者の編集状況の確認したい場合にも、その編集情報が古いものとなってしまう可能性がある。

この手法の別の問題点としては、システムコールに依存していることである。利用しているOSがシステムコールに必ずフックを設定できるOSでしか利用できない。さらに、OSごとにプログラムを用意する必要があるという問題がある。

例えば、EmacsをLinuxで利用していた開発者が、別のOSでEmacsを利用し始めた場合には情報収集を行なう別のプログラムを用意する必要がある。

3.2.2 エディタの機能を利用した情報収集

拡張が可能な多くのエディタでは、エディタのユーザが簡単なプログラム (ユーザプログラム) を書くことでエディタの拡張が可能である。特に、ユーザプログラムはファイルの保存や編集といったイベントの発生時や定期的に外部プログラムの実行が可能である。また、ユーザプログラムでは、編集しているファイルの情報が収集可能である。ユーザプログラムで編集しているファイルの情報を定期的に収集し、その情報を外部プログラムに渡して実行することで情報収集を行なう方法がある。

たとえば、プログラマが良く使うエディタとして、ここではEmacs,Vim,Eclipse付属エディタ,VisualStudio付属エディタが挙げられる。これらのエディタはユー

ザプログラムを利用してユーザが拡張を行なう事ができ、エディタ上で発生するイベントの際に外部プログラムの実行が可能となっている。さらに、編集しているファイルの情報がユーザプログラム中で収集可能である。これらの機能を使えばエディタの情報を収集することが可能である。

この手法はファイルシステムを利用する場合と違い、現在エディタが開いているファイルの内容が収集可能である。このため競合の回避などに利用する場合、早い段階で競合に気がつき回避を行なうことが可能となる。

また、外部プログラムを実行することが出来るエディタであれば、この外部プログラムを全て共通のものにすることが可能である。さらに、この情報収集プログラムを多くの環境で動作するプログラムにすることで多くの環境で情報収集を行なうことが可能となる。

Emacs エディタのように、複数の OS で動作するエディタの場合にはファイルシステム依存の収集方法では、外部プログラムを各 OS ごとに用意する必要がある。このエディタの機能を利用する方法では、外部プログラムが複数の OS で動作する場合に外部プログラムを流用することが可能である (図 3.2)。

欠点としては外部プログラムを実行できないエディタの場合には情報収集が不可能である。またエディタに設定を必ず行う必要があり導入のための作業が必要となる。

3.2.3 本研究で使用する既存のエディタからの情報収集手法

本研究では、エディタから外部プログラムを呼び出す機能を利用してエディタの情報の収集を行なう。これは、開発者が利用する際の導入の容易さ、システム自体の移植の容易さを重視しているためである。

外部プログラムの基本的な動作はほとんどのエディタで共通化することができ、ファイルシステムに依存している方法よりも移植性を確保できる。なお、このエディタが呼び出す外部プログラムのことを「エディタ連携プログラム」と呼ぶこと

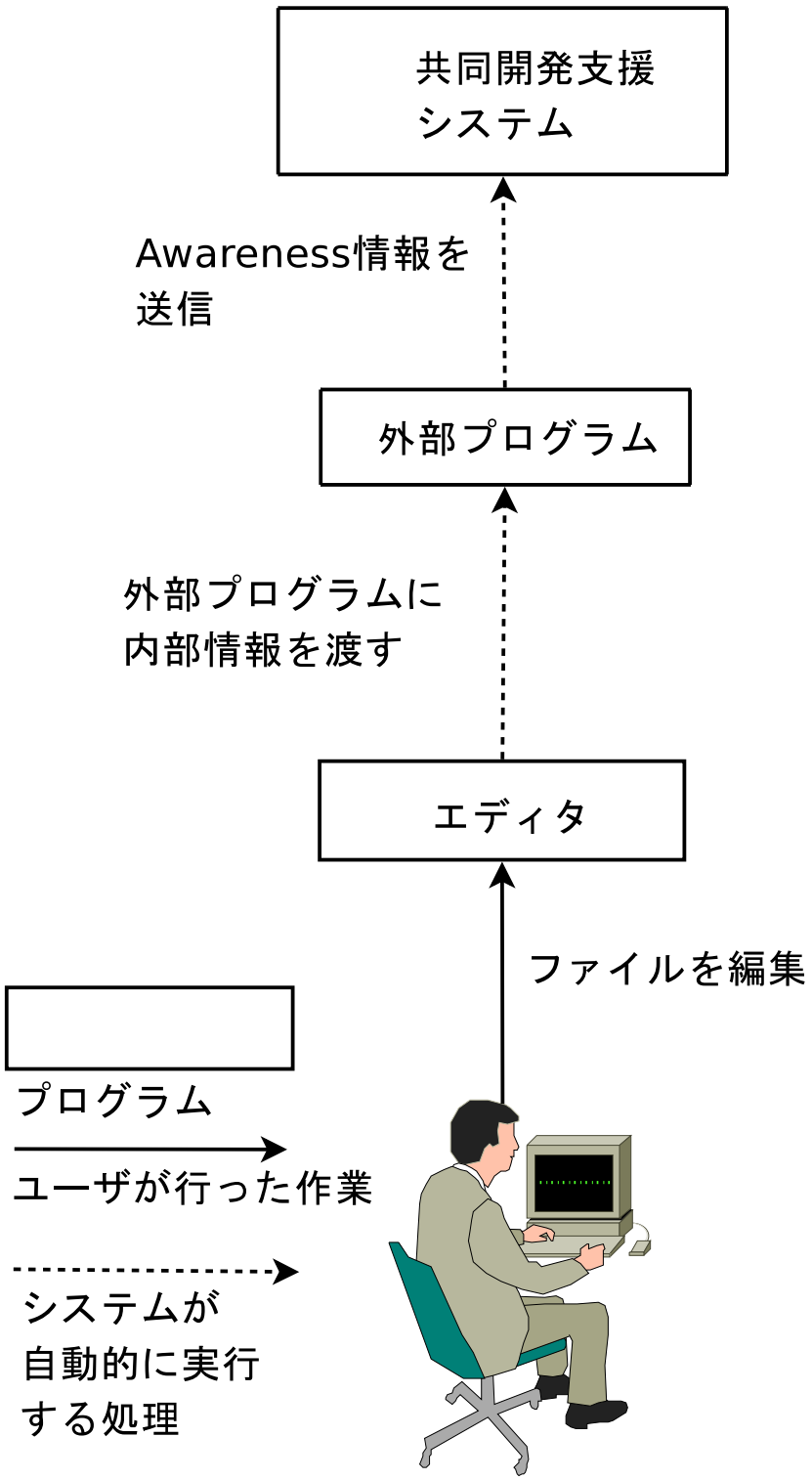

図 3.1: エディタからの情報収集

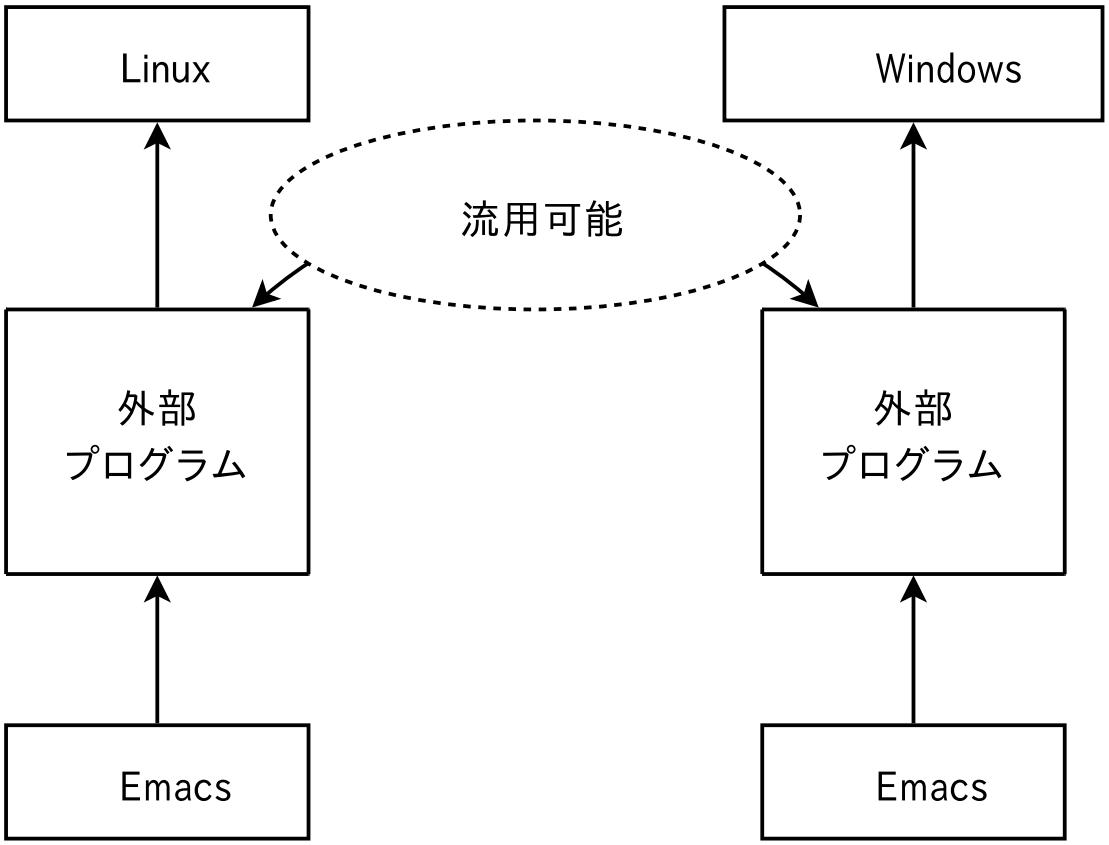

図 3.2: 異なる OS での外部プログラムの流用

とする。

また収集できる情報もファイルシステムのフックを利用する手法と比較して詳細なものを収集することができる。ファイルシステムのフックを利用する手法ではファイルを保存するまで情報を収集できないが、エディタ連携プログラムを利用する場合は、まだ保存をしていないファイルの情報も収集できる。

この手法ではファイルシステムにフックを掛ける場合と比較するとエディタに設定が必要な分若干導入は難しくなる。しかし、エディタ連携プログラムの構造を工夫することで導入を容易にすることは可能である。

また、この手法は全てのエディタで利用することはできないが、プログラマが好んで利用するエディタの多くは対応可能である。さらに、複数の OS で動作するエディタの場合には、外部プログラムも複数の OS で動作するように作成することで、外部プログラムを流用することが可能になり移植性も確保できる。

3.3 既存のバージョン管理システムからの情報収集

ここでは、既存のバージョン管理システムから Awareness 情報収集を行なう方法について議論する。バージョン管理システムはエディタと違い全ての開発者で同じバージョン管理システムを利用する。これはバージョン管理システムは全員がアクセスしソースコードを管理する共有のものだからである。このため 1 つの共同開発プロジェクトに必要なバージョン管理システムは 1 つである。この 1 つのバージョン管理システムに対応する Awareness 情報収集の方法があればその共同開発の支援を行なうことが可能となる。

バージョン管理システムから以下の情報を収集することで少人数の共同開発でも重要な各開発者のファイル編集状況の Awareness 情報を提供するために、以下の情報をバージョン管理システムから取得可能にする必要がある。

- リポジトリにファイルが追加された

- リポジトリのファイルが変更された
- リポジトリのファイルが削除された

この情報を可能な限り、多くの OS、多くのバージョン管理システムで利用できる方法で情報収集する必要がある。また、情報収集を行なうために必要な設定や準備という導入の容易さも重要である。たとえ開発者が今まで利用していたバージョン管理システムを使用であっても設定が複雑な場合、今まで利用していたバージョン管理システムを使用可能という利点が薄れてしまうからである。

ここでは、多くの OS や開発者の環境で利用できる情報収集の方法について議論する。

3.3.1 サーバからの情報収集

バージョン管理システムのサーバにコードを追加して、リポジトリに変更が行なわれた場合に共同開発支援システムに通知を行なう手法がある。この手法では、サーバにコードを追加するだけで、全てのバージョン管理システム利用者の情報を収集できる。このため、変更を行なう箇所が少なくすむ。また、バージョン管理システムのクライアントで発行された命令は必ずサーバに届くため、情報収集も一元化できる。

しかし、バージョン管理システムのサーバにコードを追加する手法では、バージョン管理システムごとに対応するコードを追加するか、既にコードの追加が完了しているバージョン管理システムに移行する必要があるため、導入のコストが増加する。また、この手法では必ずサーバに変更を行なわなければならない。このため、システム利用者がバージョン管理システムのサーバを変更する権限を持っていない場合にはシステム導入が不可能となる。

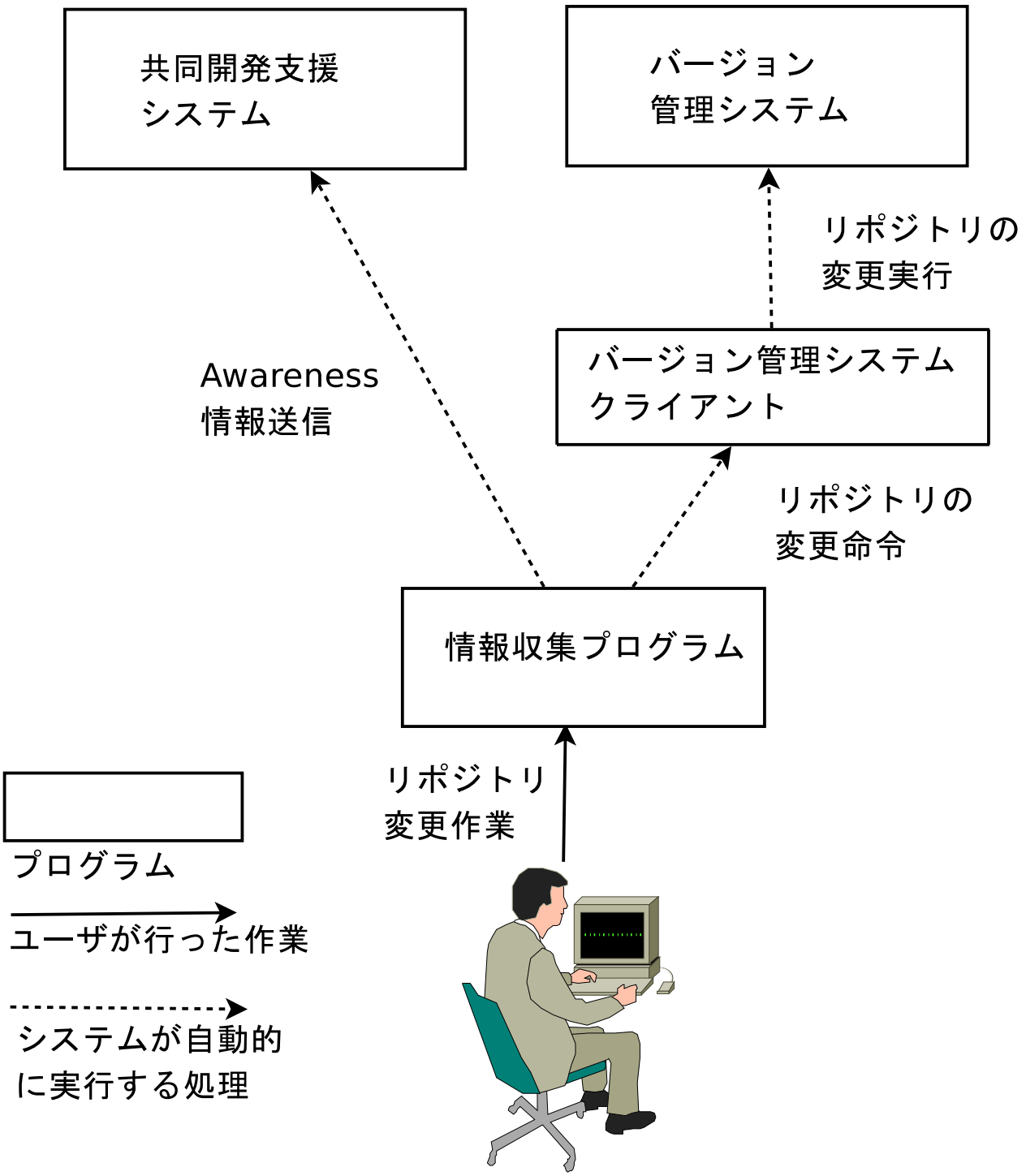

図 3.3: バージョン管理システムからの情報収集

3.3.2 クライアントからの情報収集

バージョン管理システムのクライアントに変更を加え、リポジトリに変更を行なう命令が発行された場合に、共同開発支援システムに通知を行なう手法がある。この手法では、既存の命令を呼び出す際に協調開発支援システムに通知を行なう情報収集プログラムを作成するだけで実現が可能である。図3.3がその動作の図である。

情報収集プログラムは、リポジトリに変更を行なう命令を発行する場合に、共同開発支援システムにメッセージを送信するような簡単なプログラムである。リポジトリに変更を行なわない命令を発行する場合には、そのまま普段使用しているクライアントプログラムを実行するものにしておくことで、新しいコマンドの使い方を覚える必要なく以前のコマンドと同じ使用方法で情報収集プログラムを利用できる。

情報収集プログラムを利用する手法は、情報収集プログラムを経由せずに直接バージョン管理システムのサーバと通信を行なうようなクライアントプログラムを利用している場合には適用できない。この場合、クライアントプログラムを作成した情報収集プログラムを直接利用することが可能なものに変更する必要がある。

3.3.3 本研究で利用する既存のバージョン管理システムからの情報収集手法

本研究では、情報収集プログラムを利用して、クライアントから情報収集を行なう手法を利用する。これは、開発者が利用する際の導入の容易さとシステム自体の移植性を重要視しているためである。

サーバで情報収集を行なう方法では、必ずサーバに変更を加える必要があり、サーバに変更ができない場合には導入が不可能になってしまう。クライアントに変更を行なう手法では、クライアントはユーザの権限で自由に変更ができる。この

ため、導入が不可能になる可能性が低いためである。また情報収集プログラムを移植が容易に設計しておくことで1つの情報収集プログラムを複数のOS上で動作させることが可能である。

この情報収集プログラムを本研究では「バージョン管理システム連携プログラム」と呼ぶ。

3.4 エディタ連携プログラム、バージョン管理システム連携プログラムについて

エディタ連携プログラム、バージョン管理システム連携プログラムは、システムを利用している開発者の全ての環境に応じて用意する必要がある。エディタ連携プログラム、バージョン管理システム連携プログラムが導入したい環境に対応していない場合にはエディタ連携プログラム、バージョン管理システム連携プログラムを変更しなければならない。このため1度作成したエディタ連携プログラム、バージョン管理システム連携プログラムを多くの環境で流用できるように移植性を考慮する必要がある。また、移植も既存のプログラムの流用も不可能な場合はプログラムを作成する必要がある。この場合も作成が容易である必要がある。

ここでは、エディタ連携プログラム、バージョン管理システム連携プログラムの設計について述べる。

3.4.1 エディタ連携プログラム、バージョン管理システム連携プログラムでの通信

エディタ連携プログラム、バージョン管理システム連携プログラムの両者は、Awareness情報として利用可能なエディタの情報とバージョン管理システムのリポジトリの変更の情報を共同開発支援システムに送信するプログラムである。この

プログラムは可能な限り作成や移植が容易になるように設計を行なう必要がある。

このため、既存の複雑な通信プロトコルではなく実装が容易な単純なプロトコルで情報を送信する。このプロトコルは可能な限り簡単なものにし移植や作成を容易にする必要がある。通信は複雑に送受信を行なうプログラムではなく単純に送信だけを行ない、送信する順番なども考慮せず、無秩序に情報を送信するだけのものにする。こうすることで、プログラムの構造を単純化することができ、作成や移植を容易に行えるプログラムにすることができる。

第 4 章

CAEDE:プログラミング環境と連携する協調開発支援ミドルウェア

既存のエディタ、バージョン管理システムから Awareness 情報の収集、配信を行なうミドルウェア CAEDE を実装した。ここでは、対応するエディタ、バージョン管理システムとして Emacs、CVS を選択した。CAEDE では対応するエディタ、バージョン管理システムを容易に変更できる。このため、他のエディタ、バージョン管理システムにも対応することができる。

CAEDE は、既存のエディタ、バージョン管理システムから Awareness 情報を収集する。CAEDE の導入の際には、既存のエディタ、バージョン管理システムから Awareness 情報を収集するための簡単な設定を行なうだけで導入できる。

本章ではこの CAEDE のの詳細、およびその通信方法の詳細について述べる。

4.1 共同開発支援システム:CAEDE

CAEDE は、既存のエディタ、バージョン管理システムと連携し Awareness 情報を取得しそれを通知する共同開発支援システムである。CAEDE は特定のエディタ、バージョン管理システムに依存しないため、開発者は好みのエディタを利用して共同開発を行なうことができる。

CAEDE の導入は既存のエディタ、バージョン管理システムを維持したまま行な

うことが可能である。CAEDE の導入には、Emacs にエディタ連携プログラムの transcaede を呼び出す設定、cvs コマンドをバージョン管理システム連携プログラムの mycvs コマンドに置きかえるという作業が必要である。しかし、これらの設定は簡単に行なうことができ CAEDE を導入することは容易である。

CAEDE を利用する事で、Awareness 情報である各開発者のファイルの編集状況、各開発者の在席情報を確認しながら作業を進めることが可能になる。さらに CAEDE には連絡の手段としてインスタントメッセージ (IM) の機能もある。また、確実に Awareness 情報に気づくための通知機能も提供している。

CAEDE を利用して共同開発を行なうことで Awareness 情報を逃さず把握し、他の開発者の状況、ファイルの編集内容を確認しながら作業を進めることが可能となる。これにより、開発者同士で競合の回避や他の開発者のファイルの編集内容の把握が可能となり共同開発を円滑に進める事が可能となる。

4.2 CAEDE が提供する Awareness 情報

ここでは CAEDE が提供する Awareness 情報について述べる。CAEDE が提供する Awareness 情報はその提供する機能によって 2 つに分類される。

- 各開発者のファイルの編集状況 (タスク情報)
- 各開発者の在席情報 (ユーザ情報)

タスク情報は、開発者が開いているファイルの情報やリポジトリの変更情報といったファイルに関する Awareness 情報である。これらの情報はエディタ、バージョン管理システムの内部から情報を取得する必要がある。このため、タスク情報は既存のエディタ、バージョン管理システムから情報を集めて実装されている。

CAEDE がタスク情報として提供する情報は以下の 5 種類である。

- 他の開発者の開いているリポジトリ上のファイルの一覧

- 他の開発者が行なったリポジトリ上のファイルへの編集
- リポジトリにファイルが追加された
- リポジトリのファイルが更新された
- リポジトリのファイルが削除された

この情報を利用することで、開発者は競合の回避や他の開発者の編集の状況を把握でき円滑に共同開発を進めることを可能にする。

ユーザ情報は開発者の在席の状況を表わす情報である。CAEDE では開発者の状態を開発者が席を離れている時間、仕事の状況に応じて 5 つに分類した。表 4.1 に開発者の状態の一覧を示す。この情報を利用して開発者は他の開発者の状態を確認してから、IM を利用して連絡を行なうことが可能となる。なおこの状態の名前は一般的な IM で利用されているものと同じものにした。

表 4.1: 開発者の状態の一覧表

状態名	意味
Offline	CAEDE を起動していない開発者の状態。
Online	在席していて連絡を取れる場合に使用する状態。
Busy	作業中なので、話かけて欲しくない場合に使用する状態。
Be right back	離席しているが、すぐ戻ってくる場合に使用する状態。
Away	離席していて、当分戻ってこない場合に使用する状態。

4.3 CAEDE を利用した共同開発の方法

CAEDE は既存のエディタ、ソースコード管理システムの CVS を利用した共同開発支援システムである。共同開発でメンバは Emacs を利用し、端末を利用して CVS を利用しているような開発スタイルに適している。図 4.1 に CAEDE を利用した際の共同開発の流れを示した。開発者はこの作業の間は常に CAEDE クライアントを起動しておくだけで CAEDE を利用する共同開発を行なうことができる。

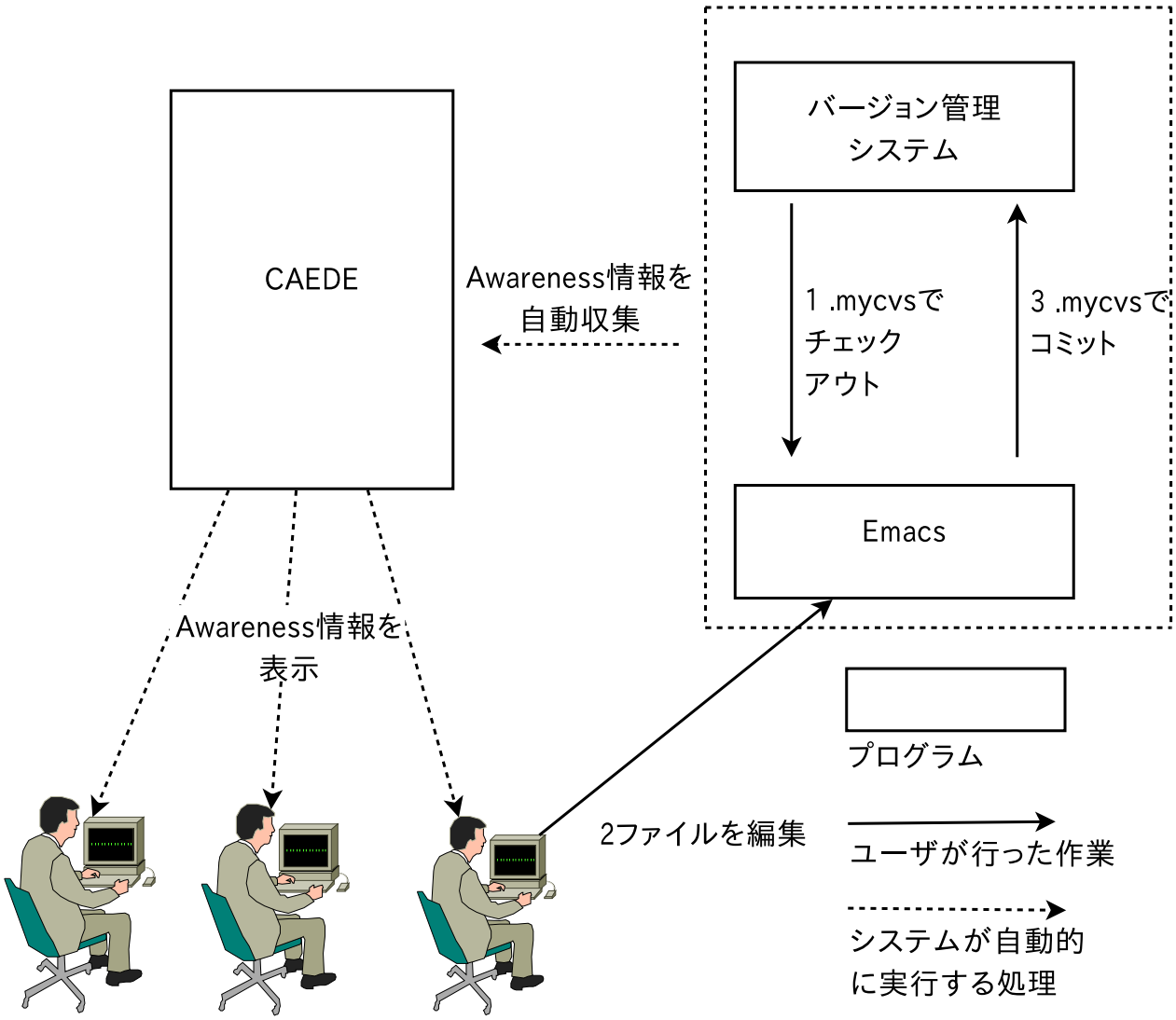

図 4.1: CAEDE を利用した共同開発

CAEDE を使った共同開発は以下の様な流れで行なわれる。

1. mycvs でリポジトリを手元にチェックアウトする。
2. Emacs でファイルを編集する。
3. mycvs でファイルへの変更をリポジトリにコミットする。

CAEDE を利用する場合の共同開発はほとんどバージョン管理システムを利用した共同開発と変わらない。cvs を利用する際のコマンドが mycvs に変わる程度である。

この作業の間、CAEDE は Emacs、CVS から自動的に Awareness 情報を収集し、Awareness 情報を開発者に表示する。CAEDE クライアントを起動するだけで開発者にほとんど意識させることなく Awareness 情報を収集し表示することができる。このため、作業の間、開発者は常に Awareness 情報を確認できる。

CAEDE を利用することで、Awareness 情報を自動的に受けとることが可能となり、Awareness している状態を保ったまま共同開発を進めることができる。

4.4 CAEDE の構造

CAEDE はサーバクライアント型の共同開発支援システムとなっている。

図 4.2 が CAEDE の概観である。開発者の計算機で直接動作する CAEDE クライアントとそのクライアントの通信相手となるサーバ部の CAEDE サーバから構成されている。

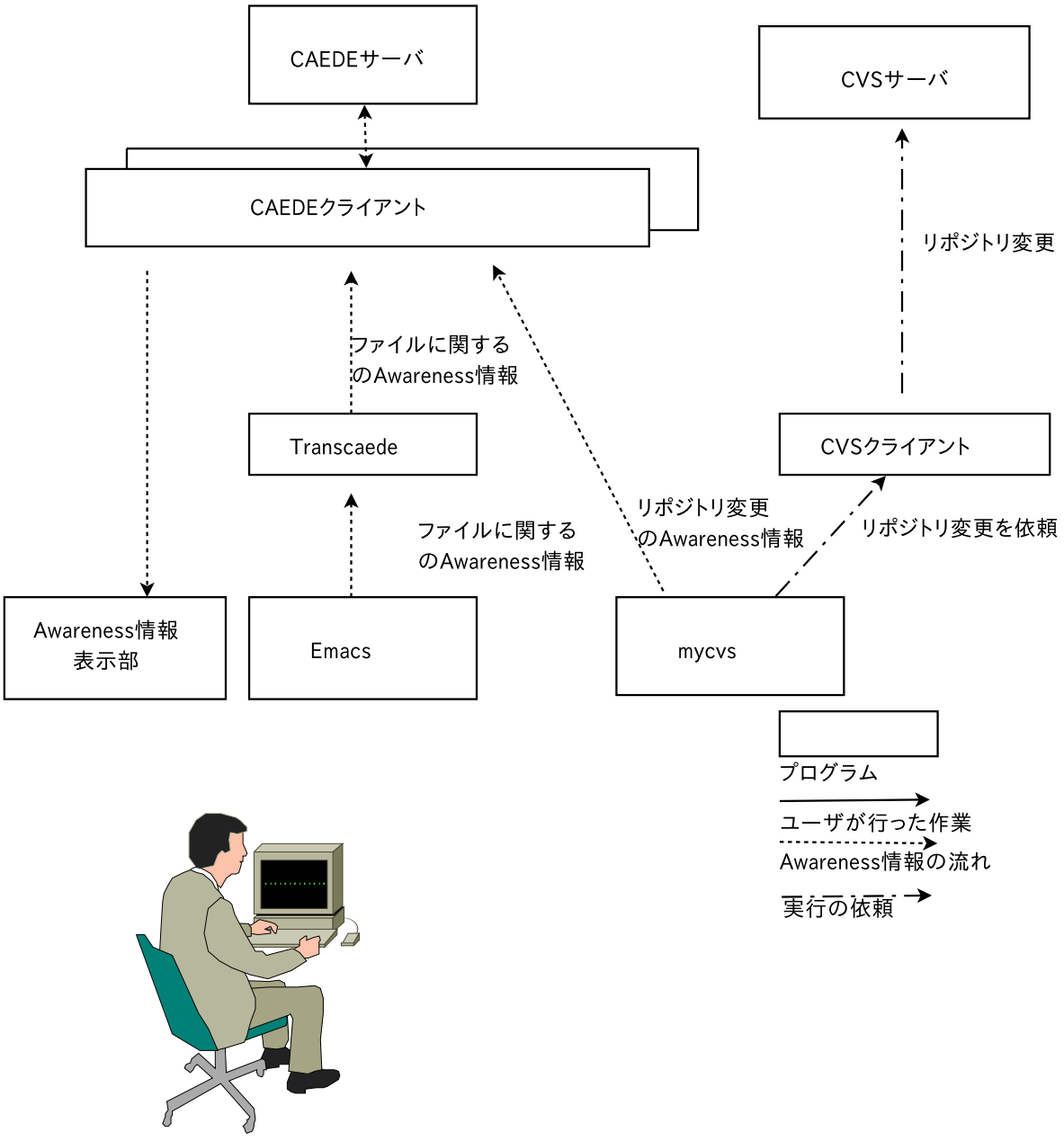

図 4.2: CAEDE の概観

CAEDE クライアントはエディタ連携プログラム、バージョン管理システム連携プログラムを通じてエディタ、バージョン管理システムからファイルに関する情報を収集する。その集めた情報を CAEDE サーバに送信するプログラムである。

CAEDE サーバはこの集めた情報を統合し、Awareness 情報として CAEDE クライアントに送信する。CAEDE クライアントがその情報を表示することで Awareness な状態を実現した。

また、共同開発支援システムを開発する場合には、その複雑さから様々な共同開発支援システム用のライブラリが提案されている [2, 8, 5]。しかし、ソフトウェア開発の手法は時代によって変化していくため、拡張性が無いものは成功できない。多くのソフトウェア開発の手法に対応するため共同開発支援を行なうシステムは拡張性が高い必要がある [6]。この考えに基づき本研究で提案する CAEDE は、可能な限りモジュール化を行なうことで高い拡張性を意識したシステムを構築している。

4.5 CAEDE クライアント

ここでは CAEDE クライアントについて解説する。CAEDE を利用する共同開発では各開発者は 1 人 1 つ CAEDE クライアントを起動する。CAEDE クライアントは Ruby で 500 行程度のプログラムである。Awareness 情報の表示を行なう GUI の実装を行なうために Ruby/tk を利用している。

この CAEDE クライアントは 4 つの機能を開発者に提供する。

- Awareness 情報の収集および送受信
- Awareness 情報の表示
- Awareness 情報の通知
- インスタントメッセージング (IM)

CAEDE クライアントは起動している開発者の Awareness 情報を、エディタ連携プログラムの transcaede、バージョン管理システム連携プログラムの mycvs を通じて収集する。CAEDE は、エディタ連携プログラムの transcaede を通じて、Emacs からエディタの情報収集を行なう。Emacs から収集した情報は、CAEDE サーバに送られて管理され、CAEDE サーバから他の開発者へ自分の作業状況を伝える Awareness 情報となる。また CAEDE ではバージョン管理システムとして CVS を利用していて、CVS の情報は mycvs というラッパプログラムを通じて CAEDE に送信される。

Awareness 情報は単純に表示するだけでは、他の開発者の作業状況が変ったことに開発者が気がつかないこともある。このため、他の開発者の作業状況を確認するためのウィンドウに加えて、Awareness 情報を通知する機能もある。

開発者は他の開発者と簡単な連絡をとるためにインスタントメッセージ機能も利用できる。これは、指定した相手と会話するためのウィンドウを開いて、短いメッセージをやりとりできるものである。

CAEDE のクライアントは図 4.3 で示すような構成となっている。CAEDE クライアントは大きく分けて 3 つのモジュールグループから構成されている。まず、CAEDE サーバとのメッセージの送受信を担当するモジュールグループのメッセージディスパッチャ、メッセージトランスミッタ。次に、受信したメッセージを処理するモジュールグループのユーザ管理モジュール、メッセージングモジュール、タスク管理モジュール。そして、Awareness 情報を表示し、開発者が操作するユーザインタフェースのモジュールグループの Awareness 情報表示部、通知モジュールの 3 種類である。これらの各モジュールが連携して CAEDE クライアントは動作する。

以下の節では、CAEDE クライアントの各構成要素について議論する。

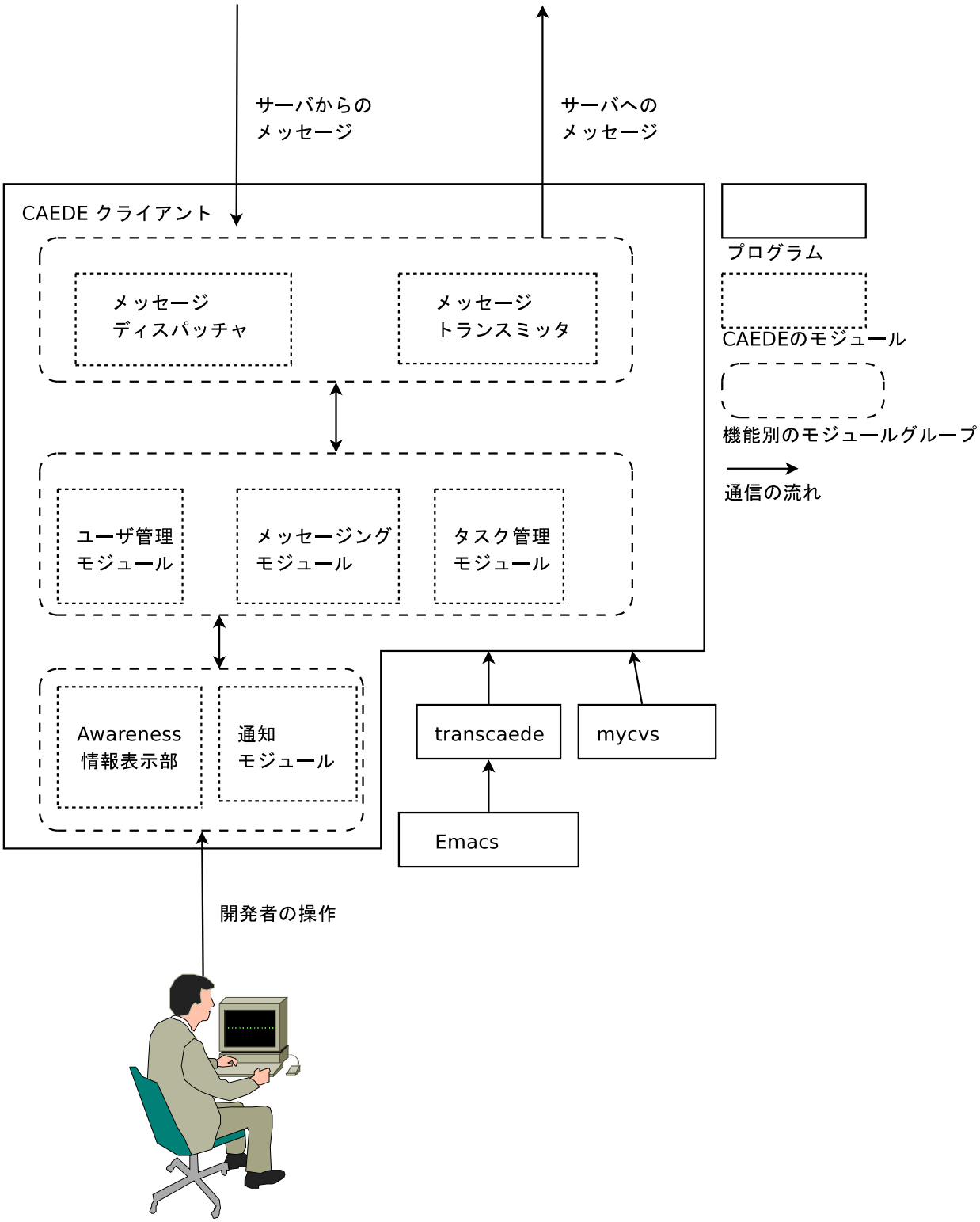

図 4.3: CAEDE クライアントの概観

4.5.1 メッセージディスパッチャ、メッセージトランスミッタ

CAEDE クライアントと CAEDE サーバの通信はメッセージディスパッチャとメッセージトランスミッタという 2 つのモジュールを通じて行なわれる。

メッセージディスパッチャは CAEDE サーバから送信されたメッセージを受け取り、それを処理するモジュールである。メッセージディスパッチャはメッセージを受けとるとメッセージを処理し、適切なモジュールにメッセージを送信する。この際、Awareness 情報の更新が行なわれていた場合には、さらに Awareness 情報表示部にもメッセージが送られる。Awareness 情報表示部はメッセージを受けとると即座に Awareness 情報を更新する。

メッセージトランスミッタは、CAEDE クライアントから CAEDE サーバにメッセージを送信するモジュールである。CAEDE クライアントから送信される IM、エディタ、バージョン管理システムから収集した Awareness 情報は全てこのメッセージトランスミッタを通じて CAEDE サーバへ送信される。

4.5.2 ユーザ管理モジュール、タスク管理モジュール

CAEDE クライアントはユーザ管理モジュール、タスク管理モジュールの 2 つのモジュールで、現在のユーザ情報、タスク情報の管理を行なう。これらのモジュールから情報を取得することで Awareness 情報表示部は Awareness 情報をユーザに表示する。

特にタスク管理モジュールは、CAEDE の内部情報だけでなく、エディタ連携プログラム、バージョン管理システム連携プログラムと通信を行ない、ユーザが使用しているエディタやバージョン管理システムからの情報収集を担当している。

Awareness 情報表示部を通じて開発者の在席情報が変更された場合や、エディタ、バージョン管理システムからの情報収集が行なわれた際には、この 2 つのモジュールが情報をメッセージトランスミッタに渡し、CAEDE サーバに送信する。

4.5.3 メッセージングモジュール

CAEDE では CAEDE のメッセージング機能を利用して、IM も実装している。これを利用する事で開発者は簡単な連絡を取ることができる。メッセージングモジュールは他の CAEDE クライアントとの IM を行なうモジュールである。この IM を利用することで、開発者は他の開発者との連絡を行なうことができる。

4.5.4 通知モジュール

Awareness を促すためには、表示しているだけでは開発者が気がつかない場合がある。このため CAEDE から開発者に情報の更新を伝える必要がある。このために、CAEDE では通知を利用している。

この通知は通知モジュールを通じて行なわれる。Awareness 情報の到着時に、通知モジュールがポップアップウィンドウを使った通知を行なう (図 4.4)。この通知は growl という通知を行なうソフトウェアを利用している。

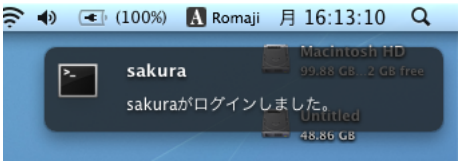

図 4.4: CAEDE の通知

通知は CAEDE サーバから通知メッセージが到着した際に行なわれる。通知メッセージは以下のタイミングで送信される。

- リポジトリにファイルを追加した時
- リポジトリのファイルの変更を行なった時
- リポジトリのファイルを削除した時

- ファイルをエディタで開いた時
- ファイルを編集している時
- ファイルをエディタで閉じた時

通知によって、開発者はこれらの Awareness 情報を確実に把握することができる。

4.5.5 Awareness 情報表示部

Awareness 情報表示部は、ユーザ管理モジュール、タスク管理モジュールを利用して取得した Awareness 情報を表示するモジュールである。また開発者が IM や在席の状態を変更するためのインタフェースにもなっている。

Awareness 情報表示部は、図 4.5 のように各開発者が現在編集しているファイルの情報を Emacs および CVS ソースコード管理システムを通じて収集し可視化する。また常駐しても困らない大きさでサイドバーとして利用できるものにした。この大きさのウィンドウで常駐するものならば開発者は長く利用する傾向にあるためである [1]。

このウィンドウには、プロジェクトに参加している開発者の在席情報、その開発者の開いているファイルのリストが表示される。このリストは、他の開発者の現在の状況に応じて変更されるようになっており、ファイルを開いたり、閉じたりすることによって適宜更新されていく。開発者はこの画面を見るだけで、他のメンバの作業の状況を一望することができ、同じファイルを開かないように、競合の発生を回避することが可能となる。またアイコンの色で在席の状況を表わすようになっている。この対応表を表 4.2 に示す。表の左が状態で右側がそのときのアイコンの色である。

図 4.5 のウィンドウの左上にあるメニューで開発者は自分の在席の状態を変更することができる。

図 4.5: CAEDE の Awareness 情報表示部

表 4.2: 在席状態と色の対応表

状態	色
Offline	赤
Online	緑
Busy	青
Be right back	黄
Away	紫

ファイルのリストをクリックする事で、図 4.6 のように他の開発者が編集しているファイルを確認でき、他の開発者がどのような変更を行なっているのかを確認することができる。

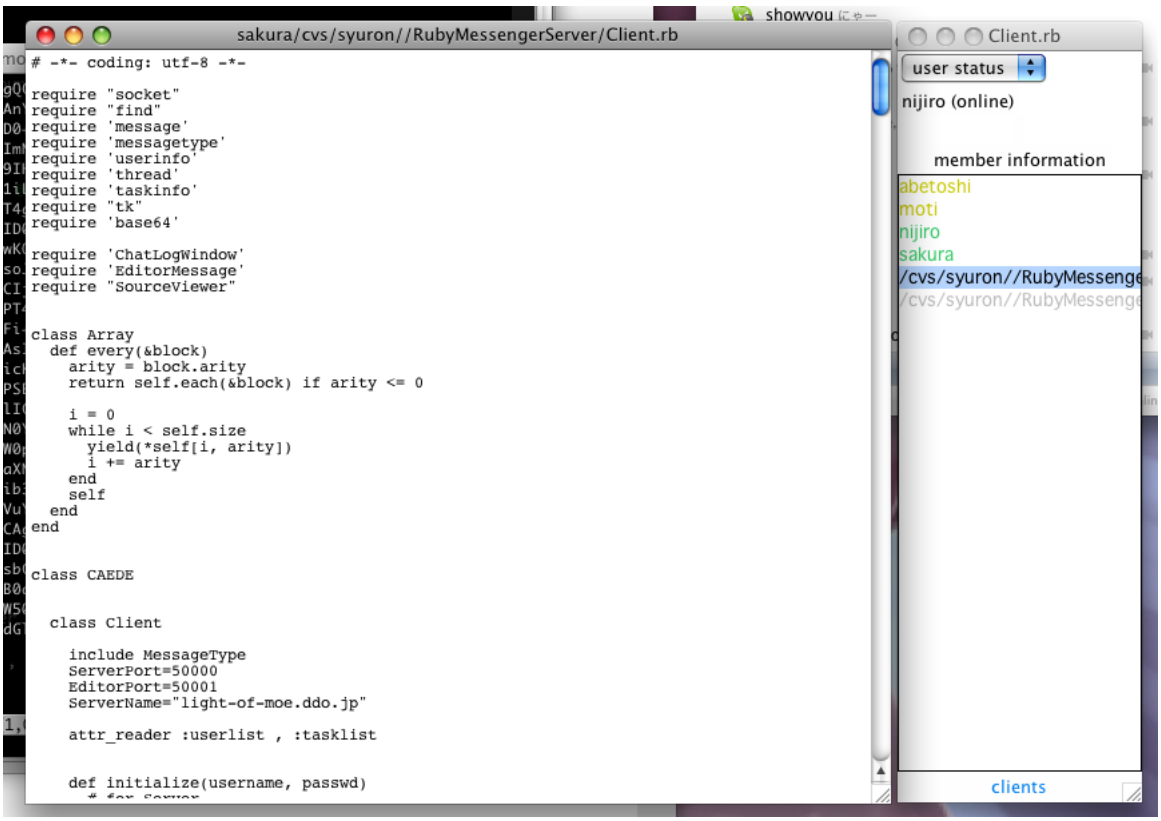

図 4.6: 他のユーザの編集状況の確認

開発者のアイコンをダブルクリックすると、図 4.7 のような開発者とのインスタントメッセージング用のウィンドウが開く。このウィンドウに文字を入力していくことで、インスタントメッセージングが可能である。

4.6 CAEDE サーバ

CAEDE サーバは CAEDE で利用される全ての Awareness 情報を管理している。Ruby で作成された 1000 行程度のサーバプログラムである。

図 4.7: CAEDE の IM 機能

CAEDE クライアントがエディタ、バージョン管理システムから収集したユーザ情報、タスク情報は、CAEDE サーバに送られ CAEDE サーバで一元管理されている。これらの情報は CAEDE クライアントから送信されてくるメッセージに基づいて、そのときの各開発者の状況に応じて更新されていく。また、ユーザ情報やタスク情報に更新があった場合に、CAEDE サーバは通知メッセージを各クライアントに送信する。例えば、ある CAEDE クライアントがファイルを開いたというメッセージを CAEDE サーバに送信した場合、CAEDE サーバはメッセージを処理し、内部のユーザ情報、タスク情報を更新し、他の CAEDE クライアントに対してファイルが開かれたという通知メッセージを送信する。

CAEDE サーバ内部のメッセージの流れを図 4.8 に示した。CAEDE では以下の流れでメッセージは処理される。

1. CAEDE クライアントからメッセージを送信。

2. CAEDE サーバにメッセージが到着。メッセージレシーバがメッセージを受

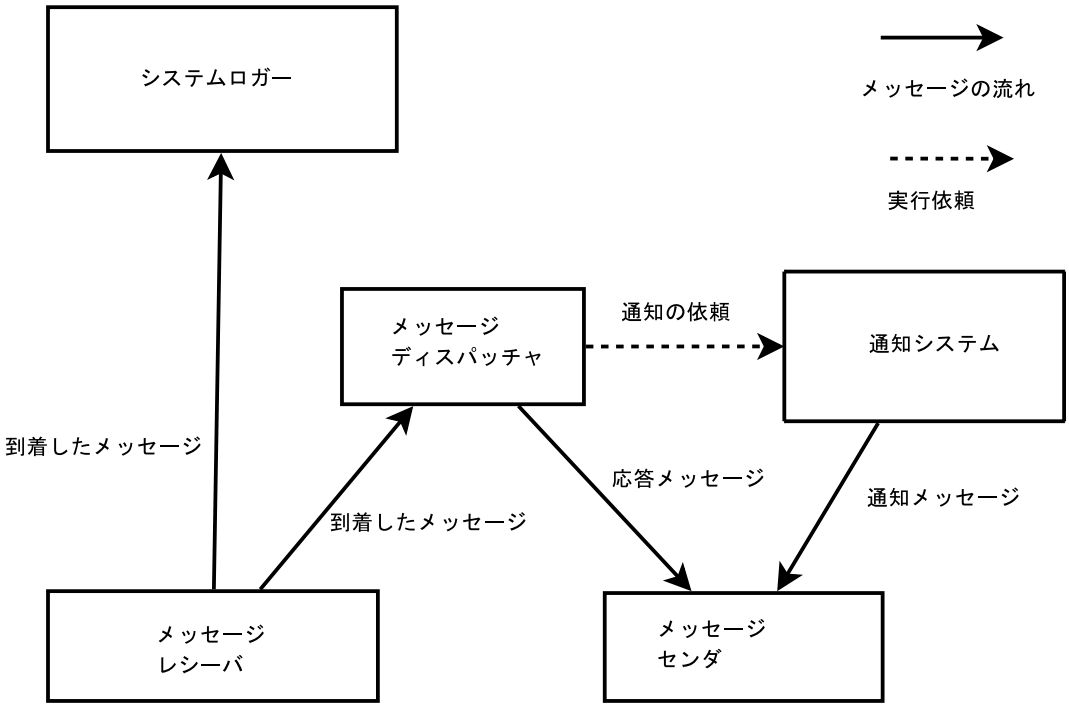

図 4.8: CAEDE サーバ内部のメッセージの流れ

けとり、メッセージがログに追加される。

3. メッセージディスパッチャがメッセージを処理。応答メッセージがメッセージセンダに渡る。

4. ディパッチャは処理する際、通知を出す必要がある場合には通知システムに依頼を出す。

5. 応答メッセージがメッセージセンダから CAEDE クライアントに送信される。必要がある場合には通知メッセージもメッセージセンダから CAEDE クライアントに送信される。

このような動作で CAEDE サーバは CAEDE クライアントに適切な Awareness 情報を常に提供する。

4.7 CAEDE プロトコル

CAEDE で行なわれる通信は CAEDE プロトコルという独自のもので行なわれる。この CAEDE プロトコルに従って、CAEDE サーバ、CAEDE クライアント、バージョン管理システム連携プログラム、エディタ連携プログラムの通信は行なわれる。

バージョン管理システム連携プログラム、エディタ連携プログラムを作成する場合は、CAEDE クライアントにこのプロトコルに沿ったメッセージを送信する必要がある。このためこのプロトコルを単純にし、これらのプログラムを容易に実装可能にした。

ここではこの CAEDE プロトコルについて述べる。

4.7.1 CAEDE プロトコルのメッセージ

CAEDE プロトコルでは、メッセージという単位で通信は行なわれる。このメッセージというのは改行文字で終了するような行指向の文字列である。

また、CAEDE プロトコルではメッセージは送信する順番に指定のない非同期なメッセージプロトコルとなっている。このような単純なプロトコルにすることで、送信の際にも順番などを考慮する必要が無いものとした。

CAEDE で使用しているメッセージ形式について述べる。図 4.9 が CAEDE で使用されているメッセージの形式である。

図 4.9: CAEDE プロトコルのメッセージ

メッセージの形式はテキストベースで、各要素は ':' で区切られているという単純なものである。各項目の意味は以下で述べる。

- id

  id は、各メッセージに振られる番号であり、同時刻に送られたメッセージを識別するために利用される。id はメッセージが送信される毎に値が 1 ずつ増加されていくものである。

- time

  time はメッセージが送信された時刻である。

- from

  from は送信してきた開発者の名前である。この情報を利用して IM などでは送信者を特定する。

- to

  to は送信する開発者の名前である。この情報で CAEDE サーバは IM を適切な相手に送信できる。

- method

  この情報でこのメッセージの処理の方法が決まる。この method(メソッド) の種類によっては content に必要な情報を収めて送信する。

- content

  content は、メソッドによって内容が異なる。開発者名、ファイル名、ファイルの内容、インスタントメッセージといったメソッドに必要な情報が収められて送信される。content は複数の情報を収める必要がある、それぞれの情報の区切りとして ':' を使用している。

各メッセージは id、time、from、to で識別される。time、from、to だけでは完全に同じ時間同じ相手に送ったメッセージの識別ができない。このためメッセージごとに id を振ることで識別可能にしている。content の内容はファイル名、開発者名、ファイルの内容が収められる。

content におけるファイル名とファイルの内容は注意する必要がある。ファイル名は各開発者のローカルのファイル名ではなく常にバージョン管理システムのリポジトリにおける位置で指定する。これは、各開発者ごとにリポジトリをチェックアウトした場所が違うため、各開発者ごとに違うファイル名になるためである。リポジトリにおける位置で指定すればどのクライアントから見ても同じファイルを指すことが可能である。このためファイルは常にバージョン管理システムのリポジトリにおける位置で指定する。

またファイルの内容を content に含み送信する場合は、そのままファイル内容を content として送信してしまうと問題が発生する。ファイルの内容によっては、':' や改行を含んでいるため区切りがうまく動作できなくなり送信できない。このため

CAEDE プロトコルでは Base64 エンコードでエンコードしてから送信している。Base64 エンコードを行なうことで全ての文字を ':' を含まない文字に変換することでき、問題無く content に含めて送信することが可能となる。

CAEDE プロトコルは簡単に解析、メッセージ生成が可能な形式である。このためエディタ連携プログラム、バージョン管理システム連携プログラムの構造を単純にすることができた。

4.7.2 CAEDE のメッセージの種類

CAEDE で使用しているメソッドは機能別の 6 種類に分類される。

- ファイル関連

  ファイル関連のメッセージは、ファイルの編集に関する Awareness 情報の送受信に利用される。このメソッドで CAEDE クライアントは既存のエディタからファイルに関する Awareness 情報を収集し、CAEDE サーバに送信する。

- バージョン管理システム関連

  バージョン管理システム関連のメッセージは、リポジトリに関する Awareness 情報の送受信に利用される。リポジトリにファイルの追加、削除、ファイルの更新が行なわれた際に送受信される。このメソッドで CAEDE クライアントは既存のバージョン管理システムから Awareness 情報を収集し、CAEDE サーバに送信する。

- 通知関連 CAEDE サーバがクライアントに対して送信する通知のメッセージである。このメッセージを受け取ったクライアントはクライアントの通知機能を使いユーザに通知を行なう。

- システム関連

  システム関連のメッセージは、主にユーザの認証、ログイン、ログアウトに関連するメッセージである。

- ユーザ情報

  ユーザ情報というのは、開発者の現在の状態を表す情報である。「Offline」「Online」「Busy」「Be right back」「Away」の 5 種類の状態に開発者の状態を変更することができる。

- メッセージ機能

  インスタントメッセージなどユーザ間の通信を行なうためのメッセージである。

4.7.3 CAEDE のメッセージの詳細

ここで CAEDE プロトコルにおけるメソッドとそのメッセージの意味、送られるタイミング、content の内容を示す。CAEDE はこれらのメソッドを組み合わせることで動作する。

- ユーザ情報関係

  - UPDATE_USERS

    開発者の状態に変更が合った時に CAEDE サーバから CAEDE クライアントに送信される。content には全員分のユーザ情報が格納されている。

  - UPDATE_STATUS

    開発者が状態を変更するときに CAEDE クライアントから CAEDE サーバに送信される。content には変更する状態が格納されている。このメッセージを受け取ると CAEDE サーバはユーザ情報を管理しているリストを更新し、UPDATE_USERS メソッドのメッセージを送信する。

- ファイル関連

  - UPDATE_TASKS

    タスク情報の変更時に CAEDE サーバから CAEDE クライアントに送

信される。content には各開発者の現在編集しているファイルの編集状況が格納されている。

- FILE_OPEN

  開発者がエディタでファイルを開いたときにエディタ連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送られる。content には開いたファイル名、開いたファイルの内容が格納されている。CAEDE サーバはこのメッセージを受け取るとファイルの編集状況を管理しているリストを更新する。

- FILE_EDITING

  開発者がエディタでファイルを編集中に定期的に、エディタ連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送られる。content には編集中のファイル名が格納されている。CAEDE サーバはこのメッセージを受け取るとファイルの編集状況を管理しているリストを更新する。

- FILE_CLOSE

  開発者がエディタでファイルを閉じたときに、エディタ連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送られる。content には閉じたファイル名が格納されている。CAEDE サーバはこのメッセージを受け取るとファイルの編集状況を管理しているリストを更新する。

- REQ_FILEINFO

  指定した開発者が編集しているファイルの編集状況を取得するためのメソッドである。CAEDE クライアントから CAEDE サーバに送信される。content には開発者名とファイル名が格納されている。

- RECV_FILEINFO

REQ_FILEINFO の要求に対して CAEDE サーバが CAEDE クライアントに対して送信するメッセージである。このメソッドの content には REQ_FILEINFO で指定した開発者、ファイルの現在編集している状況が格納されている。

- バージョン管理システム関連
  - FILE_ADD

    リポジトリにファイルが追加されたときに、バージョン管理システム連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送信される。content には追加されたファイルの名前が格納されている。
  - FILE_COMMIT

    リポジトリでファイルが更新されたときに、バージョン管理システム連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送信される。content には更新されたファイルの名前が格納されている。
  - FILE_REMOVE

    リポジトリにファイルが削除されたときに、バージョン管理システム連携プログラムから CAEDE クライアントに送信され、CAEDE クライアントから CAEDE サーバに送信される。content には削除されたファイルの名前が格納されている。
- 通知関連
  - ALT_OPEN

    開発者がエディタでファイルが開いたときにサーバから全クライアントに対して送信されるメッセージ。content には開かれたファイル名、開

いた開発者名の情報が格納されている。

- ALT_CLOSE

  開発者がエディタでファイルが閉じたときにサーバから全クライアントに対して送信されるメッセージ。content には閉じられたファイル名、閉じた開発者名の情報が格納されている。

- ALT_ADD

  リポジトリにファイルが追加されたときに全CAEDE クライアントに送信される。content には追加されたファイル名が格納されている。

- ALT_COMMIT

  リポジトリのファイルが更新されたときに全CAEDE クライアントに送信される。content には更新されたファイル名が格納されている。

- ALT_REMOVE

  リポジトリのファイルが削除されたときに全CAEDE クライアントに送信される。content には削除されたファイル名が格納されている。

- システム関連

  - SYS_LOGIN

    CAEDE クライアントがログインするときに CAEDE サーバに送信するメッセージ。content には開発者名とパスワードを格納されている。CAEDE サーバではその情報をもとにして認証を行なう。

  - AUTH_LOGIN

    SYS_LOGIN メッセージで送られてきたメッセージの開発者名とパスワードの対が、登録されている情報と一致した場合に AUTH_LOGIN メッセージを SYS_LOGIN を送ってきた CAEDE クライアントに送信する。

- SYS_ABSENT

  CAEDE クライアントが接続を切るときに送信するメッセージである。

- メッセージ機能

  - SEND_MSG

    このメッセージはクライアント同士の通信に使うためのものである。開発者同士の連絡手段として利用することを想定している。content にはメッセージの本文が格納されている。

4.8 CAEDE とエディタの連携

CAEDE はエディタと連携して動作する。ここでは、CAEDE で利用することができるエディタとその連携に使用するプログラムについて議論する。

4.8.1 CAEDE で利用できるエディタ

CAEDE では、Emacs を対応エディタとして選択している。CAEDE は特定のエディタに依存する実装にはなっていないため、以下の機能を持っているエディタは連携させることが可能である。

- ファイルを開閉時とファイルの編集中定期的にユーザスクリプトで transcaede を実行することができる。
- transcaede の引数を正しく設定できる

プログラマが好んで使用するようなエディタ (Emacs,Vim,Eclipse 付属エディタ、VisualStudio 付属エディタ) は、上の条件を満たしている。

また簡単な設定を行うだけで CAEDE とエディタの連携が可能である。Emacs で行なった設定の例をソースコード 4.1 に示す。Emacs Lisp で必要となる設定も短

く、30 行程度の設定できた。ここでは add-hook を利用して、一定時間ごと、ファイルを開いた時、ファイルを閉じた時に、エディタ連携プログラムの transcaede を適切な引数で呼びだすだけの簡単なものである。

transcaede は多くの OS で動作するスクリプト言語の Ruby で作成している。このため OS が違う場合でも、Emacs が動作すれば上記の Emacs Lisp の設定さえ行なえば CAEDE と連携して動作する。

ソースコード 4.1: Emacs に行なった設定

```
1
2  (defvar transcaede-command "/Users/sakura/syuron/RubyMessengerServer
       /utils/transcaede.rb")
3
4  ;; ファイルを開いたときに transcaede-open-hook 関数を呼び出す設定
5  (add-hook 'find-file-hook 'transcaede-open-hook)
6
7  ;; ファイルを閉じたときに transcaede-close-hook 関数を呼び出す設定
8  (add-hook 'kill-buffer-hook 'transcaede-close-hook)
9
10 ;; emacs 起動中は定期的に transcaede-update-hook を実行する設定
11 (run-at-time 30 30 'transcaede-update-hook)
12
13 ;; ファイル名を取得する関数
14 (defun get-filename (buffer)
15   (expand-file-name (buffer-name buffer)))
16 ;; エディタで編集中のファイルの内容を取得する関数
17 (defun get-file-content (buffer)
18  (buffer-substring-no-properties (point-min) (point-max)))
19
20 ;; hook functions
21 ;; transcaede を実行してファイルを開いたという情報を CAEDE に伝える関数
22 (defun transcaede-open-hook ()
23   (start-process "transcaede" "*transcaede*" transcaede-command "open"
24     (get-filename (current-buffer)))
25     (get-file-content (current-buffer)))
26
27 ;; transcaede を実行してファイルを閉じたという情報を CAEDE に伝える関数
28 (defun transcaede-close-hook ()
29  (start-process "transcaede" "*transcaede*" transcaede-command "close"
30     (get-filename (current-buffer))
```

```
31      (get-file-content (current-buffer)))
32
33 ;; ファイルを編集中定期的に呼ばれCAEDEに編集状況を伝える関数
34 (defun transcaede-update-hook ()
35  (call-process transcaede-command nil "*transcaede*" nil "fileupdate"
36      (get-filename (current-buffer))
37      (get-file-content (current-buffer))))
```

4.8.2 Emacs と連携するコマンド:transcaede

エディタ連携プログラム transcaede を Emacs と CAEDE の連携のために実装した。transcaede は、与えた引数の情報を CAEDE に伝えるプログラムである。transcaede には 3 つの引数がある。第 1 引数はファイルに対する命令の種類、第 2 引数はファイル名、第 3 引数はファイルの内容を指定する。第 1 引数のファイルに対する命令は、open、editing、close の 3 種類である。open がファイルを開いた、editing がファイルを編集中、close がファイルを閉じたという操作に対応している。例えば、hello.c というファイルを開く場合には、“transcaede open hello.c “ ... (ファイルの内容)... ““として実行する。このように実行することで、CAEDE に hello.c というファイルを開いたということを伝えることができる。

emacs からはファイルに対するイベント 2 種類が発生した際とファイル編集中の定期的なタイミングで呼び出すことを想定している。つまり、下記のタイミングで transcaede を呼び出すことを想定している。

- ファイルを開いた時
- ファイルを閉じた時
- ファイル編集中一定時間ごとに

エディタはこのタイミングで transcaede に適切な引数を与えて実行することで CAEDE と連携して動作するエディタとして利用可能となる。

transcaede は Ruby で約 50 行程度の小さなプログラムで実装されている。これは transcaede は受けとった情報を CAEDE プロトコルにしたがって、CAEDE に送信する簡単なプログラムだからである。CAEDE プロトコルのメッセージが単純であるため送信部は文字列を生成し、送信するだけの単純なプログラムとなっている。また Ruby で実装しているため多くの OS で動作する。このため複数の OS で動作する Emacs では、このプログラムを流用することも可能である。

transcaede は Emacs の Emacs Lisp というユーザプログラムから実行されることを想定している。Emacs Lisp でファイルの開閉時、指定した一定時間ごとに transcaede を実行する。この際に Emacs の開いているファイルの情報などを引数として渡すことで、CAEDE が Emacs から収集した Awareness 情報を送信する。

4.9 CAEDE とバージョン管理システムの連携

バージョン管理システムとして CAEDE では CVS を選択している。CVS との連携は mycvs というラッパプログラムを通じて行なわれる。このバージョン管理システム連携プログラムは 50 行程度の Ruby の小さなプログラムである。

mycvs コマンドは cvs のフロントエンドとなっており、通常のコマンドのかわりに mycvs コマンドを呼ぶことで CAEDE に情報が自動的に送られる。この処理は単純に行なわれており、cvs で add、remove、commit コマンドを実行しようとしたときのみ CAEDE にメッセージを送信し、それ以外の場合にはそのまま cvs コマンドを実行するものである。例えば、リポジトリに fileA を追加する場合には、mycvs add fileA と実行する。このコマンドを実行するだけで、cvs を通じて fileA がリポジトリに追加され、CAEDE にもその情報が伝えられる。

このような実装にすることで、cvs コマンドを利用していた開発者は cvs コマンドの代わりに mycvs コマンドを使うように心がけるだけで CAEDE を利用できる。cvs コマンドを利用して動作するプログラムでも、mycvs コマンドを cvs コマンド

として指定することで、cvs コマンドを利用して動作するプログラムをそのまま利用できる。

第 5 章

システムの評価

ここでは、本研究で作成した共同開発支援システム CAEDE の性能の評価を行なう。Awareness 情報は開発者が今行なっている作業の情報というリアルタイム性の高い情報である。このため、Awareness 情報を遅延なく確実に伝えることが CAEDE には求められる。

しかし、CAEDE ではエディタ連携プログラム、バージョン管理システム連携プログラムの簡略化のために独自プロトコルの CAEDE プロトコルを利用している。この CAEDE プロトコルは単純ではあるが効率良い転送を行なうことは考慮していない。このため、CAEDE を利用した通信で、Awareness 情報を十分な速度で開発者に伝えることができるかを評価する必要がある。

ここでは CAEDE で頻繁に通信を行なった際の遅延について評価を行う。特に CAEDE がインターネットを通じて頻繁に通信を行なっている状況下での遅延時間を評価する。実際に CAEDE を使った共同開発を行なう場合に、Awareness 情報を送信するために十分な速度でメッセージをやり取りできることを示す。

5.1 ネットワーク性能の評価

ここでは CAEDE で頻繁に通信を行ない、その時の CAEDE の通信で発生するネットワークの遅延について調べる。CAEDE サーバでは通信が頻繁に行なわれている場合、メッセージレシーバに大量のメッセージが届く。メッセージレシーバ

はこのメッセージを到着順に処理していくため、大量にメッセージが届くと、メッセージがメッセージディスパッチャに渡るまでの時間が長くなる。そのため大量のメッセージがメッセージレシーバに届くと処理に遅延が発生する。ネットワーク自体の混雑による遅延、メッセージレシーバで発生する遅延の両方を合わせたものが、CAEDE における頻繁な通信における遅延である。この変化を見ることで、頻繁な通信が CAEDE に与える影響を評価する。

手法としては、まず CAEDE サーバに 20kbyte ほどのメッセージを定期的に送ることでネットワークをに混雑している状況にし、メッセージがメッセージレシーバに定期的に届くようにする。その状況で CAEDE に短い IM のメッセージを送信する。IM のメッセージは CAEDE サーバでの計算時間が短い通信である。CAEDE サーバは、IM のメッセージをメッセージディスパッチャでほとんど計算を行なわず、応答メッセージをメッセージセンダに渡して処理する。この IM のメッセージの送受信にかかった時間の変化を調べることでメッセージの処理に時間がかかった時間ではなく、ネットワーク自体の混雑による遅延とメッセージレシーバで発生する遅延を測定することができる。そして、この IM の送受信にかかった時間の変化を測定することで CAEDE の頻繁な通信に対する遅延の測定が可能である。

実験は、ある一定間隔でメッセージが送られてくるという状況で実験を行なう。このメッセージを送信する間隔の範囲は、ここでは 10 秒から 1 秒とした。このメッセージを送る間隔は、CAEDE を利用している開発者がファイルの編集や保存を行なう頻度として考えることができる。つまり、1 秒の間隔でメッセージを送信する場合、1 秒 1 度に誰かがファイルの保存を行なうような場合となる。しかし、一般に開発者がエディタを利用している際、30 秒や 1 分といった間隔で保存は行なう。このため、ここで実験を行なう 1 秒に 1 度という頻度は通常の CAEDE の利用時には発生しない頻度の通信である。

5.2 実験結果

5.2.1 実験環境

ここでは実験を行なった環境について述べる。

ネットワーク環境

実験には実効速度 20Mbits/s のインターネット回線を使用した。このネットワーク帯域を調査するために、iperf というプログラムを利用した。このプログラムは 2 点の間でネットワークの帯域を調査するものである。このプログラムを 10 回実行した平均の値をこのネットワークの帯域とした。

計算機

実験に使った CAEDE サーバは

- Core 2 Duo 2.6GHz の CPU
- 4GB のメモリ
- Marvell 社の 100Mbps の Ethernet ネットワークカード

を搭載している計算機上で動作させた。

CAEDE クライアントの計算機として

- Core 2 Duo 2.6GHz の CPU
- 4GB のメモリ
- Intel 社の 100Mbps の Ethernet ネットワークカード

の計算機上で CAEDE を動作させた。

5.2.2 実験手順

ここでは、行なった実験の手順について解説する。

まず、CAEDE サーバを CAEDE サーバ用の計算機で動作をさせておく。次に、CAEDE サーバが動作している計算機とは違う計算機で CAEDE クライアントを起動する。この CAEDE クライアントが起動している計算機と CAEDE サーバの通信は実効速度 20Mbits/s のインターネット回線を通じて行なわれる。そして、実験用の特殊な CAEDE クライアントを動作させる。この CAEDE クライアントのソースコード 5.1 に示す。この CAEDE クライアントは、CAEDE にエディタがファイルを開いたという FILE_OPEN メッセージを指定した時間ごとに送信するものである。なお、この送信されるメッセージの大きさは 20kbyte 程度である。この CAEDE クライアントを利用して、CAEDE に頻繁にメッセージが送受信される状態にする。

頻繁にメッセージがやりとりされている状態で、別の CAEDE クライアントから図 5.1 で示す短い IM のメッセージを送信する。この短い IM のメッセージは from の項目と to が同じになっている。このため、CAEDE クライアントから送信され、CAEDE サーバが受けとると、CAEDE サーバはメッセージを送信した CAEDE クライアントに同じの内容のメッセージを送り返すという挙動を行なう。この際、コピーなどの処理に時間を取られないため IM 本文を短いものとした。なお、図 5.1 のメッセージの id と time は適当なものである。実験の際はその都度正しい id と time を設定する。このメッセージの送信時刻と受信時刻からネットワークの転送にかかった時間を測定する。測定はメッセージを 50 回ランダムなタイミングで送信し、そのメッセージの送受信にかかった時間の中央値、標準誤差を調べた。この中央値をその頻繁なネットワークにおける、ネットワークの遅延とした。

上述の実験を、図 5.1 のプログラムのメッセージを送る間隔の設定を、10 秒から 1 秒で 1 秒刻みの 10 個の状態について実験し、それぞれの状態における IM の送受信にかかった時間を調べた。

10:20090113201126.729:user:user:SEND_MSG:ping

図 5.1: 実験で送信したメッセージの例

5.2.3 実験結果

図 5.2 が実験結果である。この図は、横軸はメッセージを送る間隔で、0 に近い方がメッセージをより頻繁に送ることを意味している。図では右側にいくほど通信の間隔は短くなっていき、より頻繁にメッセージを送信する状況となる。縦軸は、そのメッセージを送る間隔における、IM メッセージ送受信にかかった時間である。

5.3 考察

実験の結果では、10 秒から 1 秒に 1 度程度のメッセージを送信する状況では、メッセージの送受信にかかる時間は、0.025 秒から 0.03 秒程度で大きく変化は表われなかった。つまりこの程度の頻度の通信では CAEDE プロトコルによる通信は問題なく送受信が可能ということである。0.025 秒から 0.03 秒程度の遅延であれば、Awareness 情報を伝えるには十分な速度である。

この結果は、20Mbits/s のインターネット環境において、CAEDE は、1 秒に 1 度というかなり頻繁にメッセージが送受信されるような状況においても、Awareness 情報を開発者にほとんど遅延なく伝えることが可能であることを示している。CAEDE は CAEDE プロトコルの実装の単純さにも係らず Awareness 情報を送受信するには十分機能することを示している。

ソースコード 5.1: ネットワークに頻繁にデータ送信を行なうプログラム

```
1 # CAEDE用の共通メソッドを読みこむ。
2 # CAEDE::Client などのメソッドが利用できる。
3 require 'CAEDE'
4
```

```
5  # 送信するファイルを読みこむ。
6  $filename = "./test.c"
7  f = open($filename)
8  $content=f.read
9  f.close
10
11 # メッセージを送信する間隔を秒で指定
12 $timing = 0.1
13
14 # テスト用クライアントクラス
15 class TestClient
16   # テスト用クライアントの初期化メソッド
17   def initialize
18     # CAEDE との通信を初期化
19     @client = CAEDE::Client.new("sakura" , "passwd")
20   end
21
22   # テスト用クライアントのメインメソッド
23   def run
24     # CAEDE サーバと通信開始
25     @client.start
26
27     # 指定したタイミングでFILE_OPEN のメッセージを送信しつづける。
28     while true
29       sleep $timing
30       @client.sendMessage("sakura","sakura","FILE_OPEN",$filename+":"+
             $content)
31     end
32   end
33 end
34
35 begin
36   # テスト用クライアントを作成
37   client = TestClient.new
38   # テスト用クライアントを実行
39   client.run
40 rescue Interrupt
41   puts "exiting..."
42   exit
43 end
```

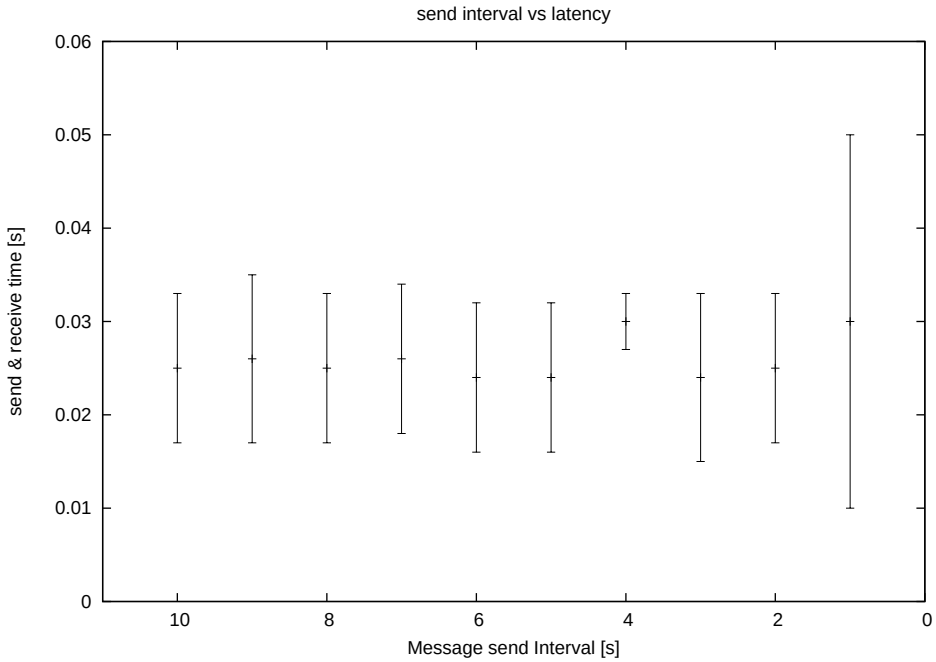

図 5.2: ネットワークの遅延のテスト結果

第 6 章

関連研究

6.1 Awareness

共同作業の支援を計算機で行なう CSCW の分野で、Awareness は提案された。Awareness は、共同作業を行なっている他のメンバの作業の状況把握のことである [3]。Awareness を導入することで、共同作業において個人作業の全体における役割や意義の把握、全体の進捗状況の把握も容易となる。つまり、Awareness によりグループの作業工程の管理が容易になる。また Awareness は作業を行なっている者が自ら情報を取得するものではなくシステムが自動的に通知するものである。本研究はこの Awareness を少人数の共同開発に導入した。また、既存のエディタを利用して Awareness の提供を行なった。

Awareness として利用できる情報のことを Awareness 情報と呼ぶ。この Awareness 情報を Franz らは二つに分類した [7]。Awareness 情報を Social Awareness 情報と Task-oriented Awareness 情報の二種類に分類した。Social Awareness 情報は他のユーザの存在、活動に関する情報である。Task-oriented Awareness では、共有している成果物に対する活動に関する情報である。本研究は、Task-oriented Awareness 情報を既存のエディタ、既存のバージョン管理システムを利用して提供可能にした。

6.2 共同開発支援システム

共同開発支援では Awareness として色々な物が提案されている。T.Schummer[9] は、複数人でソースコードを読んでいる際、そのソースコードに関連がある部分を読んでいる人をユーザに示し、そのユーザとの情報共有を勧めることでソースコードへの理解の速度を向上させるという Awareness 情報を提供している。Carl Cook[2] らは、CAISE という共同開発支援ソフトウェアのためのツールを作成した。CAISE では、現在編集を行なっているユーザおよびその編集しているファイルという Awareness 情報、現在編集を同時に行なっている者が居る場合にはその内容をエディタ、UML エディタの上に表示するという Awareness 情報を提供している。

また、共同開発では競合の発生によって開発の進行が低下するという問題もある。GRAM[10] では、ユーザの状況や編集の状況を確認できる開発環境を構築し、競合の発生を抑えるといった Awareness 情報も提案されている。

これらの Awareness 情報を利用するためには、既存のシステムでは特定のエディタやバージョン管理システムに依存した実装が行なわれている。このため開発者が普段使用している環境から、別の環境に移行する必要がある。その結果、既存のエディタの利用法などのノウハウが利用できなくなるといった問題がある。本研究ではこの問題を解決するため、既存のエディタを利用して、Awareness 情報の収集を実現した。これによりユーザは自分の好きなエディタを共同開発で利用可能となっている。

Gerakdline[4] らは、CVS のコミットログを通知することで、コミットログを Awareness として利用可能である事を示し、その効果を示した。本研究もこの考えに基づき、ソースコード管理システムを利用する Awareness の提供を行なっている。共同開発でも Awareness 情報を使うことで作業を円滑に進める事が可能である。

第 7 章

結論

7.1 まとめ

本研究では少人数の共同開発においてこれまでの支援システムの問題点である、開発者が好みのエディタ、バージョン管理システムを選択できない問題の解決を図った。

本研究では既存のエディタ、バージョン管理システムから取得でき、少人数の共同開発でも重要となる Awareness 情報選別を行なった。さらに、既存のエディタ、バージョン管理システムから少人数の共同開発においても重要な Awareness 情報を収集可能にする、エディタ連携プログラム、バージョン管理システム連携プログラムを提案した。

本研究では既存のエディタ、ソースコード管理システムを利用することが可能な少人数の共同開発支援システム CAEDE を実装した。CAEDE を利用することで、共同開発において開発者は既存のエディタや既存のバージョン管理システムからの移行作業を行なう必要がなくなる。これにより、共同開発支援システムの導入のコストを下げることができる。

本研究で実装した CAEDE は Emacs、CVS と連携して動作する共同開発システムである。エディタ連携プログラム、バージョン管理システム連携プログラムの、transcaede、mycvs を利用して Awareness 情報を収集する。このエディタ連携プログラム、バージョン管理システム連携プログラムの作成は、CAEDE で採用されて

いるプロトコルである CAEDE プロトコルが単純なものであるため、簡単に実装することができた。これらのプログラムはそれぞれ 50 行程度の小さな Ruby のプログラムで実装できた。また、CAEDE を利用するために必要な Emacs の設定も短いもので済んだ。

CAEDE に現在対応していないエディタであっても、外部プログラムを呼び出す機能と CAEDE と連携するための要件を満たすエディタであればエディタ連携プログラムを作成することで対応できる。バージョン管理システムも同様にラッパプログラムのバージョン管理システム連携プログラムを新たに作成することで対応することができる。CAEDE プロトコルが単純なプロトコルであるため、エディタ連携プログラム、バージョン管理システム連携プログラムの実装は容易に行なる。このため新しいエディタ、バージョン管理システムに対応する、エディタ連携プログラム、バージョン管理システム連携プログラムを作成することも容易である。

また、Awareness 情報はリアルタイム性が高い情報であるため、その送信にかかる時間というものじゃ重要である。そこで、本研究では CAEDE のインターネットを通じて利用する場合の遅延を調査した。その結果、CAEDE は 1 秒に 1 度メッセージが届くような頻繁な通信状況でも十分に Awareness 情報を提供できることを示した。

7.2 今後の課題

今回実装した CAEDE は Emacs、CVS という環境に対応している共同開発支援システムである。今後は実際に他のエディタ、Vim、Eclipse 付属エディタ、VisualStudio 付属エディタといったエディタに対応しやすいように、エディタ連携プログラムを準備していく予定である。

また、ネットワーク帯域が十分でない環境での評価を行なう予定である。今回の評価では通信が頻繁に行なわれている状態での評価は行なったが、ネットワーク帯

域が十分でない環境での評価は行なっていない。例えば、携帯電話などのモバイル環境で開発の状況をチェックするといった用途にもCAEDEを利用することが可能にするためには、ネットワーク帯域が十分でない環境でのテストは重要である。

実際、共同開発で長期に渡り利用してもらいCAEDEの評価を行なうことも重要な課題である。この評価を行なうことでCAEDEが実際に導入が容易であるかどうか、共同開発のAwarenessの実現に有益なのか、長期的に開発者の利用に耐えられるものなのかといった評価を行なうことができる。

謝辞

本研究を遂行するにあたっては、いろいろな方々にお世話になりました。

まず、指導教員の多田好克先生には日頃から熱心なご指導、そしてご鞭撻を賜わりました。また、ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。

また、佐藤喬先生には、いつも自分の考えが至らず迷惑ばかりかけていたにも係らず熱心にご指導を頂きました。感謝しております。

本研究が完成したのは研究方針や方法論について議論をし、共に研究生活をおくってきた多田研の皆様のおかげでもあります。

違う研究室にもかかわらず研究に関する議論に忙しいときでも応じてくれた友人の酒井慎平氏。自分の研究の動機となった、普段共同開発を行なっている寺田遼平氏。経済的な援助をして下さった両親。

最後に、これらの皆さんに感謝いたします。

参考文献

[1] J. J. Cadiz, Gina Venolia, Gavin Jancke, and Anoop Gupta. Designing and deploying an information awareness interface. In *CSCW '02: Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pp. 314–323, New York, NY, USA, 2002. ACM.

[2] Carl Cook and Neville Churcher. Constructing real-time collaborative software engineering tools using caise, an architecture for supporting tool development. In *ACSC '06: Proceedings of the 29th Australasian Computer Science Conference*, pp. 267–276, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[3] Paul Dourish and Victoria Bellotti. Awareness and coordination in shared workspaces. In *CSCW '92: Proceedings of the 1992 ACM conference on Computer-supported cooperative work*, pp. 107–114, New York, NY, USA, 1992. ACM.

[4] Geraldine Fitzpatrick, Paul Marshall, and Anthony Phillips. Cvs integration with notification and chat: lightweight software team collaboration. In *CSCW '06: Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pp. 49–58, New York, NY, USA, 2006. ACM.

[5] T. Kawashima and J. Ma. Tomscop - a synchronous p2p collaboration platform over jxta. *Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on*, pp. 85–90, March 2004.

[6] M.A.S. Mangan, M.R.S. Borges, and C.M.L. Werner. A middleware to increase awareness in distributed software development workspaces. *WebMedia*

and LA-Web, 2004. Proceedings, pp. 62–64, 2004.

[7] Wolfgang Prinz. Nessie: an awareness environment for cooperative settings. In *ECSCW'99: Proceedings of the sixth conference on European Conference on Computer Supported Cooperative Work*, pp. 391–410, Norwell, MA, USA, 1999. Kluwer Academic Publishers.

[8] Christian Schuckmann, Lutz Kirchner, Jan Schümmer, and Jörg M. Haake. Designing object-oriented synchronous groupware with coast. In *CSCW '96: Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pp. 30–38, New York, NY, USA, 1996. ACM.

[9] T. Schummer. Lost and found in software space. *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*, pp. 1–10, Jan. 2001.

[10] Katsuhiro Takata and Jianhua Ma. Gram - a p2p system of group revision assistance management. In *AINA '04: Proceedings of the 18th International Conference on Advanced Information Networking and Applications*, pp. 587–593, Washington, DC, USA, 2004. IEEE Computer Society.