ソフトウェア工学 129－6
（2000．11．16）

ソフトウェアを複数のプログラマが協調して開発するための環境の構築

宮脇冨士夫　渡邉勝正

奈良先端科学技術大学院大学 情報科学研究科

インターネットの発展にともなって、複数のプログラマが分散的な環境で一つのソフトウェアを協力して開発することが可能になった。そこでは、設計とプログラミングを明確に分離することなく、プログラムを公開することによって、逐次、更新が繰り返されてソフトウェアが発展・成長していく方式が注目されている。いわゆるオープン・ソフト・システムである。この方式ではプログラマの協調性、独創性、積極性をいかにして支援するかということが課題となる。 本研究では、1つの方法として、プログラマの貢献度を評価する仕組みを導入した。また、お互いのプログラムの理解を助けるために、日本語でプログラムを記述するツールを導入した。本論文では前者に重点をおいて述べる。

An Environment to develop a Software by Decentralized Collaborating Programmers

Miyawaki Fujio　Watanabe Katsumasa

Graduate School of Information Science,
Nara Institute of Science and Technology

The growth of Internet technologies makes it possible to develop a software by decentralized programmers. In there, the traditional designers and programmers are not independently exist, all programs are made open and they are improved by many programmers. It is a so-called open software system. The important problem in the system is how to activate cooperativeness, originality, and positivity of the collaborating programmers. We take notice of evaluating contribution of each programmer to make intensive to those factors, and introduce a programming language in Japanese to make it easy to understand programs in each other. This paper describes mainly the former.

1. はじめに

一つのソフトウェアを何人かのプログラマが協力して開発することは幾多のところで実践されており、ウォーターフォール型とかプロトタイプ型などの開発方式もそれなりの認知を得ている[1]。しかし、ソフトウェアの開発にあたって、設計とプログラミングを明確に区別することなく、お互いのプログラムを見せ合いながら設計とプログラミングを螺旋的にくりかえしながら進行することがある。いわゆるオープン・ソフト・システムである[2]。本研究の目指すところもそこに焦点をしぼり、数人のプログラマが協調して、数万行のプログラムを開発することを想定している。オープン・ソフト・システムの管理ツールとしてはRCS(Revision Control System)、CVS(Concurrent Versions System)などが知られており、

NetBSD、LINUX の開発に活用されている[3]。われわれが目標とする環境はお互いのプログラムの参照と編集の自由度を大きくしてプログラマの独創性を重視するところに特徴がある。

2. 目標とプロジェクトモデル

われわれが目標とするところは、一つのソフトウェアを開発するにあたって、メンバーの独創性、協調性、積極性が鼓舞される環境を構築することである。

対象とするプロジェクトモデルの特徴を列記すると、下記のようになる。

1. 複数のメンバーが一つのソフトウェアを設計・開発する。
   1. 中心人物(プロジェクトリーダ)が存在する。
   2. 他のメンバーはリーダに協力する。
   3. 最終的にはプロジェクトリーダがプログラムをまとめる。
2. 各メンバーの使用する計算機環境は異なり、インターネットを通じて交信する。
3. プロジェクトの進行過程。
   1. プロジェクトリーダが目標を設定し、説明をする。
   2. プロジェクトの内容を理解できるメンバーが参画する。
   3. 開発プログラムについて、各メンバーの理解を一致させることは容易ではない。したがって、各メンバーは自分の理解に基づいて構想をたて、目標のプログラムを設計・記述する。
      リーダは全体の進行を見守り、収束する方向に導く。
   4. 各メンバーのプログラムは公開し、相互の参照・コピーは自由である。
      このことによって、共通の理解を図り、プロジェクトを収束の方向に導くとともに、プログラムの品質が向上することを期待する。
   5. 部分的な参画も可能である。自分の関与する以外の部分はコピーする。

目標を同じくして各メンバーが独立的に同じ仕様に対するプログラムを開発するところがこのプロジェクトモデルの特徴である。 作業の結果として、メンバーの数だけプログラムが出来上がる。

3. 開発環境のモデル

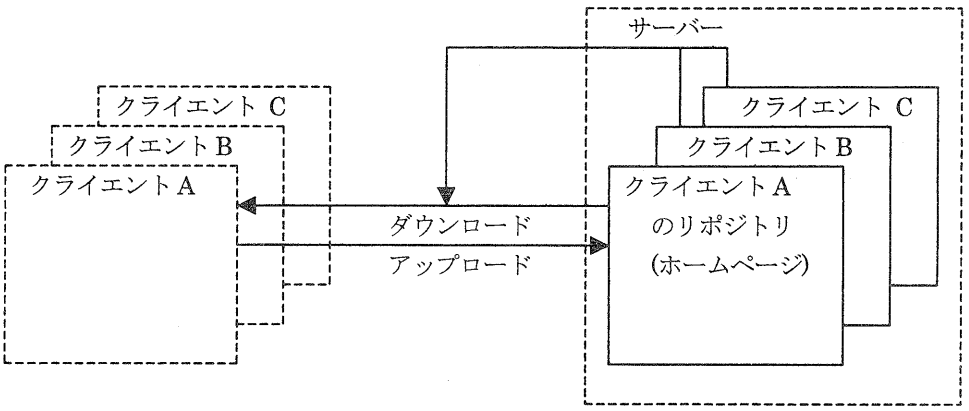

図 1. 環境の概念図

開発環境(通称:nara.cowork)はクライエント／サーバー型で、一つのサーバーに各メンバーがクライエントとして、インターネットで接続されている。サーバー上にはメンバー毎にリポジトリ(ホームペー

ジ)が割り当てられており、プログラムその他の情報を公開する。各メンバーは他のメンバーのリポジトリを閲覧することは自由である。しかし、サーバー側へのアップロードは自分のリポジトリに限られている。したがって、決して他のメンバーの作業を犯すことはないし、自分の作業が犯されることもない。

メンバーはサーバーにあるホームページに案内されて目的のリポジトリにアクセスし、ファイル単位で自由に情報を取り込んで、自分のクライアント計算機環境で作業する。環境のモデルを概念的に示すと図1のようになる。

4. 開発環境の備えるべき要件

メンバーが独創性、協調性、積極性を発揮することを期待するためには、一つはプロジェクトに対するメンバー個人の貢献度を評価する仕組みが重要であり、もう一つは、お互いのプログラムの理解を容易にし、共通の理解に至るための道具立てが必要である。

nara.cowork は個人の貢献度を評価するために次の仕組みを備えている。

1. プログラムファイルのバージョンを管理する。プログラムの記述履歴が残っている。
2. 他のプログラムを参照すれば、ファイル名、バージョン番号、参照日時を記録する。
3. プログラムは行毎および要素(6.3 参照)毎に記述者(出典ファイル名)を明らかにできる。
4. バージョンをたどれば行毎に記述の変遷を調べることができる。

また、プロググラムの理解を助けるために、つぎに述べるような日本語でプログラムを記述するツールを備えている。

5. 日本語によるプログラムの記述と処理

われわれの環境における日本語プログラムの記述と処理を図2.に示す。

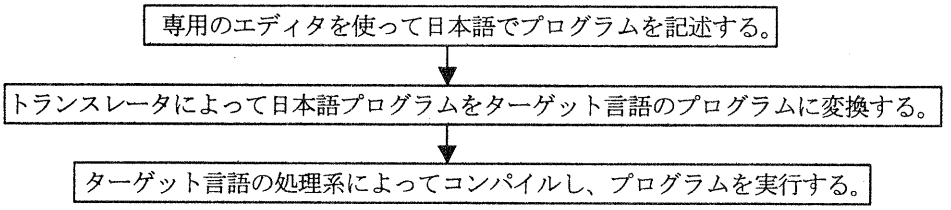

図2. 日本語プログラムの記述と処理

ターゲット言語としては現在のところ C++と JAVA を実装している。 また、前節で述べたバージョンの管理と個人の貢献度を評価するためのデータは全て専用のエディタが処理している。

6. 専用エディタによるファイル管理について

ファイルにはバージョン管理ファイルとプログラムファイルがある。バージョン管理ファイルにはバージョンの更新情報が時系列的に蓄積されている。プログラムファイルはバージョン管理ファイルから特定のバージョンを抜き出したものである。図2.のトランスレータが処理するのはこのプログラムファイルである。

プログラマはバージョン管理ファイルにプログラムを記述する。バージョン管理フアイルの上でプログラムの記述を支援すること、バージョン管理ファイルから特定のバージョンを抜き出してプログラム

ファイルにすることがエディタの機能に含まれている。

6.1　バージョン管理ファイルのバージョンの概念

ここでいうバージョンは論理的な区切りを意味するものではなく、作業の時系列的な区切りである。すなわち、作業の切れ目がバージョンの区切りである。したがって、作業を中断すれば、その度にバージョン番号が更新される。しかし、このようにするとバージョン更新の頻度が多くなりすぎることもあるので、ファイルの更新は単なる更新と継続更新を区別している。単なる更新は作業を終了すれば新たなバージョン番号に更新されるが、継続更新は先の更新が継続され、作業の中断があってもバージョン番号は更新されない。

6.2　バージョン管理フアイルのデータ構造

バージョン管理ファイルは4.で述べた評価のための要件を満足しなければならない。大きくは図3に示すようにバージョン部分が時系列的に蓄積されている。

バージョン部分は更新された部分のみのデータが格納されている。したがって、所望のバージョンのプログラムファイルを取り出すためにはそれ以前のバージョンが必要である。

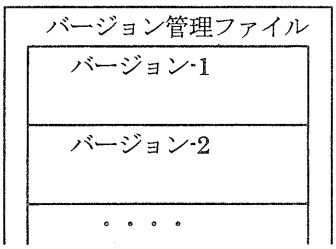

図3.バージョン管理ファイルの構造

6.3 バージョン部分のデータ構造

各バージョン部分は次の三つの部分からなっている。

a. バージョン番号行　　1行

b. 文行　　更新された行毎に1行

c. 参照記録行　　他のファイルを参照する毎に1行

a.　バージョン番号行は図4に示すように4つのフィールドからなっている。

ファイル名	バージョン番号	作成時刻	公開マーク

図4　バージョン番号行のフィールド

各フィールドの意味は以下に説明する通りである。

a1 ファイル名　　ＯＳが管理するファイル名。全てのバージョン部分に同じ名前が付いている。バージョン部分の区切りの目印である。

a2 バージョン番号　時系列に対応する順序番号である。

a3 作成時刻　　作業の区切り時刻(年月日時分秒)である。

a4 公開マーク　　他者に公開するか否かのマークである。

b.　文行は図5.に示すように5つのフィールドからなっている。

行番号	参照バージョン番号	旧行番号	更新有無	文の要素　・・・

図5.　文行のフィールド

各フィールドの意味は以下に説明する通りである。 なお、文行の詳細は6.4で説明する。

b1 行番号　　該当バージョン上の行番号である。
最後の行は-(マイナス)記号を付けて目印としている。

b2 参照バージョン番号　後に述べる参照記録行を特定するための順序番号である。

b3 旧行番号　　参照バージョン中の行番号である。

b4 更新の有無　　更新が行われた行は(1), 行われなかった行は(0)のマークを付けている。

b5 文の要素　　文の要素には次の7つの要素がある。それぞれ先頭にタグマーク(@)が付ている。

識別名	@n 文字列	文字列に@マークを含む場合は@@として出力される。
定数	@t 文字列	〃
演算子	@k 文字列	〃
コメント	@c 文字列	〃
空白	@sx	(x: 空白の数)
参照マーク	@pi	(i: 前述(b2)の参照バージョン番号と同じ)
行の終わり	@r	

c. 参照記録行は図6.に示すように3つのフィールドからなっている。

参照時刻	ファイル名	バージョン番号

図6. 参照記録行のフィールド

各フィールドの意味は以下に説明する通りである。

c1 参照時刻　　他のファイルを参照した時刻(年月日時分秒)である。最後の行は(*)マークを付けて目印としている。

c2 ファイル名　　被参照フアイルのフルパス名である。

c3 バージョン番号　被参照バージョン番号である。

* 参照記録行の第1行目は被更新バージョンのバージョン番号である。ただし、バージョン1は被更新バージョンに該当するものが無いのでダミー行となる。

6.4 文行の詳細

文行はプログラマの記述したプログラムに対応する部分である。4.で述べた要件を満足するのに必要十分なデータ構造になっている。 各フィールドについて追加説明する。

b1 行番号

バージョン部分は更新に関係する部分だけであるから、行を特定するために行番号が必要である。

b2 参照バージョン番号

その行が他のファイルからコピーしたものか否かを記録するためのデータである。その値がiであれば、参照記録行のi行目をみれば被参照バージョン番号が明らかになる。なお、このデータが0の行

は新たに記述された行である。

b3　旧行番号

被参照バージョン中の行番号である。行の履歴をたどるために必要である。また、エディタは内部に被更新バージョンを再現するときに、旧行番号を手がかりにしてバージョン管理ファイルのバージョン部分をさかのぼって該当行に到達する。

b4　更新の有無

他バージョンからコピーをした後、変更されたか否かを区別している。被更新バージョンからコピーして、変更の無い場合は、次の文の要素を省略する。

b5　文の要素

前述の7つの要素のうち識別名、定数、演算子、コメントがプログラマの意識するプログラム要素であり、後の3つは補助の要素である。

このエディタでは入力の時点で要素ごとに区別して入力する方法を採用しており、画面の上には現れないが、内部的にはこのように区別されている。

これらの要素のうち特に説明を要するのは参照マーク(@pi)である。これは識別名、定数、コメントの各要素の後に付いてその要素がどのバージョンからコピーされたものか追跡できるようになっている。

7. プログラムファイルのデータ構造

プログラムファイルはバージョン管理ファイルから特定のバージョンについて全文を復元して取り出したものであるが、要素ごとに区切られていて、その要素は先に文の要素で説明したものと同じである。ただし、参照マークは不要であるので取り除いてある。

このように、入力の段階で要素の分離がされているので、後のトランスレータの部分で字句解析が不要となる。

8. バージョン管理ファイルの例

図7に日本語プログラムの例を示す。

```
10 整数 新規ﾌｧｲﾙ?(文字* 目的ﾌｧｲﾙ名)
20 {文字 既存ﾌｧｲﾙ名[80]。文字 *ｶｰｿﾙ_r、ｶｰｿﾙ_t。
30  (ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ=fopen(ﾌｧｲﾙ名集積ﾌｧｲﾙ、"r")==NULL)ならば
40      戻値 1 で 呼出点へ戻る。
50  (fscanf(ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ、"%s"、既存ﾌｧｲﾙ名)!=EOF)ならば
60  次の文を繰り返す。
70  「繰返条件(ｶｰｿﾙ_r=既存ﾌｧｲﾙ名, ｶｰｿﾙ_t=目的ﾌｧｲﾙ名。
80              *ｶｰｿﾙ_t!=0 && *ｶｰｿﾙ_r!=0。ｶｰｿﾙ_t++, ｶｰｿﾙ_r++)で
90              次の文を繰り返す。(*ｶｰｿﾙ_t != *ｶｰｿﾙ_r)ならば 繰返終了。
100  (*ｶｰｿﾙ_t == *ｶｰｿﾙ_r)ならば
110  「fclose(ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ)。戻値 0 で呼出点へ戻る。」」
120  fclose(ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ)。
130  戻値 1 で呼出点へ戻る。}
```

図7　日本語プログラムの例

このプログラムを入力するにあたって、10 行から 110 行までをバージョン-1 として入力し、120 行から 130 行を追加してバージョン-2 とした。また、バージョン-1 を入力する際に、50 行から 60 行を入力していなかったので、バージョン-2 を入力する際にそれらも入力した。

この結果のバージョン管理ファイルを図 8 に示す。紙面の都合で本来 1 行の文行が 2 行になっている部分もある。

```
sample1.ver 1 Thu_Sep_21_11:44:13_2000 0
10 0 0 1 @k 整数@s1@n 新規ﾌｧｲﾙ?@p0@k(@k 文字@k*@s1@n 目的ﾌｧｲﾙ名@p0@k)@r
20 0 0 1 @k{@k 文字@s1@n 既存ﾌｧｲﾙ名@p0@k[@t80@p0@k]@k。@s1@k 文字@s1@k*@n ｶｰｿﾙ_r@p0@k、
        @s1@n ｶｰｿﾙ_t@p0@k。@r
30 0 0 1 @s1@k(@n ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ@p0@k=@nfopen@p0@k(@n ﾌｧｲﾙ名集積ﾌｧｲﾙ@p0
        @k、@s1@t"r"@p0@k)@k==@nNULL@p0@k)@k ならば@r
40 0 0 1 @s5@k 戻値@s1@t1@p0@s1@k で@s1@k 呼出点へ戻る@k。@r
50 0 0 1 @s1@k「@k 繰返条件@k(@n ｶｰｿﾙ_r@k=@n 既存ﾌｧｲﾙ名@k,@s1@n ｶｰｿﾙ_t@k=@n 目的ﾌｧｲﾙ名@k。
        @r
60 0 0 1 @s11@k*@n ｶｰｿﾙ_t@k!=@t0@p0@s1@k&&@s1@k*@n ｶｰｿﾙ_r@k!=@t0@k。
        @s1@n ｶｰｿﾙ_t@k++@k,@s1@n ｶｰｿﾙ_r@k++@k)@k で@r
70 0 0 1 @s11@k 次の文を繰り返す@k。@s1@k(@k*@n ｶｰｿﾙ_t@s1@k!=@s1@k*@n ｶｰｿﾙ_r@k)@k ならば
        @s1@k 繰返終了@k。@r
80 0 0 1 @s2@k(@k*@n ｶｰｿﾙ_t@s1@k==@s1@k*@n ｶｰｿﾙ_r@k)@k ならば@r
-90 0 0 1 @s2@k「@nfclose@p0@k(@n ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ@k)@k。@s1@k 戻値@s1@t0@s1@k で@k
        呼出点へ戻る@k。@k」@k」@r
*Thu_Sep_21_11:07:19_2000 sample1.ver 0
sample1.ver 2 Thu_Sep_21_11:52:59_2000 0
40 1 40 0
50 0 0 1 @s1@k(@nfscanf@p0@k(@n ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ@k、@s1@t"%s"@p0@k、@s1
        @n 既存ﾌｧｲﾙ名@k)@k!=@nEOF@p0@k)@k ならば@r
60 0 0 1 @s1@k 次の文を繰り返す@k。@r
110 1 90 0
120 0 0 1 @s1@nfclose@k(@n ﾌｧｲﾙ名集積ﾌｧｲﾙﾊﾝﾄﾞﾗ@k)@k。@r
-130 0 0 1 @s1@k 戻値@s1@t1@s1@k で@k 呼出点へ戻る@k。@k}@r
*Thu_Sep_21_11:44:13_2000 sample1.ver 1
```

図 8　バージョン管理ファイルの例

文行について説明を補足する。

@pi は最初に現れた要素にのみ付けてそれ以降の同じ要素には付けていない。また、被更新バージョンに含まれていた要素には付けていない。したがって、@p1 は現れない。いずれもバージョン管理ファイルが大きくなるのを避けるためである。

最後に nara.cowork を総括的に図示すると、図 9 のようになる。

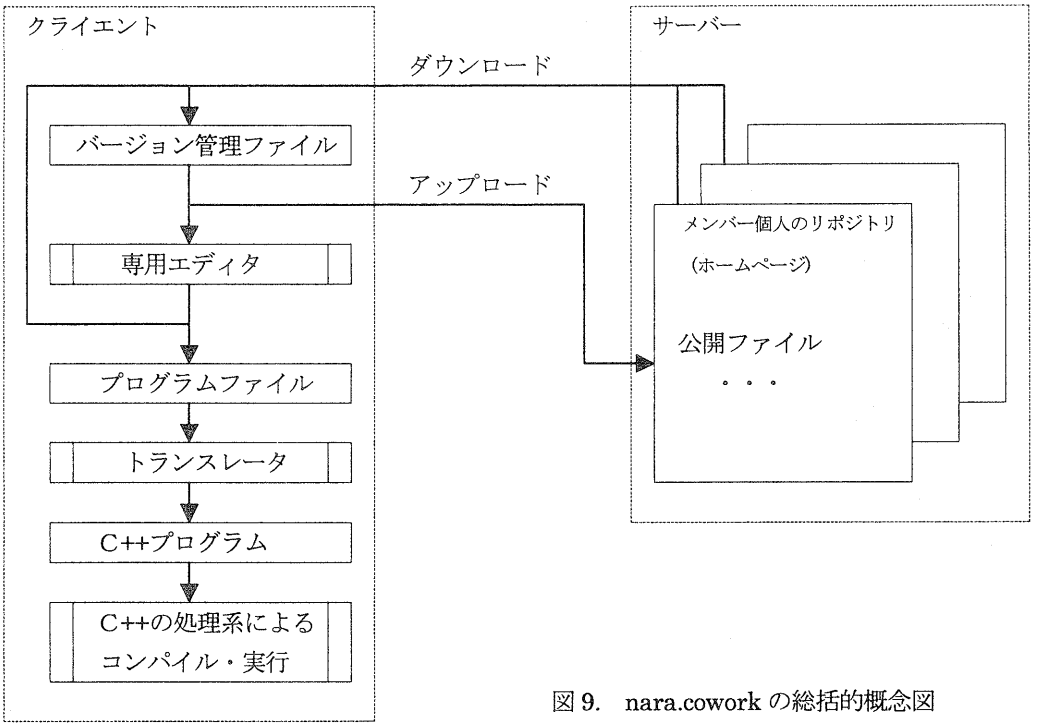

図 9. nara.cowork の総括的概念図

9. おわりに

われわれはソフトウェアの共同開発環境において参画者の協調性、独創性、積極性を支援することの重要性に着目し、その方法として参画者の貢献度を評価することと、参画者相互の理解を容易にするための手段として日本語によるプログラミングを導入した。本報告では前者の仕組みを重点的に述べた。

参画者の貢献度を評価する尺度は多様であるが、誰がどの部分を記述したかということを明かにすることが基礎になる。今後は nara.cowork を実際に使用してみる中で、我々の着目の有効性とデータ構造の必要十分性を検証したい。

謝辞

本研究に関して協力して下さっている研究室の皆さんに感謝します。なお、本研究は一部文部省科学研究費補助金　基礎研究(B)(2)11480068 によるものです。

参考文献

[1] 青山 幹雄 ：ソフトウェア技術者のグローバルスタンダード化，情報処理，Vol.39, No.11, pp.1144-1147, (1998)

[2] John Edwards 翻訳：青山 幹雄：フリーウェアの台頭と変貌, 情報処理, Vol.40, No.1, pp.32-35, (1999)

[3] 力武健次：分散環境でのソフト開発手法 共同作業をツールで支援，日経コンピュータ(No.495), pp.150-153,(May 8 2000).