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.
SS97-86 (1998-03)

オブジェクト指向モデルMonoProcessを用いたソフトウェア開発管理システムにおけるユーザインターフェース部

大下 誠　永山 裕之　山本 哲男　松下 誠　楠本 真二　井上 克郎

大阪大学大学院基礎工学研究科情報数理系専攻
〒560-8531 大阪府豊中市待兼山町 1-3
Phone: 06-850-6571　Fax:06-850-6574
Email: {oshita,nagayama,t-yamamt,matusita,kusumoto,inoue}@ics.es.osaka-u.ac.jp

あらまし　従来のプロセス記述言語やそれに基づくソフトウェアプロセス環境では，プロダクトを生成する手順に着目した記述や環境の構築を行なっていた．しかし，ソフトウェアの再利用やコンポーネントベースの開発といった最近のソフトウェア開発環境は，作業を行なう際にプロダクト等の作るべき対象に着目した環境となっている．このような新しい開発環境に対して従来のプロセス環境が十分に支援できているとは言えない．本稿では，我々が既に提案を行なっているオブジェクト指向モデル MonoProcess を用いたソフトウェア開発管理システム MonoProcess/SME の概要とその試作について，システムのユーザインタフェース部を中心に述べる．また，本システムをソフトウェアプロセスの共通例題に適用した例について述べ，本システムが作業者の開発支援を行なうとともに，進捗状況の把握を行えることを示す．

キーワード: ソフトウェアプロセス，プロセス中心型開発環境，プロジェクト管理，ユーザインタフェース

User-Interface of Software Development Management System with MonoProcess Object-Centered Software Process Modeling

Makoto Oshita　Hiroyuki Nagayama　Tetsuo Yamamoto
Makoto Matsushita　Shinji Kusumoto　Katsuro Inoue

Department of Informatics and Mathematical Science,
Graduate School of Engineering Science, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
Phone: 06-850-6571　Fax:06-850-6574
Email: {oshita,nagayama,t-yamamt,matusita,kusumoto,inoue}@ics.es.osaka-u.ac.jp

Abstract　Most of process-centered software engineering environments and those languages focus on how to produce the product. However, recent software development tend to require to focus on what should be made, because of emergence of various types of software developments; e.g., software reuse, component-based composition, and so on. To achieve this, we propose in this paper a new development environment named MonoProcess/SME, which is based on an object-centered software process model MonoProcess we already proposed. MonoProcess/SME is an software development environment for project management and development support, using the idea of MonoProcess modeling. We have also enacted ISPW-6 software process example with this system. Our system provides an environment for software process execution and process management.

Keywords: Software Process, Process-Centerd Software Development Environment, Project Managemnt, User-Interface

1 まえがき

ソフトウェア開発作業を効率よく行ない，かつ，高品質のソフトウェアを作成するために，ソフトウェアの開発プロセスをあらかじめ記述し[3]，それに基づいた開発作業を行なうようになってきている[5, 11]．プロセスの記述を行なう際にはプロセス記述言語が用いられるが，既存の多くの言語は「ソフトウェアを作る際の作業手順」に着目した物であり，[2, 6, 7, 12]それらに基づいた環境もまた開発作業に着目した物となっている[1, 4, 13]．しかし，オブジェクト指向プログラミングやソフトウェアの再利用，コンポーネントに基づいたプログラム開発等に代表される，近年さかんに行なわれている開発形態は「ソフトウェア開発の際に作られるべき生成物」に着目して作業を整理している．このような生成物に着目している開発形態を考慮したプロセス記述言語を用いてプロセスを記述することによって，よりわかりやすいプロセスの記述を行なうことができると考えられる．

これらの問題に対して，我々は既にオブジェクトモデル MonoProcess の提案を行なっている[9, 10]．MonoProcess は開発環境を直接表現することによって，ソフトウェアプロセスを表現，管理，進化させるための枠組を提供することを目的としている．本稿では，本モデルに基づいて設計された MonoProcess/SME の試作について述べる．本システムはサーバクライアント型のシステムであり，利用者のためのユーザインタフェースと，プロセスの記述を管理するオブジェクトベースから構成される．このうち本敲では，ユーザインタフェース部を中心に述べる．本ユーザインタフェースは，ソフトウェア開発の管理者がプロセス記述を行なう場合や，進捗管理を行なう上で必要となる情報を収集，表示する際に用いられる．また，ソフトウェアの開発者に対しては，オブジェクトベースにて管理されている情報を用いて開発作業を行なう際にも用いられる．

本稿ではまた，試作したシステムをソフトウェアプロセスの共通例題へ適用した例について述べ，本システムを用いて開発作業の支援や進捗状況の把握がどのように行なわれるかを示す．

以下2節では，我々が提案している MonoProcess プロセスモデルについて説明を行なう．3節では，本モデルに基づいて設計されたソフトウェア開発管理システム MonoProcess/SME の試作を行なった内容について述べる．4 節では本システムを共通例題へ適用した例について述べる．最後に5節で本敲の内容をまとめ，今後の課題について述べる．

2 オブジェクト指向モデル MonoProcess

本節では，我々が提案しているプロセス記述言語である MonoProcess について説明する．MonoProcess では，開発環境中に存在する生成物や資源を表現する「オブジェクト」を単位としたプロセスの記述を行う．以下では MonoProcess におけるオブジェクトの定義について述べ，MonoProcess によってソフトウェアプロセスがどう表現されるか説明する．

2.1 オブジェクトの定義

オブジェクトはソフトウェア開発環境におけるさまざまな生成物や資源を表現する．図 1 は MonoProcess オブジェクトの記述例である．この記述は，.Doc.Design と名付けられたあるデザインドキュメントを記述している．

```
Object .Doc.Design def
    Attribute @Owner .Person.Matsushita;
    Attribute @Type "Design Document";
    Attribute @Input (.Doc.Specification
                              .Doc.Schedule);
    Attribute @Localtion .ShareDisk.Document
    Method &Edit def
        $editor = .caller&GetEditor(@Type);
        if ($editor) {
            &View;
            invoke($editor,
                   @Localtion . "design.doc");
        }
    endMethod
    Method &View def
        $viewer = .caller&GetViewer();
        if ($viewer) {
            invoke($viewer, @Input);
        }
    endMethod
endObject
```

図 1: Object の例

オブジェクトの名前(ラベル)は "." (ドット)に続く1つの単語で始まり，必要に応じて複数の「ドット+単語」を繋げることによって構成される．オブジェクトは属性とメソドから構成される．属性はオブジェクトの特性を表現し，メソドはオブジェクトに適用される関数である．

属性にはどのような種類かを示すためにラベルが一意に付けられている．図1 では 4 つの属性が定義されており，@Owner, @Type, @Input そして @Location がそれぞれの属性に付けられている属性ラベルである．属性値としては次のような種類がある．

- 整数/実数 (例: 1, -3.5)

- 文字列 (例: "ABC", "あいうえお")
- オブジェクトラベル (例 .A, .X.Y.Z)
- 上記 3 つの型を要素として持つリスト

例えば図1 で示されるデザインドキュメントを示すオブジェクト中で定義される属性 @Owner の値 .Person.Matsushita は「このオブジェクトの管理者は .Person.Matsushita として表現されるオブジェクトである」ことを意味する.

属性と同様, メソドに対しても, その操作内容を示すためにラベルが一意に付けられている. 図 1 では 2 つのメソドが定義されており, &Edit や &View がメソドに付けられたメソドラベルであり, それぞれ, 編集と閲覧の作業を表わしている.

メソドによって表現される作業の内容は, オブジェクトの集合内で閉じた演算であるメソド関数として記述される. メソド関数によって行なわれる操作には, 次のものがある.

- オブジェクトの保持する情報への操作: 属性値の参照, 属性値の変更, 属性/メソド一覧の入手, メソドの実行.
- その他のオブジェクトに関する操作: 新規オブジェクトの生成, 外部ツールの起動.
- 通常のプログラミング言語に見られる操作: 整数/実数の演算, 文字列演算, 集合演算.

また, MonoProcess ではメソド関数の機能のみを定義しており, その文法等は定義しないため, 複数のメソド記述言語を用いることができる. 図1 では, Perl 風の簡単な記述言語を用いてメソド内の記述を行なっている.

2.2 ソフトウェア開発プロセス

部分オブジェクト Op を, あるオブジェクト O に対して定義する. Op は O の持つラベルを接頭語として持ち, O の持つ属性やメソドの部分集合を持つものとする. Op は対象となるオブジェクト O の部分情報を持っており, ある観点において興味のある特徴を抽出したものである.

このようにして定義された部分オブジェクトを用いて, 状態オブジェクト Os を定義する. Os は, ソフトウェア開発環境におけるある状態を表現し, 部分オブジェクトの集合とする. 我々は, 開発環境におけるある状態は, 開発環境中に存在する生成物や資源によって表現できると考えている. MonoProcess では, ソフトウェア開発プロセスをこれら状態オブジェクトの遷移系列として定義する.

簡単な例として .SPEC, .CODE, .TEST という 3 つのオブジェクト (それぞれ, 仕様書, ソースコード, テスト結果, とする) が存在する状況を仮定する. 各オブジェクトには同一の属性ラベル @FINISHED があり, これによって, それぞれの生成物に対する作業が完了しているかどうかを表現しているとする.

この時, 開発の進行状況を示す .SAMPLEOBJ と名付けられた状態オブジェクトを, .SPEC, .CODE, .TEST それぞれのオブジェクトから @FINISHED だけを抜きだした部分オブジェクト 3 つの集合として定義する. 作業が進行するにつれて, .SAMPLEOBJ は次のように状態を変化させる.

(1) まず, 開発開始時には .SAMPLEOBJ の持つ 3 つの状態オブジェクトの属性が全て "false" になっている状態になっている.

(2) 仕様書の作成が終了した時点で, .SPEC の持つ属性 @FINISHED が "true" へ変化する.

(3) さらにコーディングの作業が終了すると, .CODE の持つ属性が @FINISHED "true" へと変化する. つまり, .TEST だけが "false" の状態になる.

(4) 作業が終了して, 全てが "true" に変化した状態になる.

このような .SAMPLEOBJ の遷移系列を, MonoProcess におけるソフトウェア開発プロセスと考える.

3 ソフトウェア開発管理システム

本節では, 前節で述べたモデルを用いて構築されるソフトウェア開発管理システムについて述べる. まず本システムの概要を示し, 今回行った試作の内容について述べる.

3.1 システムの概要

本システムは, ソフトウェア開発を行なう開発者に対して開発作業支援を行ない, かつその際の情報を管理者に対して呈示し, 進捗状況の把握を容易に行えるようにすることを目的とする. 図2に, 開発管理システムの概要図を示す.

本システムはサーバクライアント型の形態を取っており, 主に利用者のためのユーザインタフェース部分と, MonoProcess オブジェクトを保持するオブジェクトベース部分から構成されている. ユーザインタフェースはリポジトリと通信動作を行なうことによって, オブジェクトベース内にあるリポジトリの内容を更新する.

3.1.1 オブジェクトベース

オブジェクトベース部は主に, マネジャ, ライブラリ, リポジトリ, メソドエンジンから構成されている.

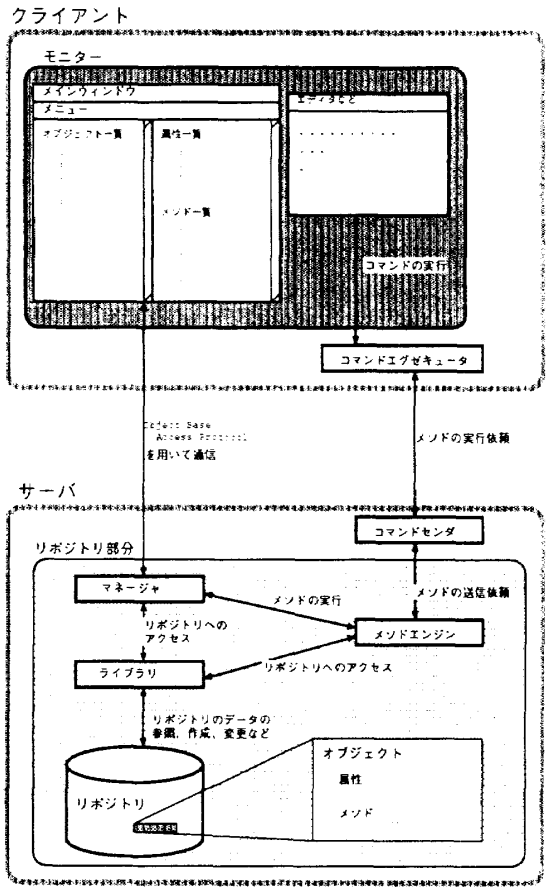

図 2: システムの概要

マネジャはオブジェクトベースの中心となる部分であり，ユーザインタフェースからの要求を受けつけリポジトリの変更や参照，メソドエンジンの起動等のを行なう．ライブラリはマネジャやメソドエンジンが利用する，リポジトリの操作を行なうための関数群である．メソドエンジンはユーザインタフェース側からの要求に応じて，メソドを解釈実行するものである．

リポジトリが操作される際には，それらの操作は全てリポジトリ側で自動的にその操作履歴が記録される．履歴は各属性，メソドごとに記録される．これらの記録は各オブジェクト単位や作業者単位でも同時に記録することにより，履歴の参照を高速に行なうことができる．リポジトリに対する操作は，各作業者ごとに操作できる対象が制限される．これにより，作業者が誤った操作を行なうことを未然に防ぐことができる．

3.1.2 ユーザインタフェース

ユーザはユーザインターフェースを用いて，オブジェクトベース内のオブジェクトを操作する．オブジェクトに対して行える操作は以下の通りである．

(1) オブジェクトの作成・削除

(2) 属性の作成・削除・変更・参照

(3) メソドの作成・削除・変更・参照・実行

ソフトウェア開発者は，ユーザインタフェースを用いてオブジェクトの属性を参照することによって，現在自分が担当するプロダクトや他の作業者が開発を行なっているプロダクトの状況を把握することができる．また，該当するプロダクトを表わすオブジェクトに設けられているメソドの実行を行なうことにより，その作業に必要となるツール等を起動したり，他の作業者への連絡などを行なうことができる．管理者はこれらの作業等に加え，オブジェクトベースに蓄えられているオブジェクトの属性やメソドの実行履歴をインタフェースを用いて参照することによって，現在行なわれている開発作業の進捗状況を把握できる．

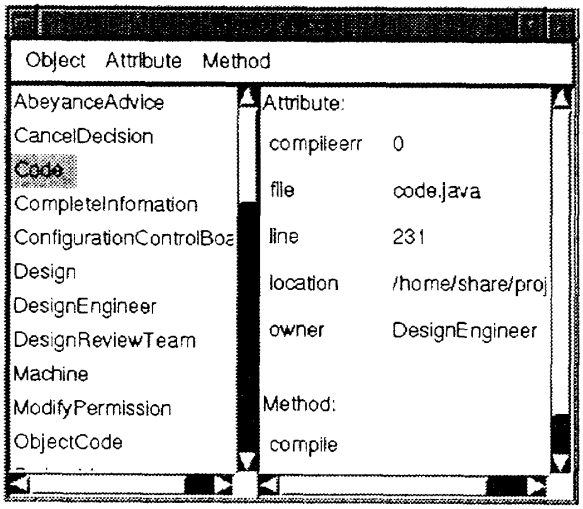

図 3: インタフェースの外観

ユーザインタフェースの外観を図3に示す．画面全体は大きく縦に二分されており，左側には記述されているオブジェクト一覧が，右側には左側で選択されたオブジェクトが持つ属性やメソドの一覧が表示される．また，画面上部には，各オブジェクト，属性，メソドに対する操作を行なうためのメニューが配置されている．

オブジェクトの操作 オブジェクトの操作は，主にインタフェース上部のメニューから該当する操作を選択することによって行なえる．

例えば，オブジェクトの生成を行なう場合には，Object メニュー内に存在する項目 New を選択する．これを選択すると，生成するオブジェクトの名前やオ

ブジェクトが持つ属性，メソドの入力を促すウィンドウが新たに表示される．属性やメソドを既存のオブジェクトに追加する場合には，画面右側のオブジェクト一覧より追加したいオブジェクトを選択してメニュー内の項目を選択することにより，同様に行なうことができる．

既存のオブジェクトが持つ属性やメソドを変更する場合には，変更対象となる属性やメソドをウィンドウ右側に表示させ選択を行ない，Attribute あるいは Menu メニュー内の項目 Change を選択する．この選択を行なうと，変更内容の入力を促すウィンドウが新たに表示される．なお，属性の持つ値を変更する場合には，画面左側の該当する項目をマウスによりダブルクリックすることによって，その画面上で直接変更を実行することができる．これにより，属性値の細かな修正を容易に行なうことができる．

履歴の表示　後述するオブジェクトベース側で，ユーザインタフェース側で行なった作業内容が記録されている．これらの履歴を参照することにより，管理者が該当するプロダクト等が過去にどのような状態であったかを知ることができ，これらに基づいた正確な進捗状況の把握を行なうことができる．

履歴を表示させる場合には，対象となる属性やメソドを画面左側より選択した状態で，Attribute あるいは Method メニュー内の項目 History を選択する．これによって，該当する属性やメソドに対する全ての操作について，行なわれた時刻，操作内容，操作結果，操作を行なった作業者等の情報がリストの形で表示される．さらに，数値を持つ属性に対する履歴の表示などの場合には，その数値変化だけを抜きだしてグラフの形で表示する．これにより，より具体的に値の変化を把握することが可能となる．

3.2　システムの試作

我々は前節で述べた開発管理システムの試作を行なった．試作したシステムは，分散環境下における小規模のグループによる開発を対象としている．本システムは，開発者に対して作業支援を行ない，管理者が進捗状況の把握を容易に行なえるようにすることを目的とする．

開発者が用いる環境はさまざまであるため，システムを利用するインタフェースとしては可搬性の高い言語を用いて実装をすることが望ましい．そこで，本試作ではユーザインタフェース部を Java 言語を用いて実装した．また，オブジェクトベースは多くの処理を確実に処理できるようにするため，UNIX 上で動作することとした．リポジトリは UNIX ファイルシステムのディレクトリとファイルとして実現されており，オブジェクトベースの中心となるマネジャは C 言語を用いて実装している．メソドエンジンは Perl で記述されており，メソドの記述言語としては Perl を用いた簡易言語を作成してこれを用いた．リポジトリへの API となるライブラリは，等価な機能を C と Perl の両方で実装した．

試作したシステムの具体的な動作については，4節で具体的な適用例と共に述べる．

4　共通例題への適用

本節では3.2節で述べた試作システムを，ソフトウェアプロセスのための共通例題へ適用した具体例について述べる．まず，共通例題について簡単に解説する．次に，本システム上で共通例題がどのように表現されるかを示し，開発者に対する作業の支援や進捗状況の把握がどのように行なわれるかを説明する．

4.1　共通例題

ソフトウェアプロセスのための共通例題[8]とは，ソフトウェアプロセスモデリングにおけるさまざまなアプローチの理解，比較を助けることを目的として設計されたものである．今回は，共通例題で述べられた問題のうち，核問題と呼ばれる部分をとりあげることとする．

核問題は，ソフトウェアシステムの設計，コーディング，単体テストという，ソフトウェア変更プロセスの比較的限定された開発作業について記述を行なっている．例題は，プロジェクトマネージャが変更をスケジューリングし，適当なスタッフに仕事を割り当てることから始まる．新バージョンのコードが新しい単体テストをパスした時点で，例題は終了する．その作業が8つのサブステップに分けて記述されている．共通例題の概略を図4に示す．

本例題は開発時に生成あるいは参照される生成物にどのような物が存在しているかも述べている．それらは，紙の上の手書きのもの(設計書等)と，計算機上にあるもの(ソースコード等)などといったさまざまな形態を定義している．また本例題では，開発を行うプロジェクトチームの組織構造について述べている．プロジェクトチームは対象となるソフトウェアシステムに関する仕事に責任を持つ．チームは一人のプロジェクトマネージャと，ソフトウェアエンジニアのグループで構成される．ソフトウェアエンジニアは設計とコーディングに責任を持つ設計エンジニアと，テストの作成と実行に責任を持つ品質保証エンジニアから構成される．各タスクにおけるこれらの人々あるいはチームの責任は，各々のタスクにおいて示される．また，コンフィグレーション管理委員会という組織がプロジェクトチームとは別に存在し，統括的な方向付けの機能と必要な権限を持っている．

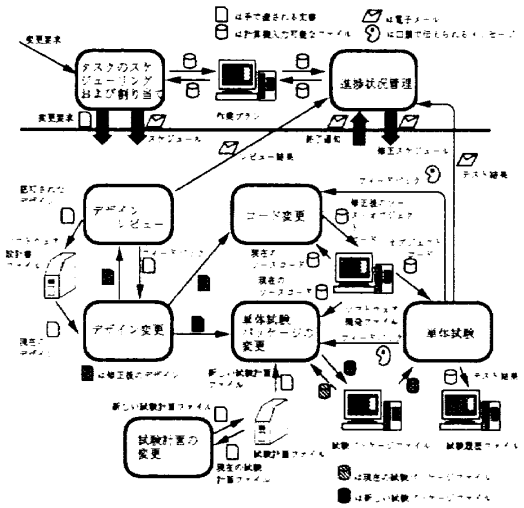

図 4: 共通例題

4.2 システムの適用

4.2.1 オブジェクトの抽出

共通例題を実際の開発システムに適用を行なう場合には，開発者の人数や，計算機上に存在する生成物のファイル名等など，具体的な内容を想定する必要がある．これは，共通例題においてはこれらの具体的な内容についてまでは記述が行なわれていないが，何らかの形で決定されない限り実際の開発作業を行なうための記述としては不十分となるからである．

今回の適用においては，プロジェクトチームを 8 人(プロジェクトマネジャ1 人，コンフィギュレーション管理委員会1人，品質保証エンジニア，設計エンジニア，ソフトウェアエンジニア各2名)で構成し，単一のグループとして作業を行なう状況を仮定した．そして，共通例題で述べられている作業者の役割と，各作業者個人をそれぞれオブジェクトとして記述した．

本システムは計算機上で行なわれる作業を対象としており，その作業履歴を記録することを目的としている．従って，全てのプロダクトは計算機上で扱うこととし，本システムにおいてはオブジェクトとして表現することにする．具体的には，手書きのものは計算機上に置きメールによってやりとりする，口頭で述べられるものについてはメールによってやり取りを行なう．また，すべてのプロダクトが計算機上に実現されているため，各種プロダクトを格納すべき場所も計算機上に実現する必要がある．よってこれらを各種プロダクトを格納するオブジェクトとして定義する．

結果として，生成物を表わすオブジェクトとして 23 個，役割を表わすオブジェクトを 5 個，作業者を表わすオブジェクトとして 8 個の計 36 個を共通例題の記述から抽出できた．

4.2.2 システムの運用

試作したシステムを用いて各オブジェクトの記述を行ない，これに基づいて例題で述べられている作業が実際に動作するかを確認した．以下では，作業がどのようにシステム上で行なわれるかについて述べ，これらの作業の履歴がどのように記録され，表示されるかについて述べる．

メソドによる作業の実行　ソフトウェアエンジニアがソースコードを閲覧する状況を考える．まず，ソフトウェアエンジニアはソースコードを表わすオブジェクトをユーザインタフェースを用いて表示させる．インタフェース上ではこのオブジェクトが持つ属性とメソドの一覧が表示されている．

ソースコードの閲覧を行なう際には，該当するオブジェクトが持つ参照を行なうためのメソドをユーザインタフェースを用いて実行を指示する．この結果，インタフェースを通じてオブジェクトベースへ指示を行なわれる．例えば，閲覧を行なうためのメソドの内容が「実行したユーザの画面へ該当ファイルを表示する」ような操作として記述されていた場合には，具体的には以下の動作が順に行なわれる．

(1) マネージャはクライアントからの要求がメソドの実行であることを判断し，メソドエンジンにソースコードオブジェクト中にある閲覧メソドの実行を依頼する．

(2) メソドエンジンはライブラリを通じてリポジトリから該当メソドの記述内容を入手し，その内容の解釈実行を開始する．

(3) メソドエンジンはリポジトリの内容を参照して，該当ソースコードがどこに保存されているかを検索し，具体的なファイル名を得る．

(4) メソドエンジンは，あらかじめ記述された表示方法を使い，引数に先の操作で得られたファイル名を伴なってコマンドセンダに実行を依頼する

(5) コマンドセンダはコマンドエグゼキュータにコマンドの実行を依頼する．

(6) コマンドエグゼキュータは渡されたコマンド列を実行する．ここでは，この操作を要求したソフトウェアエンジニアが利用するファイル閲覧ツールを，ファイル名を引数に伴なって起動する．

図5はメソドが完了したときに作業者が使用している計算機の画面である．新たなウインドウにプログラムが表示されている．このように，本システムは開発者に対して実際の作業を行なう際に必要なツールを起動する等の支援を行なうことができる．

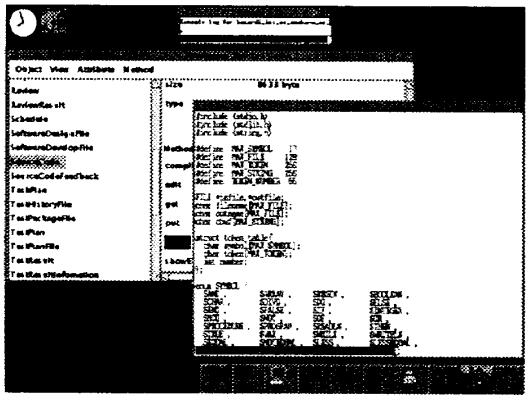

図 5: メソドの実行

履歴情報の表示　プロジェクトマネジャが，ソースコードの変更がどの程度行なわれているかその進捗状況を把握する状況を考える．この場合，例えばプロジェクトマネジャはユーザインタフェースを用いて，該当ソースコードを表わすオブジェクトのファイルサイズを表わす属性を選択し，その履歴をリスト形式で表示するように指定する．この結果，図6 のような画面が表示される．

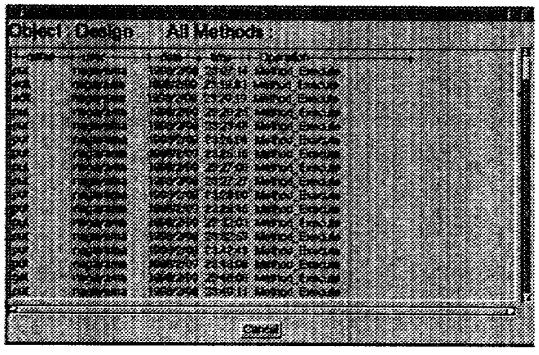

図 6: メソドに対する履歴の例

この図より，作業者がこのオブジェクトに対して，いつどのような作業を行なったかが分かるため，プロジェクトマネジャが進捗状況を把握する際にその判断を行いやすくなる．また，ソフトウェアエンジニアもこの情報を閲覧することによって，誤って行なった作業を発見する際の助けとすることもできる．このように，履歴情報を用いることによって本システムは開発支援の役割も果たしていることが分かる．

以上のように，本システムを実際に例題へ適用することによって，記述されたプロセスをシステム上で動作させ，開発者に対して開発作業の支援を行なえることがわかった．また，管理者に対して，進捗状況の把握を行なうための情報を呈示できることがわかった．

5　まとめ

本稿では，我々がすでに提案を行なっているプロセスモデル MonoProcess に基づいたソフトウェア開発管理システムについて，その概要を述べると共にシステムの試作について説明した．また，試作したシステムをソフトウェアプロセスの共通例題へ適用した例について述べ，本システムを用いることによりソフトウェア開発支援や進捗状況の把握が効果的に行なえることを示した．

今後の課題としては，本システムを完成させ，より効果的なオブジェクトの表示および操作方法の確立やオブジェクトベースにおける実行制御等の機能拡張を行いたい．また，本システムを実際の開発環境下で用いることにより，システムで記録された作業履歴と実際の進捗状況間の相関に関するデータの収集，解析を行いたい．

参考文献

[1] Bandinalli, S., Nitto, E. and Fuggetta, A.: Supporting Cooperation in the SPADE-1 Enviornment, *IEEE Transaction on Software Engineering*, Vol. 22, No. 12, pp. 841–865 (1996).

[2] Bandinelli, S., Fuggetta, A. and Grigolli, S.: Process Modeling in-the-large with SLANG, *Proceedings of the Second International Conference on the Software Process*, pp. 75–83 (1993).

[3] Curtis, B., Kellner, M. and Over, J.: Process Modeling, *Communication of the ACM*, Vol. 35, No. 9, pp. 75–90 (1995).

[4] Fernstrom, C. and Innvation, C. G.: PROCESS WEAVER: Adding Process Support to UNIX, *Proceedings of 2nd International Conference on Software Process*, pp. 12–26 (1993).

[5] 井上克郎: ソフトウェアプロセスの研究動向, ソフトウェア科学会研究報告, Vol. 95, No. SP-2-1, pp. 1–10 (1995).

[6] Kaiser, G., Feiler, P. and Popovich, S.: Intelligent Assistance ofr Software Development and Main-

tenance, *IEEE Software*, Vol. 5, No. 3, pp. 40–49 (1988).

[7] Katayama, T.: A Hierarchical and Functional Software Process Description and Its Enaction, *Proceedings of 11th International Conference on Software Engineering*, pp. 343–352 (1989).

[8] Kellner, M., Feiler, P., Finkelstein, A., Katayama, T., Osterweil, L., Penedo, M. and Rombach, H.: Software Process Modeling Example Problem, *Proceedings of the 6th International Software Process Workshop*, pp. 19–29 (1991).

[9] Matsushita, M., Oshita, M., Iida, H. and Inoue, K.: Conceptual Issues of Object-Centered Process Model, *Proceedings of 1997 Asia-Pacific Software Engineering Conference and International Computer Science Conference*, pp. 519–520 (1997).

[10] 松下誠, 大下誠, 飯田元, 井上克郎: オブジェクトモデルを用いたソフトウェアプロセス記述用言語 MonoProcess, 情処研報, Vol. PRO-14, No. 17, pp. 97–102 (1997).

[11] 落水浩一郎: ソフトウエアプロセスに関する研究の概要, 情報処理, Vol. 36, No. 5, pp. 379–391 (1995).

[12] Sutton Jr., S., Heimbigner, D. and Osterweil, L.: APPL/A: A Language for Software Process Programming, *ACM Transactions on Software Engineering and Methodology*, Vol. 4, No. 3, pp. 221–286 (1995).

[13] Tarr, P. and Clarke, L.: PLEIADES: An Object Management System for Software Engineering, *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Vol. 18, pp. 56–70 (1993).