ソフトウェアプロセスのモデル化と支援環境の構築に関する研究

松下 誠

1998 年 9 月

ソフトウェアプロセスのモデル化と支援環境の構築に関する研究

松下 誠

1998 年 9 月

内容梗概

近年，ソフトウェアを開発する際の過程(ソフトウェアプロセス)を明確にし，その改善によってソフトウェア品質の向上や生産性の向上をめざすことを目標とした研究が活発に行われている．ソフトウェアプロセスに関する研究では，まず開発作業をある観点に基づいてモデル化し，モデルに基づいてソフトウェアプロセスを文字や図等を用いて記述することにより，対象を明確にすることが重要である．これまでに，プロセスモデルやその実行環境等について，さまざまな研究が行われてきている．

プロセスモデルを定義する際の目的は，主に次の (1) ～ (3) の 3 項目に分類することができる．(1) プロセス理解: モデルに基づいたソフトウェアプロセスの記述を用いることにより，開発作業として行わなければならない内容をそれに関わる人々の間で共有し理解することを可能とする．(2) プロセス実行: モデルに基づいたソフトウェアプロセスの記述を開発環境中で実行することにより，作業者に対する開発作業の軽減や，作業の自動化を行うことを目的とする．(3) プロセス改善: 基準となるモデルに基づいてデータを収集，分析し，対象となるプロセスを改善することによってソフトウェアプロセスの品質を向上させることを目的とする．

従来の研究では，これらの 3 つの目的に応じてさまざまなプロセスモデルが提案されてきた．しかし，各目的において充分なモデルを定義することは難しく，解決されなければならない問題も多く指摘されている．

本論文では，実際の開発環境においてソフトウェアプロセスを有効に活用することを目標とした研究の第一歩として，プロセスモデルにおける 3 つの目的である，プロセス理解，プロセス実行，プロセス改善のそれぞれにおける現状の問題点を明らかにする．更に，明らかにした問題点の一部を解決することを目的とした新しいソフトウェアプロセスモデルの提案を行う．モデルに基づいて記述されたソフトウェアプロセスを実際の開発環境中で用いるための支援環境を構築する．支援環境を構築することにより，記述したソフトウェアプロセスを開発環境において役立てることが可能となる．

まず，プロセス理解を目的として，開発環境中における開発者間の連絡や生成物の授受など，開発者間における関係を記述するプロセスモデルを提案する．従来のプロセスモデルにおいては，これらのインタラクションを明示的に扱えなかったが，本モデルを併用することにより作業者間の連絡等を明示的に表現することができる．また，本モデルを用いて記述されたソフトウェアプロセスに基づき，開発作業中における連絡を行うための支援

環境の試作を行った．本モデルと試作環境により，開発環境中に存在する作業連絡等を軸としてソフトウェアプロセスを理解し，それを開発環境中で利用することができた．これにより，従来よりもソフトウェアプロセスをよりわかりやすく表現し理解できることが確認できた．

次に，プロセス実行を目的として，近年の分散開発環境に適した，ソフトウェア開発作業の進捗状況を管理することを目的としたモデルである MonoProcess プロセスモデルを提案する．MonoProcess プロセスモデルでは，ソフトウェア開発環境における生成物や資源をオブジェクトとして記述することができる．本モデルはオブジェクトの集合とその状態遷移によって開発環境を表現する．また，開発環境中における生成物等に対する作業の排他制御や状態の履歴管理等をオブジェクトを用いて抽象化することが可能となっている．本モデルを用いて記述されたソフトウェアプロセスの実行環境を作成するにあたり，近年の分散開発環境下を想定し，既存の開発環境と共存するような実行環境を Web を用いて構築し，運用実験を行った．従来の実行環境では，独自の実行環境を利用することを作業者に対して強いる形となっていたが，本実行環境のような形態でもプロセスの実行環境を構築できることと，実行環境の導入が容易であること，さらに実行環境の有用性を確認できた．この実験の成果に基づいて，MonoProcess プロセスモデルを用いた開発管理支援環境の構築を行った．これらの研究の結果，ソフトウェアプロセスの実行環境と，実行された開発作業の状況を観察することが可能となった．

最後に，プロセス改善を目的として，開発作業の評価を行うために記述された文書 (品質評価規格文書) で述べられている内容を表現するモデルの提案を行い，それを SGML を用いて記述した．品質評価規格文書自体を用いた記述により，品質評価規格が述べている内容と，提案するモデルに基づいた記述を同時に参照することができ，品質評価規格文書の内容をよりわかりやすく表現できることがわかった．また，本モデルを用いて，プロセスの改善手法に関して充分な知識を持たない利用者を対象としたプロセス評価支援環境の構築を行った．本環境により，品質評価規格を用いたプロセス評価作業を容易に実行できた．

以上，プロセスモデルにおける 3 つの目的について，従来の研究で触れられていなかった問題を解決するモデルを構築することができた．また，モデルを用いて記述されたソフトウェアプロセスを解釈するための環境を構築することができた．これにより，実際の開発環境でソフトウェアプロセスを有効に活用するための方法に関する知見を得ることができた．

主要論文一覧

1. 松下誠, 飯田元, 井上克郎: Web を用いたソフトウェア開発環境のためのプロセスモデリング, 情報処理学会論文誌, 39 巻 3 号, pp.830-832 (1998).
2. 松下誠, 飯田元, 井上克郎: 品質評価規格文書のモデル化とそれに基づく評価支援システムの構成, 電子情報通信学会論文誌 D-I, Vol.J81-D-1, No.8, pp.986-993 (1998).
3. 松下誠, 飯田元, 井上克郎: ソフトウェア開発におけるインタラクションの支援方式, 電子情報通信学会論文誌 D-I (採録決定).
4. 松下誠, 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の表現方法, 電気学会 C 部門論文誌 (採録決定).
5. Makoto Matsushita, Hajimu Iida and Katsuro Inoue: An Interaction Support Mechanism in Software Development, In Proceedings of 1996 Asia-Pacific Software Engineering Conference, pp.66-73 (1996).
6. Makoto Matsushita, Makoto Oshita, Hajimu Iida and Katsuro Inoue: Conceptual Issues of Object-Centered Process Model, In Proceedings of 1997 Asia-Pacific Software Engineering Conference and International Computer Science Conference, pp.519-520 (1997).
7. Makoto Matsushita, Makoto Oshita, Hajimu Iida and Katsuro Inoue: Distributed Process Management System Based on Object-Centered Process Modeling, In Proceedings of 2nd International Conference on Worldwide Computing & Its Applications '98, pp.108-119 (1998).
8. Makoto Matsushita, Makoto Oshita, Hajimu Iida and Katsuro Inoue: Web-based Process Management System with Object-Centered Process Modeling, In Proceedings of The International Symposium on Internet Technology 1998, pp.92-97 (1998).

謝辞

本研究の全般に関し，常日頃より適切な御指導を賜わりました，大阪大学大学院基礎工学研究科情報数理系専攻 井上克郎教授に，心から深く感謝申し上げます．

本論文を執筆するにあたり，適切なご助言とご指導を頂きました，大阪大学大学院基礎工学研究科情報数理系専攻 谷口健一教授，菊野亨教授に，心から感謝致します．

本研究を行うにあたり，常日頃より適切なご助言とご指導を賜りました，奈良先端科学技術大学院大学情報科学研究科 鳥居宏次教授に心から感謝致します．

大阪大学大学院基礎工学研究科物理系専攻在席中に，適切なご助言とご指導を頂きました，大阪大学大学院基礎工学研究科情報数理系専攻 藤原融教授，柏原敏伸教授，今井正治教授，都倉信樹教授，萩原兼一教授，藤井護教授，宮原秀夫教授，橋本昭洋教授に感謝致します．

本研究に関して，適切な御助言を頂きました，神戸大学自然科学研究科情報知能工学専攻 荻原剛志助教授に，心より感謝致します．

本研究に関して，直接具体的な御指導を頂きました，奈良先端科学技術大学院大学情報科学センター 飯田元助教授に，心より感謝致します．

本論文を執筆するにあたり，直接具体的な御指導を頂きました，大阪大学大学院基礎工学研究科情報数理系専攻 楠本真二講師に，心より感謝致します．

本研究を行うにあたり，ご助言やご指導を頂きました，株式会社日立製作所 青山和之氏に感謝致します．

開発環境の設計に関し，ご協力を頂いた大阪大学大学院基礎工学研究科情報数理系専攻 山本哲男氏に感謝致します．

開発環境の設計と試作に関し，ご協力を頂いた大阪大学基礎工学研究科情報数理系専攻 (現 日本オラクル株式会社) 大下誠氏に感謝致します．

開発環境を用いた実験に関し，御協力を頂いた，大阪大学基礎工学研究科情報数理系専攻 (現 沖電気工業株式会社) 島本勝紀氏に感謝致します．

最後に，井上研究室の皆様の御助言，御協力に御礼申し上げます．

目 次

図目次

表 目 次

第1章　まえがき

1.1　ソフトウェアプロセス

ソフトウェアシステムの規模はますます増大する傾向にある．このため，ソフトウェア開発作業はますます複雑となってきている．また，ソフトウェアシステムが担う社会的役割もますます重要となっており，高品質なソフトウェアを効率良く開発することは，ソフトウェアに関する研究において重要なテーマとなっている [41].

ソフトウェアシステムの品質改善，開発作業における生産性の向上を目指して，さまざまな研究活動が現在行われている．その中でも，ソフトウェアが開発される過程 (ソフトウェアプロセス) を明確にし，その改善によってソフトウェア品質の向上や生産性の向上をめざす研究分野が近年注目されている [14]．ソフトウェアプロセスを理解し，ソフトウェアプロセスがソフトウェアシステムの品質や生産性に与える影響について多くの研究が行われてきている [15, 50].

ソフトウェアプロセスに関する研究の中心は，まずソフトウェアの開発過程をモデル化 (ソフトウェアプロセスモデル，あるいはプロセスモデル) し，それを文字や図などを用いて記述することによって，ソフトウェアの開発過程を明示的に扱えるようにすることである．モデルを用いてソフトウェアプロセスを記述し，それを実際の開発において利用することによって，質の高いソフトウェア開発を行うための基礎とすることができる．また，曖昧に定義されがちなソフトウェア開発過程を明確にすることができ，開発を行うために必要となる情報の不足を検出したり，矛盾や誤解なくソフトウェア開発過程を理解することができる．

プロセスモデルは，ソフトウェア開発環境に存在するさまざまな事象をある観点に基づいて抽象化したものである．従って，扱われる範囲によって多様なプロセスモデルが存在する [18]．例えば，プロセスモデルはソフトウェア開発のライフサイクル全体を扱うこともあるし，ソースコードの編集といった小さな単位を扱うこともある．文献 [7] におけるスパイラルモデルなどは前者の例であろう．

1.2 プロセスモデルの目的

プロセスモデルには，主に以下の 3 つの目的がある [12].

1. プロセス記述を用いた開発作業の理解 (プロセス理解)

    複数の作業者によってソフトウェアを作成するような開発形態では，共同作業をする人々の間において，開発を実行するにあたって用いるソフトウェアプロセスに関して共通の理解を持つことが重要である．お互いの作業を理解することによって，共同作業や相互のコミュニケーションを円滑に進めることができる．また，自分自身の作業内容を把握することにより，作業を正確にかつ効率よく進めることが可能となる．

    また，ソフトウェアプロセスを理解することは，ソフトウェアの開発者だけではなく，プロジェクト管理者にとっても同様である．現在行われている開発作業が事前に設定したソフトウェアプロセスの手順通りに進められているかどうかを判断するためには，ソフトウェアプロセスの理解が重要である．

    プロセスモデルを用いることで，ソフトウェアプロセスを理解することを助けることができる．モデルを用いてソフトウェアプロセスを記述することにより，ソフトウェアプロセスを明確に定義することができる．また，図などを用いて表現することにより，記述された中身をわかりやすく理解することができるであろう．

2. プロセス記述を用いた開発作業の実行 (プロセス実行)

    プロセスモデルを用い，文字や図などによって記述されたソフトウェアプロセスをプログラムとして捉えた場合 [38]，それをある解釈系を用い，特定の開発環境上で実行するさせることが可能である．

    そもそもソフトウェア開発における作業は，大きく 2 つに分類することができる．1 つは 人間が行う創造的な作業であり，もう 1 つは計算機上で動作するツール等によって機械的に行われる作業である [10]．記述されたソフトウェアプロセスを実行することは，人間が行う作業に対して補助的な作業を行うことと，機械的な作業を解釈系によって自動的に実行することなど

が含まれる．

人間に対する補助的作業には，例えば次の作業の指示や，現在行われてる他の開発作業等に関する情報提供などがある．また，機械的な作業を自動実行する場合でも，作業者に対して自動実行された作業内容を提示する等が人間に対する補助的作業として含まれる．また，実行されたソフトウェアプロセスの状況を開発管理者が観察できるようにすることにより，開発作業の進捗状況を管理する上で有用な情報を提供することが可能となる [3]．

このように，ソフトウェアプロセスを実行させることにより，開発作業の補助や進捗状況の把握など，曖昧にされやすい開発作業の内容をより明確にすることが可能となる．

3. プロセスモデルを用いた開発作業の改善 (プロセス改善)

理想的な開発を行えるようなプロセスモデルを事前に定義し，それを既存の開発環境に照らしあわせ，モデルとどの程度適合しているかによって，その開発環境がどの程度効率よくソフトウェア開発を行っているかを判断する試みが近年広く行われている．また，その判断を用いて，開発作業の改善を行う動きがさかんに行われている．

このようなソフトウェアプロセスの改善を行うことを目的として設計されたプロセスモデルには，SEI (Software Engineering Institute) の CMM (Capability Maturity Model)[39, 40] に代表される，一般的に共通のモデルとして用いられているものや，個別の組織が独自に理想として設定するものがある [52]．

これらのモデルを用いる場合には，まず現状の開発作業について，具体的なデータを収集しそれを整理して，プロセスモデルとどの程度適合しているかを判断する．次に，適合していない部分を整理し，その部分を改善するための方針と目標を設定し，それを実際の開発作業にフィードバックすることによって，プロセスモデルを開発作業の改善に役立てることが可能となる．改善作業を行った後，再度データの収集を行うことによって，目標が達成されているか確認することができる．

1.3 既存のプロセスモデルにおける問題点

既存のソフトウェアプロセスに関する研究では，前節で述べた目的に応じてさまざまなプロセスモデルが提案されている．しかし，既存のモデルでは以下のような点が不足していると考えられる．

1. モデルを用いて記述された内容が誤りなく理解できることは重要であるが，理解した内容をソフトウェアの開発時において役立てることがこの種のモデルでは期待される．近年の複数人における共同作業としてのソフトウェア開発においては，作業者間との協調作業は重要な要素であるため，プロセスモデルにおいても，協調作業を明確にモデル化できることが望ましい．

   しかし，既存のプロセスモデルにおいては，この種の協調作業を関数として捉えた場合の引数といった形で扱う物はあるが [44]，開発作業における作業者間，あるいは作業者とツール間などの多様な関係を記述するための柔軟性に欠ける．

   開発作業における作業者間や作業者とツール間などの関係 (インタラクション) を明確にプロセスモデルとして抽象化することにより，開発作業を理解することはもちろんのこと，作業間の関係をよりわかりやすくすることができると考えられる．
2. 近年の分散環境におけるソフトウェア開発を表現するようなソフトウェアプロセスモデルを用いて記述された内容を実行する場合には，分散環境に適応した実行環境を持つ必要がある．開発者は，この種の実行環境を用いてソフトウェア開発を行うことになる．

   しかし，既存のこの種の研究においては，それ自身が閉じた環境を持っており，作業者はその環境中での作業を強いられるものや [16]，既存の開発環境をプロセス実行環境に適合させる実行環境であり [10]，優れたプロセス実行環境であっても実際の開発作業現場に導入しにくいといった問題がある．

   この種のプロセス実行環境を構築する場合には，既存の開発環境になるべく影響を与えない形で構築できることが望ましい．これにより，プロセス実行環境を容易に導入することが可能となる．
3. モデルを用いて開発環境の評価を行う場合，モデルとして既存の一般的なモデル (品質評価規格) を用いる形態が一般的である．[21, 42]．この種のモデルでは，モデルを十分に理解した担当者によって評価作業を行うこと，評価のために必要な情報を充分

に収集し，第三者によって客観的な評価を行うことが定められている．
しかし，これらの評価作業を実際に行うには多くの人的や金銭的コストがかかるため，特に小さな開発環境などにおいては評価を導入しにくいといった問題が指摘されている [49]．一方，開発組織自身によるソフトウェアプロセスの評価方法が求められており，これに関する研究も行われているが，各組織が個別に評価基準を定めることは難しいとされている [12]．
一般に用いられている品質評価規格を，開発組織自身によって自己評価する際に用いることにより，これらの問題を解決することが可能であると考えられる．

1.4 本論文の概要

本論文では，実際の開発環境においてソフトウェアプロセスを有効に活用することを目標とした研究の第一歩として，ソフトウェアプロセスにおける 3 つの目的である，プロセス理解，プロセス実行，プロセス改善のそれぞれにおける問題点をまず明らかにする．

更に，これらの問題の一部について，これを解決する新しいソフトウェアプロセスモデルの提案と，モデルを用いて理解，実行，改善を行うための支援環境の構築方法について述べる．

- 開発環境中における構成要素の相互関係を用いたプロセス理解のための環境
  ソフトウェアプロセスを理解しやすくすることを目的として，生成物や資源といった開発環境中の構成要素と，その要素間に存在するインタラクションに着目する従来にない視点を持ったモデルを構築した．また，モデルの実行環境の試作を行った．これにより，ソフトウェアプロセスにおけるインタラクションをよりわかりやすく表現し理解できることがわかった．

- 既存環境と共存する分散環境を想定したプロセス実行のための環境
  近年の分散開発環境に適した，ソフトウェア開発作業の進捗状況を管理することを目的としたモデルである MonoProcess プロセスモデルを提案した．また，分散環境をを想定した，従来にない既存の開発環境と共存するプロセスの実行環境を Web を用いて構築し，運用実験を通じてその有効性を確認した．本実験の成果を用いて，MonoProcess プロセスモデルを用いた実行環境の試作を行った．これにより，ソフトウェアプロセス実行しその実行状況を観察することができた．

- 開発作業者による評価を想定した品質評価規格によるプロセス評価環境

  品質評価規格で述べられている開発作業のモデルを SGML を用いて記述した．また，また，本モデルを用いて，プロセスの改善手法に関して充分な知識を持たない利用者を対象としたプロセス評価支援環境の構築を行った．これにより，開発組織が自己のソフトウェアプロセスを評価する際に，既存の品質評価規格を用いて容易に評価することが可能となった．

以下，2 章では，開発環境中の構成要素の相互関係を用いたモデル化について述べる．3 章では，既存環境と共存する分散環境を想定したプロセス実行のための環境について述べる．4 章では開発作業者による評価を想定した品質評価規格によるプロセス評価環境について述べる．最後に 5 章で本論文の研究について纏め，今後の研究の方針について述べる．

第2章　開発環境中の構成要素の相互関係を用いたプロセス理解のための環境

近年のソフトウェア開発はより分散化してきており，ネットワークを用いた開発形態はごく一般的になってきている [1]．このような開発形態においては，従来の一箇所に集中して行っていた開発とは異なり，進捗状況を開発者が相互に理解し，かつ開発者間の相互連絡を確立することが非常に重要となる．

これらの問題を解決するために，ソフトウェア開発におけるそれぞれの開発作業を明らかにし，それら相互の関連を考えるソフトウェアプロセスの研究が行なわれている [14, 35, 50]．これらの研究では，ソフトウェア開発作業とそれに関連する開発環境を再構成することによって問題の解決をはかっている．ソフトウェアプロセスに関する研究分野として，ソフトウェアプロセスをモデル化して記述するプロセスプログラミング [38] や，記述されたソフトウェアプロセスの定義に基づいてソフトウェア開発の作業者を支援するプロセス中心型開発環境の構築 [16, 17, 43] などがあげられる．

ソフトウェア開発作業を，作業者間の対話や中間生成物の受け渡しといったインタラクションに着目して捉えた場合，ソフトウェア開発作業自体はさまざまなインタラクションから構成されると言える．各開発者は，さまざまな対象とインタラクションを行いながら開発作業を進行させている．しかし，従来提案されていたソフトウェアプロセスモデルやそれに基づいたプロセス中心型開発環境では，これらのさまざまな種類のインタラクションは明示的に扱われていないか，若しくは単純な作業の型の１つとしてしか扱われていないため，インタラクションの記述や支援が十分ではない．

本節では，ソフトウェア開発プロセスを構成する要素間の相互関係 (インタラクション) に着目したプロセスモデルを提案する [24, 28, 32] ．本モデルにより，ソフトウェア開発プロセスにおけるインタラクションを明示的に記述することができ，ソフトウェアプロセスを理解するための枠組を提供できる．また，インタラクションを支援する開発環境の設

計を行う．本モデルではソフトウェア開発環境におけるインタラクションを実際にインタラクションを行う主体としてのエージェントと，インタラクションの内容を表現する通信チャネルの集合として表す．

以下 2.1 節では，ソフトウェアプロセスにおけるインタラクションについて考察する．2.2 節では，インタラクションのモデル化方法について説明する．2.3 節では，提案するモデルに基づいた開発支援環境について述べる．最後に 2.5 で本研究について纏める．

2.1 開発環境における相互関係

本節では，ソフトウェアプロセスにおける開発作業と，開発作業中におけるインタラクションについて述べる．

ソフトウェアプロセス

ソフトウェア開発作業の各要素をソフトウェアプロセスとして記述する方法はさまざまな方法がある．現在の開発作業をソフトウェアプロセスとして記述することによって，開発組織全体で適用する標準プロセスやプロセスの開発環境への適応などが非常に簡単に行える．

また，記述されたソフトウェアプロセスは量的あるいは視覚的方法による進捗状況の把握にも用いることができる．形式的に記述されたソフトウェアプロセスを用いることによって，記述自体を計算機によって自動的に実行させることができる．また，ソフトウェア開発のシミュレーションを行うための基礎とすることができる．

ソフトウェア開発中におけるインタラクション

ソフトウェア開発環境には，さまざまな種類のインタラクションが存在する．例えば「生成物の受け渡し」「作業の開始/終了通知」「共同作業者間での会話」などはインタラクションである．また，インタラクションの内容は多岐にわたる．例えば，インタラクションの例として，次のような状況を想定する（図2.1）．

> あるアプリケーションの開発が行われているとする．このアプリケーションは共通ライブラリ（$libR$, $libS$, $libT$ の 3 つ）と，それを使ういくつかのコンポー

ネントから構成されているとする．

共通ライブラリの開発を行う作業者が 3 名（A, B, C）おり，各ライブラリはそれぞれ複数人で開発を行っており，開発責任者があらかじめ決定されているとする．ライブラリの開発責任者はライブラリの開発状況を把握する仕事を受けもつ．

また，コンポーネントの開発者，つまり，共通ライブラリを利用する作業者が 4 名（X, Y, Z, W）おり，各開発者は（1 つ以上の）共通ライブラリを利用してコンポーネントの開発を行っている．これらの開発者は，自分が利用する共通ライブラリに関する告知（例えば仕様変更やバグフィックス報告など）を知りたいと思っており，あるいはライブラリの開発者に対して質問を行ったり，他のライブラリの利用者とライブラリに関する情報交換を行いたいとも思っている．

このような状況において，例えば以下のようなインタラクションが存在する．

- 共通ライブラリ利用者が新たなライブラリを利用し始める
  もしコンポーネントの開発者がこれまで利用してなかった共通ライブラリを利用したいと思った際には，各種の告知を送ってもらうよう，該当する共通ライブラリの開発者へ知らせておく．但し，自分の使っていない共通ライブラリの告知を読む必要はないため，自分が使っている共通ライブラリに関する告知だけを受けとる．

- バグを発見した場合にそれを報告する
  共通ライブラリの中にバグを発見した場合，それを該当するライブラリの開発者へ報告する．また，報告されたバグがどう修正されたか（されなかったか）の結果が通知される．あるいは，誰かが既に発見したバグについても同様に通知される．

- 不明な点について問いあわせを行う．
  共通ライブラリの機能や利用方法についての質問を開発者に投げる．

この例はごく小さな開発チームが仮定されているにも関わらず，多くの種類のインタラクションを挙げることができる．

本研究では，開発環境においてメッセージのやりとりやファイルの送受信等，計算機を用いて行われるインタラクションを支援するための，プロセス中心型インタラクションモ

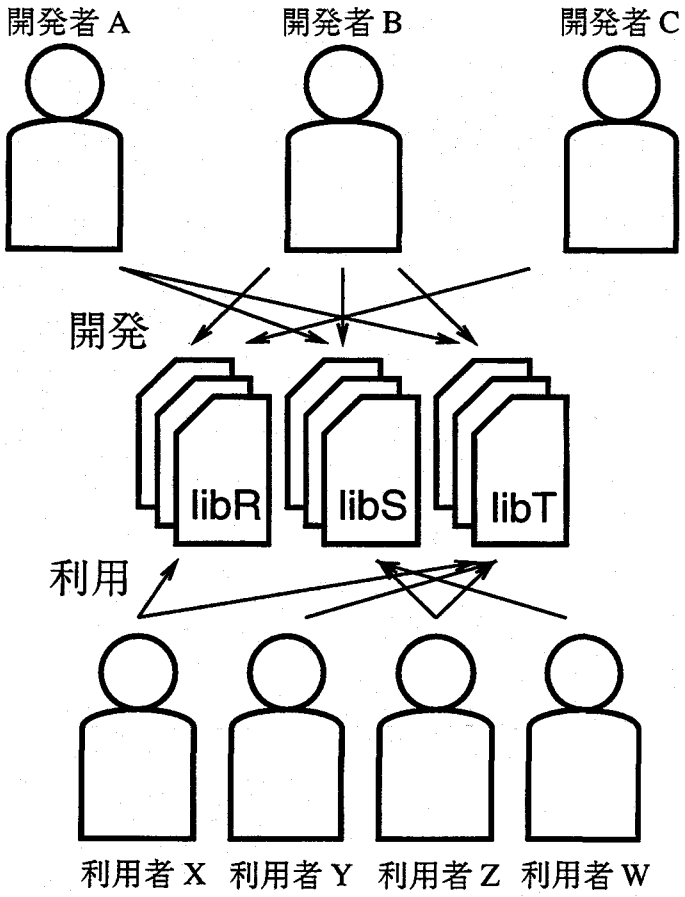

図 2.1: アプリケーション開発の例題

デルを提案する．本モデルは，インタラクションのひな型のためにソフトウェアプロセス記述を用いている．モデルを表現するために，作業エージェント（WorkAgent）と呼ぶインタラクションの主体となる物を導入する．作業エージェントはチャネル（Channel）を用いてインタラクションを行う．

2.2 相互関係のモデル化

本節では，本論文で提案するプロセス中心型インタラクションモデルについて説明する．まず最初に，モデルに対する要求事項を纏めた上でモデルの概要について説明する．次に，モデルで用いられる作業エージェントとチャネルの詳細について解説する．最後に，前述の例を用いて具体的なインタラクションの記述例や記述されたインタラクションの表現を示す．

2.2.1 必要とされる事項

提案するモデルは，ソフトウェアプロセスにおけるインタラクションを記述し支援するために次のような機能が必要となる．

- プロセス中心型であること
  本モデルは，効率の良いインタラクションの方法を表現する必要がある．つまり，プロセスの構造は直接インタラクションの構造を直接表現できる必要がある．

- インタラクションの主体を考慮すること
  開発環境中で行われるあらゆるインタラクションを把握するために，提案するモデルはインタラクションの主体，つまり，インタラクションを実際に行っている作業者等について考慮する必要がある．

- インタラクションの中身を考慮すること
  モデルはインタラクションの中で扱われれている情報の内容もまた考慮しなければいけない．なぜならば，この種の情報は開発プロセスの管理等を効果的に行う上で非常に重要な対象となるからである．

- 簡潔な枠組であること
  本研究では，このモデルが他のプロセスモデルと併用して用いられることを想定し

ている．つまり，開発作業中で用いられるさまざまなプロセスモデルにおける1つのコンポーネントとなることを想定している．従って，インタラクションモデルは適度な粒度と簡潔さを持って，汎用のプロセスモデルにおける他のコンポーネントと統合できるものでなければならない．

2.2.2 定義

概要

本モデルではまず，ソフトウェアプロセスをタスクの集合として定義する．各タスクはアクティビティの系列とする．アクティビティはソフトウェアプロセスにおける不可分の動作であり，各タスクは他のタスクと相互にメッセージを交している．

2.2.1 節で述べた要求を満たすため，ここではオブジェクト指向風のモデルを採用した．モデルには作業エージェントとチャネルと呼ばれる2つのクラスがあり，これらを用いてモデルを構成する．

作業エージェントはインタラクションとなるイベントを発生される実体であり，そのインスタンスは，人間，プログラムあるいは外部のシステムによって実行されるタスクとなる．

チャネルはインタラクションの内容および種類に基づいてクラス分けされた仮想的な通信路である．そのインスタンスはソフトウェアプロセスにおけるある話題，例えば「単体テストとその実行結果」などを表現するために生成される．

作業エージェントは他の作業エージェントとチャネルを媒体としてインタラクションを行う．この際，作業エージェントは媒体として用いるチャネルに接続することによって情報の送受信を行う．

形式的には，提案するプロセス中心型モデルは以下のように定義できる．

インタラクション ::=
　[作業エージェントの集合, チャネルの集合, 接続]
作業エージェント ::= イベントを発生する有限状態機械
チャネル ::= 作業エージェントが用いる仮想通信路
接続 ::= チャネルから作業エージェントへの対応関数

以下では，作業エージェントとチャネルの詳細について述べる．

作業エージェント

インタラクションをモデル化するという観点で言えば，すべてのタスクはインタラクションを生成する実体であると考えられる．よって，タスクを作業エージェントとしてモデル化する．言いかえれば，作業エージェントは一連の実行の間メッセージのやりとりを行ってインタラクションを行っているものである．

作業エージェントは次のように定義される．

作業エージェント ::= [名前, 属性の集合, 作業内容の集合]
名前 ::= 文字列
属性 ::= [属性名, 属性値]
属性名 ::= 文字列
属性値 ::= スカラー値 (範囲はプロセス記述毎に決定)
作業内容 ::= [前条件, タスク]
前条件 ::= 入力に対して真偽値を返す関数
タスク ::= アクティビティの系列
アクティビティ ::= 開発中における不可分の動作

作業エージェントは名前，属性，作業内容から構成される．名前は作業エージェントを特定するために用いられる．属性は列挙型の変数であり，作業エージェントの行う作業の特性を表現するために用いられる．このため，各変数の値域についてはプロセスを記述するたびごとに定義される．作業内容はアクティビティの系列であり，その実行前には前条件を満たすことが要求される．プロセス中に存在するアクティビティについては各プロセスおよび作業エージェントごとに決定される．

チャネル

ソフトウェアプロセス中のインタラクションは，作業エージェントによって生成された一連のメッセージとして定義する．これらのインタラクションは，その内容，インタラクションを行う形態，関連する作業エージェント等に基づいて分類することができる．

従って作業エージェント間には，分類された作業エージェントによって交される一連のメッセージによってある関係が定義できる．本モデルではこの関係をチャネルとして定義する．

チャネルは次のように定義される．

チャネル ::= [名前, 属性の集合, 閉包, 情報形式の集合]
名前 ::= 文字列
属性 ::= [属性名, 属性値]
属性名 ::= 文字列
属性値 ::= スカラー値 (範囲はプロセス記述毎に決定)
閉包 ::= [内閉包, 外閉包]
内閉包 ::= 作業エージェントで定義される属性値の組
(外閉包は内閉包と同様に定義される)
情報形式 ::= [型名, フィールド宣言の集合]
型名 ::= 文字列
フィールド宣言 ::= [フィールド名, フィールド型]
フィールド名 ::= 文字列
フィールド型 ::= 文字列, 論理値など

チャネルは名前，属性，閉包，情報形式から構成される．名前と属性については作業エージェントと同様に定義され利用される．

閉包はインタラクションを行う範囲を作業エージェントの型を用いて制御するために定義される．閉包には内閉包と外閉包の 2 種類がある．内閉包はチャネルに必ず接続していなければならない作業エージェントを指定し，外閉包はチャネルに接続できない作業エージェントを指定する．閉包の指定は作業エージェントのもつ属性の組で行う．これにより，例えば内閉包を用いることによって必要なインタラクションを定義でき，外閉包を用いることによって情報を隠蔽したい範囲を定義することができる．情報形式はチャネルで交されるメッセージの型定義であり，型の名前とメッセージの構成を宣言するフィールド宣言から構成される．

作業エージェントはチャネルへの接続/切断動作をチャネルを用いてインタラクションを行う前/インタラクションが終了した後に必ず行う．これによって，チャネルへの接続状態が現在の作業エージェントが関連するインタラクションの数および種類を表現できる．

2.2.3 記述例

ここでは，具体的にモデルによるインタラクションの表現方法について紹介し，どのように作業エージェントやチャネルが動作するかを説明する．このために，2.1 節で用いたアプリケーション開発の例題を再度用いることとする．

まず図 2.2 は，この例題をモデルを用いて表現したものである．

この図では，角の取れた四角形が作業エージェントを示し，二重円がチャネルを示す．この図では，7 つの作業エージェントと 3 つのチャネルが書かれている．作業エージェント

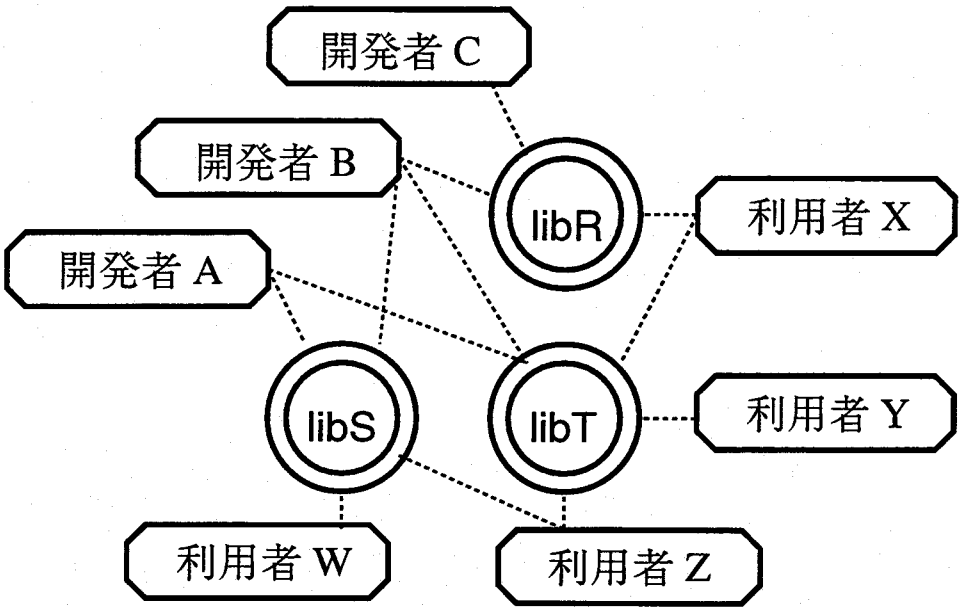

図 2.2: モデルを用いた例題の表現

とチャネル間に引かれた点線は，作業エージェントとチャネルとの接続を示す．以下では，これらの記述を行う手順の詳細を順を追って説明する．

作業エージェントおよびチャネルの特定

例題のような複数のライブラリを用いた開発の際に発生するインタラクションには，さまざまなものが考えられる．例えば，特定のライブラリに関して「開発者間の連絡」「開発者と利用者間における情報交換」「開発中ライブラリの流通」といったインタラクションが考えられる．これらを表現するためには，これらの分類したい項目それぞれにチャネルを定義することになる．また,「ライブラリ中で定義される関数の利用方法」のようなインタラクションは，各ライブラリの相互作用等が話題になることも考えられるため，ライブラリの数に依存せず全体で 1 つチャネルを定義すれば十分であろう．このように，本モデルでは，ある状況に対して複数のモデルによる記述を行うことが可能である．ここでは，記述をわかりするために，単一のプロダクトに対して 1 つのチャネルを割り当て，それぞれのライブラリに関するインタラクションを単一のチャネルで表現することとする．

この例題では，共通ライブラリの開発者と利用者との間でインタラクションが発生すると考えられる．よって，ここでは開発者および利用者をそれぞれ作業エージェントに対応

付けることとする．インタラクションの内容としては互いに開発/利用している共通ライブラリに関する事項となるであろうことから，ここではライブラリの名前をチャネルに対応付けた（図 2.2）．

各作業エージェントからチャネルへの接続は，それぞれの開発者/利用者が関連している共通ライブラリへの関係を示すように引かれている．つまり，開発者を示す作業エージェントはその開発者が担当している共通ライブラリのチャネルへ，また，利用者を示す作業エージェントはその作業者がコンポーネントの開発の際に利用している共通ライブラリのチャネルへ引かれている．

作業エージェントの記述

図 2.3 は開発者 A に対応する作業エージェントの記述例である．

```
define workagent Developer_A
begin
   attribute begin
      role: develop;
      primary: S;
   end
   vars begin
      bugbuffer: array of bug-report;
      qbuffer: array of question; res: bool;
   end
   initaction begin  join(libS); join(libT);  end
   primitive add consult, debugging, makeans;
   action begin
      channel libS begin
         bug-report:
            push(bugbuffer, info); res=consult(info);
            if (res == True) { send(libS, debugging(info); }
         question:
            push(qbuffer, info); send(libS, makeans(info));
      end
      channel libT begin
         bug-report:
            res = consult(info);
            if (res == True) { send(libT, debugging(info); }
         question:
            send(libT, makeans(info));
      end
   end
end
```

図 2.3: 作業エージェントの記述

作業者 A の属性として，ここでは役割と主担当という 2 つの属性を定義した．まず役割

は「開発プロセスにおけるこの作業エージェントの役割分担」を示し,「開発者」「利用者」の2つの値を取るものとした．主担当は「作業エージェントが主に関わるライブラリ」を意味し，開発者を表す作業エージェントであれば自らが開発責任者となっているライブラリの名前，利用者を表す作業エージェントでは自らが利用するライブラリのうち一番利用頻度の高いライブラリの名前を表現するものとした．

作業エージェントが実際に動作するための記述を行うために，2.2.2節で述べた内容（attribute, action節）の他に，変数定義（vars節），初期化動作（initaction節），基本動作宣言（primitive節）等の記述を行う．

変数定義は作業エージェント内部で情報やインタラクションを保持するために用いられる変数を定義するものである．初期化動作では作業エージェントが動作を開始する際に行っておく動作を定義する．この例では，2種類のチャネルへの接続動作が定義されている．基本動作の宣言ではどのような動作が記述する作業エージェントにおけるアクティビティであるかを宣言するが，その動作の具体的内容はモデルの考慮外であるために記述されない．これにより，同じ基本動作宣言を記述しながら，作業エージェントが動作する環境によって実際の挙動を変更することが可能となるため，作業エージェント記述の可搬性を高くすることができる．

チャネルの記述

図2.4は，共通ライブラリ *libS* に対するチャネルの記述例である．

作業エージェントと同様，チャネルの属性も定義される．ここでは，ライブラリ名が属性として定義されている．2.2.2節で述べた属性，閉包，情報形式はそれぞれattribute, restriction, infotype節として記述されている．

この例では，チャネルに対して内閉包の定義があり「共通ライブラリ *libS* の開発者は必ずこのチャネルに接続していなければならない」ことを記述している[1]．また，4つの型名が定義されており，それぞれ「告知」「バグ報告」「質問」「回答」を示している．各型名には複数のフィールド宣言があり，話題を示す文字列やキーワード，メッセージの書かれたファイルなどの存在が定義されている．

[1] 外閉包を定義する場合，例えば「開発者は接続してはいけない」チャネルを定義する際には，対象となる開発者を示す作業エージェントがもつ属性の組を指定する．

```
define channel libS
begin
   attribute begin
      libname: S;
   end
   restriction
      in: (develop, S)
   end
   infotype begin
      announce:
         (topic: string, contents: file);
      bug-report:
         (topic: string, contents: file);
      question:
         (topic: string, body: file, keyword: string);
      answer:
         (topic: string, body: string);
   end
end
```

図 2.4: チャネルの記述

作業エージェントとチャネルの振舞い

図 2.2 で示されている作業エージェントとチャネル定義や接続状況は定義できているため，これを用いて作業エージェントやチャネルの記述が実際にどのように動作するか，いくつかのシナリオを用いて説明する．

シナリオ 1: 新しいライブラリの利用　利用者 Y が新たに共通ライブラリ $libR$ を利用し始める状況を考える．この場合，利用者 Y に対応する作業エージェントで "join（$libR$）" という動作を行う．これによって，この作業エージェントが $libR$ を利用することを示す．また，これによって $libT$ の利用を中止する場合には，"leave（$libT$）" という動作を行ってチャネル $libT$ からの切断を行うことになる．

シナリオ 2: バグの発見　開発者 B が共通ライブラリ $libS$ にバグを発見した状況を考える．まず，開発者 B はバグ報告をチャネル $libS$ に送信する．このバグ報告はチャネル $libS$ に接続しているすべての作業エージェント，つまり，利用者 W, Z と開発者 A に送信される．開発者 A は共通ライブラリ $libS$ の開発責任者であるため，A は送られたバグ報告をバッファに蓄え，これが本当にバグであるかの検証を行う．その結果バグであることが判明したため，開発者 A はデバッグを行ってその結果をチャネル $libS$ に送信する．

なお，ライブラリの開発者 B だけでなく，チャネル $libS$ に接続しているすべての作業

エージェントがチャネルを用いてバグ報告を送信することができ，その報告もまた同様にチャネル *libS* に接続しているすべての作業エージェントが受信することができる．これにより，共通ライブラリの利用者は情報を迅速に把握することができる上，送信する側は送信相手を意識することなく適切なインタラクションを行うことができる．

シナリオ 3: 問いあわせの送信　シナリオ 2 と同様，ある共通ライブラリに対する質問はすべての開発者に対して送信され，その回答はその共通ライブラリの利用者だけに確実に届けられる．これらの質問/回答はチャネルに接続された作業エージェント間で共有されており，従って重複した質問などの無駄なインタラクションを排除できる．

2.3　モデルに基づいた開発環境

本節では，2.2 節で提案したモデルに基づいたソフトウェア開発支援環境について述べる．まず支援環境の概要について述べ，次にその実装方法を示す．

2.3.1　概要

モデルに基づいた支援環境は以下のような機能を提供する．

- 作業エージェント間のインタラクション実行
  ネットワーク環境下で，記述された作業エージェントやチャネルが実際に機能することができる．
- 作業エージェントの実行支援
  作業エージェントで行われる作業が計算機によって実行可能であった場合に，それを自動的に実行する．
- ソフトウェア開発者に対する作業誘導
  作業エージェントで行われる作業が人手によるものであった場合，作業に必要なツールの実行どを行う．また，作業エージェントのもつ情報を統合して現在の作業スケジュールを示し，次に行える作業の誘導を行う．

2.3.2 開発環境の構成

図 2.5 は今回設計した支援環境の構成を表したものである.

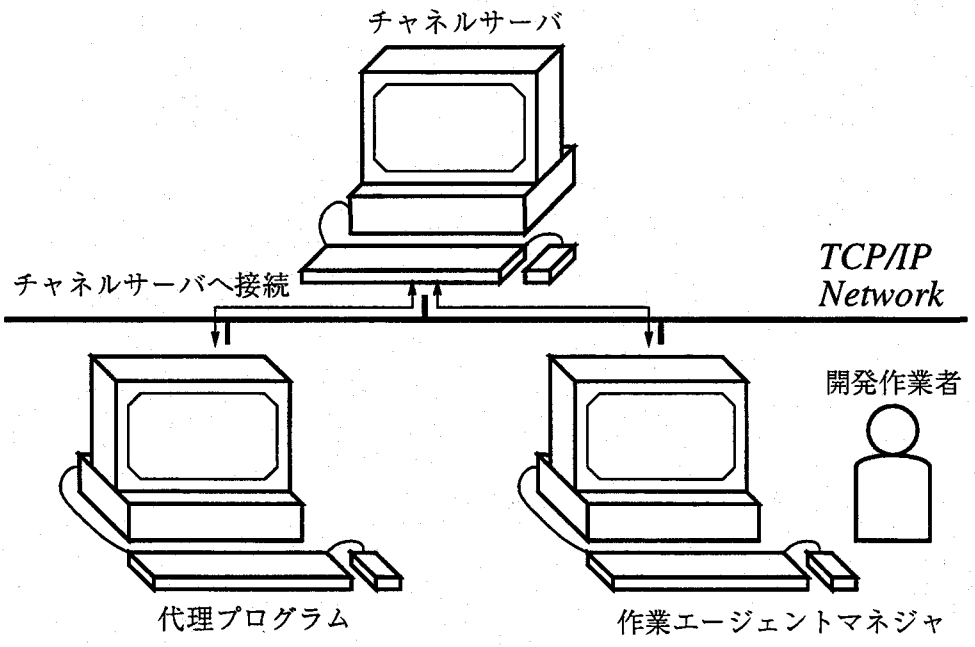

図 2.5: 支援環境の構成

本支援環境は比較的小規模のネットワーク上で用いらえることを想定しており，次のような部分から構成される.

- 作業エージェント代理プログラム
- チャネルサーバ
- 作業エージェントマネジャ

各作業エージェントはその代理プログラムによって構築される. 代理プログラムは作業エージェントの定義を読み記述された動作を代行する. すべてのチャネルはチャネルサーバによって制御される. 代理プログラムとチャネルサーバはネットワーク越しに TCP/IP 接続されている.

作業エージェントマネジャは複数の代理プログラムを統轄し，ソフトウェア開発者に対するユーザインターフェースとなる. 作業エージェントマネジャは現在の代理プログラムの状況を示し，ツール起動などの作業誘導を行う.

作業エージェントマネジャに管理されない代理プログラムは単体で動作を行う．生成物管理ツールやプロジェクト管理ツール等のような外部プログラムの実行，事前に定義された作業の実行どだけを行う作業エージェントはこのような形で実現される．

チャネルサーバは代理プログラムとの接続を管理し，受けとった情報を適切な代理プログラムへ送出する．チャネルサーバを通過するインタラクションの内容は履歴として保存する．保存された履歴を用いて，開発作業の進捗状況の分析等を行うことができる．

2.3.3 開発環境の試作

本システムの試作を行った．試作したシステムは BSD UNIX 上で C 言語を用いて実装された．代理プログラムは作業エージェントの定義を読みこみ，チャネルサーバと TCP/IP で接続し，メッセージの送受信を行うことができる．

試作した代理プログラムは単一のファイルや文字列といった単純なメッセージの送受信を行う．作業エージェントマネジャの実装を行っていないため，代理プログラムが自分自身で端末型ユーザインターフェースを持ち，ソフトウェア開発作業者からの操作を受けつける．代理プログラムがチャネルからファイルを受信した場合にはそれをあらかじめ定められた場所に保存し，ファイルの到着を開発作業者へ知らせる．文字列を受信した場合にはそれをそのまま画面へ出力する．

試作したチャネルサーバは，ソケット処理エンジン，チャネル管理テーブル，ソケット管理テーブルなどから構成されており，代理サーバからの接続を受けつけ，メッセージの送受信を管理，記録する．

試作した代理プログラムとチャネルサーバを 2.2 節で記述したアプリケーション開発の例題に適用しその動作を確認した．

2.4 関連研究との比較

多くの既存のプロセス中心型開発環境は，2.1 で述べたようなインタラクションをパラメータの受け渡しや単純なメッセージ送受信として捉えている．このような環境では，一般的に実際にインタラクションを行う対象（例えば，誰がインタラクションの相手なのか）を具体的に把握する必要があるか，プロセス記述の抽象度が高いためのインタラクション自身を表現することが難しくなっている．

一方，既に提案，運用されているコミュニケーション支援ツールを導入してインタラクションの支援を行うとする場合，次のような問題が無視できない．

- インターネットで用いられている電子メールを用いるツールの場合，届けられるべき相手先に確実に届けられることを保証することが難しい．更に，利用者間の会話等に電子メールを用いるのはあまり適切ではない．

- talk や phone 等の対話ツールを用いることにより，利用者間の会話等には効果的な支援が行える．しかし，例えば1つのファイルのような比較的大きな情報を転送する場合にはあまり適さない．また，個々の対話が独立して行われるために，インタラクションの全体的な制御や記録を行うことは難しい．

- [20, 22, 23, 51] 等のグループウェアを分散データベースとして用いることによっても効果的な支援が行える．しかし，これらの研究では比較的固定した対話経路に対する支援を行うものが多く，ソフトウェア開発作業のように作業の結果や進捗状況によって動的に構成が変化する物には不適切であると考える．また，対象が人間同士のコミュニケーションを想定しているものが多い．

本研究により，インタラクションをチャネルとして抽象化することにより，インタラクションを具体的相手先を意識することなく行うことが可能となっている．また，チャネルや作業エージェントの定義により，インタラクションを確実に実行することができる．

2.5 まとめ

本節では，ソフトウェア開発環境におけるインタラクションを表現するプロセス中心型インタラクションモデルの提案を行った．本モデルは作業エージェントとチャネルと呼ばれる2つの要素から構成される．これにより，インタラクションはチャネルを媒体して作業エージェントによって生成された一連のイベントとして定義される．本モデルを用いることにより，開発環境中のインタラクションが明確に特定され記述でき，プロセスを理解するために有用であることがわかった．

また，本節ではモデルに基づいた開発支援環境の設計を行い，その試作を行った．本支援環境はモデルにて定義されたインタラクションを実際に動作させることができ，自動実行支援，ツール起動や作業誘導を行うことができる．試作したシステムを用いて，文字列

やファイルの送受信を行って実際に開発作業者間でのインタラクションが行えることを確認した.

今後の課題として，更に実験や実装を重ねることによってモデルおよび支援環境の評価や，更に高度なインタラクションモデルによるより進んだ作業者間のコミュニケーション支援を行う枠組について考察を行うことが挙げられる．また，支援環境の実装にあたっては，記述された内容の依存関係を追跡することによって，記述の一貫性を確認しそれをわかりやすく表示することが必要であると考えられる.

付録: 作業エージェントおよびチャネルの記述言語

本節では，今回試作したシステムにおいて，作業エージェントおよびチャネルを記述する際に用いられる言語の言語仕様を EBNF にて示す．また，各言語要素の意味について簡単に説明する．なお，記述全体は以下に述べる言語仕様における非終端記号 InteractionDescrs にて表される.

言語仕様

```
InteractionDescrs ::- {InteractionDescr}.

InteractionDescr ::- "define" WorkAgentOrChannel.

WorkAgentOrChannel ::- WorkAgentDescr | ChannelDescr.


WorkAgentDescr ::-

        "workagent" WorkAgentName WorkAgentDescrBody.

ChannelDescr ::- "channel" ChannelName ChannelDescrBody.

WorkAgentName ::- NAME.

ChannelName ::- NAME.

WorkAgentBody ::- "begin" WorkAgentElements "end".

ChannelBody ::- "begin" ChannelElements "end".

WorkAgentElements ::-
```

```
        AttributeElement [VarsElement]
        [InitActionElement] [PrimitiveElement]
        ActionElement.
ChannelElements ::- AttributeElement RestrictionElement
                    InfoTypeElement.

AttributeElement ::- "attribute" AttributeContents.
AttributeContents ::-
        "begin" AttributeContent
        {AttributeContent} "end" | AttributeContent.
AttributeContent ::- AttributeName ":" AttributeValue ";".
AttributeName ::- NAME.
AttributeValue ::- NAME.

VarsElement ::- "vars" VarsContents.
VarsContents ::- "begin" VarsContent {VarsContent} "end" |
                 VarsContent.
VarsContent ::- VarsName ":" VarsType ";".
VarsName ::- NAME.
VarsType ::- BuiltinType | InfoTypeType | ArrayType.
BuiltinType ::- "integer" | "number" | "string" |
                "file" | "boolean".
ArrayType ::- "array of" ArrayComposeType.
ArrayComposeType ::- BuiltinType | InfoTypeType.
```

```
InitActionElement ::- "initaction" InitActionContents.
InitActionContents ::- "begin" InitActionContent
                       {InitActionContent} "end" |
                       InitActionContent.
InitActionContent ::- ActionBody {ActionBody}.

PrimitiveElement ::- "primitive" PrimitiveContents.
PrimitiveContents ::- "begin" PrimitiveContent
                      {PrimitiveContent} "end" |
                      PrimitiveContent.
PrimitiveContent ::- "add" PrimitiveAddSymbol
                     {"," PrimitiveAddSymbol}.
PrimitiveAddSymbol ::- NAME.

ActionElement ::- "action" ActionContent {ActionContent}.
ActionContent ::- "channel" ActionChannelName "begin"
                  ActionWithTag "end".
ActionChannelName ::- NAME.
ActionWithTag ::- ActionTag ":" ActionBody.
ActionTag ::- InfoTypeType.
ActionBody ::- ActionStatement {";" ActionStatement} ";".
ActionStatements ::- "begin" ActionBody "end"
ActionStatement ::- ActionBasic | ActionConditional
ActionConditional ::-
        "if" "(" Expression ")" "{" ActionTBody "}"
```

```
        "else" "{" ActionBody "}" |
        "if" "(" Expression ")" "{" ActionBody "}"
        "while" "(" Expression ")" "{" ActionBody "}"
ActionTBody ::- ActionTStatement {";" ActionTStatement} ";".
ActionTStatement ::- ActionBasic | ActionTConditional
ActionTConditional ::-
        "if" "(" Expression ")" "{" ActionTBody "}"
        "else" "{" ActionTBody "}" |
        "while" "(" Expression ")" "{" ActionTBody "}"
ActionBasic ::- ActionEquation | ActionFuncall |
                ActionStatements | ActionNull.
ActionNull ::-.
ActionEquation ::- ActionEquLeft "=" Expression.
ActionEquLeft ::- VarsName.
ActionFuncall ::- FuncName "(" [Expressions] ")".
FuncName ::- NAME.
Expressions ::- Expression {"," Expression}.
Expression ::- SimpleExpr [RelationOpr SimpleExpr].
SimpleExpr ::- [Flag] Term {AddOpr Term}.
Term ::- Factor {MultOpr Factor}.
Factor ::- VarsName | Constant |
          "(" Expression ")" | ActionFuncall.
RelationOpr ::- "==" | "!=" | ">" | ">=" | "<=" | "<".
AddOpr ::- "+" | "-" | "||".
MultOpr ::- "*" | "/" | "&&".
```

```
Flag ::- "+" | "-".

Constant ::- SimpleNumber | String | "True" | "False".

SimpleNumber ::- NUM {NUM}.

String ::- """ StringContent """.

StringContent ::-

        arbitally character except double-quotation.


RestrictionElement ::- "restriction" RestrictionContents.

RestrictionContents ::-

        [InclosureContents] [ExclosureContents].

InclosureContents ::- "in:" InclosureElement

                      {"," InclosureElement}.

InclosureElement ::- "(" AttributeSymbols ")".

ExclosureContents ::- "out:" ExclosureElement

                      {"," ExclosureElement}.

ExclosureElement ::- "(" AttributeSymbols ")".

AttributeSymbols ::-

        AttributeSymbol {"," AttributeSymbol}.

AttributeSymbol ::- NAME.


InfoTypeElement ::- "infotype" InfoTypeContents.

InfoTypeContents ::- "begin" InfoTypeContent

                     {InfoTypeContent} "end" |

                     InfoTypeContent.

InfoTypeContent ::-
```

```
        InfoTypeType ":" "(" FieldSymbols ")" ";".

InfoTypeType ::- NAME.

FieldSymbols ::- FieldSymbol {"," FieldSymbol}.

FieldSymbol ::- FieldName ":" FieldType.

FieldName ::- NAME.

FieldType ::- VarsType.


NAME ::- ALPH { ALPH | NUM }.

ALPH ::- "a" | ... | "z" | "A" | ... | "Z" | "_".

NUM ::- "0" | ... "9".
```

言語要素の意味

作業エージェント （WorkAgentDescr）

- 属性（AttributeElement）
  実行時には，記述されるすべての作業エージェントにおいて，同じ属性名が定義されているか，定義された順番が同一であるかどうかが確認される．この定義された順番は，チャネルの記述における閉包の定義を記述する際に，属性値の組に書かれる順番と対応付けられる．

- 変数定義（VarsElement）
  作業内容の記述の際に，一時的な情報を蓄えておくための変数は宣言されなければならない．変数は定義された作業エージェント内でのみ有効となる．変数には型があり，組み込みの型（整数，実数，文字列，ファイル，論理値）か，若しくはチャネルにおける情報形式の型を用いることができる．変数はその初期値として数字ならば0，文字列やファイルならば空文字列，論理値ならば偽を取る．

- 初期化動作（InitActionElement）
  初期化動作は作業エージェントの動作開始時に一度だけ評価される．この時点では，作業エージェントはどのチャネルにも所属していないため，他の作業エージェントか

らの情報を受けとることはできないが，明示的にチャネルに所属することによって，チャネルへ情報を送ることは可能である．初期化動作として記述された動作を終了する際には，各チャネルに記述された内閉包の条件が調べられる．もし，条件に該当するチャネルに接続していなかった場合には，初期化動作として該当するチャネルへの接続動作が自動的に行なわれる．

- 基本動作宣言（PrimitiveElement）
  宣言される動作は，実行系においてあらかじめその動作内容が定義されていなければならない．基本動作宣言に記述されているが，その内容が定義されていない動作を実行した場合には，実行時にエラーとなり処理は行われない．

- 作業内容（ActionElement）
  前条件として，現時点ではチャネル名と情報形式の組で指定することによって特定のチャネルに特定の情報が流れた場合を指定することができる．作業内容の記述中では，変数名 info にチャネルより流れてきた情報の内容が納められている．作業内容自体の記述は一般に見られる簡単な手続き型言語と同様に，数値演算や制御構造を記述することができる．アクティビティは関数呼び出しの形で記述される．アクティビティとしては，組み込みのアクティビティである「チャネルへの接続（join）」「チャネル接続の切断（leave）」「チャネルへの情報送出（send）」の他，基本動作宣言にて宣言された動作がある．send を用いる際には，引数で明示しない限り，その記述の前条件で指定されるチャネルあるいは情報形式の情報を送出すると解釈される．

チャネル （ChannelDescr）

- 属性（AttributeElement）
  作業エージェントの時と同様に，同じ属性名が定義されているかが動作時に確認される．

- 閉包（RestrictionElement）
  属性値の組として並べた順番と，作業エージェントにおける属性の定義の順番は一致しているとして，閉包の指定は解釈される．つまり，作業エージェントの記述の際，A, B という属性をこの順に定義した場合には，閉包の定義は 2 つの属性値の組（x, y）として記述され，それぞれ属性 A として x, 属性 B として y を指定したとみな

す．作業エージェントがチャネルへ明示的に接続する際，あるいは接続を切断する際には，該当チャネルの外閉包，内閉包として指定された値を調べる．これにより，作業エージェントがチャネルへの接続が許されるか，あるいはチャネルからの切断が許されるかの判断が実行時に行われる．

- 情報形式（InfoTypeElement）
  フィールドの型は，作業エージェントの変数定義における組み込みの型と同一の物が用いられる．チャネルが動作する際には，作業エージェントから送られてくる情報が事前に定義されている情報形式と一致しているか確認される．もし一致しない場合には，その情報は破棄される．

第3章　既存環境と共存する分散環境を想定したプロセス実行のための環境

ソフトウェア開発作業を効率良く行い，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 [9] をあらかじめ記述し，それに基づいた開発作業を行うようになってきている [14, 36]．ソフトウェアプロセスを表現するために，ソフトウェアプロセスのモデル化に関する研究が広く行われている．しかし，既存の多くのプロセスモデルは「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, 16, 17, 44]，それらに基づいたプロセス中心型開発環境もまた，開発作業に着目した物となっている [3, 10, 47]．

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，ネットワークを活用したプログラム部品の開発と，プログラム部品の組み合せによるソフトウェア開発等，近年さかんに行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」に着目した開発作業となっている．このような開発形態は従来とは大きく異なっているため，従来のソフトウェアプロセスモデルや，そのようなモデルに基づいたプロセス中心型開発環境は，そのまま適用できない可能性が高い．

このような開発形態におけるソフトウェアプロセスを考える際には，開発ごとに作成される各種のプロダクトをその要素の中心にする必要があることは容易に想像できる．例えば，PCTE[48] に代表されるプロダクト指向の CASE 環境を用いて，開発環境それ自身はもちろん，その環境で行われるソフトウェアプロセスを記述し，その実行方法，実行状況の把握等を行う開発環境を構築する方法が考えられる．しかし，このような開発環境は従来行ってきた開発環境を置きかえる物であり，導入時に大きな環境の変化を伴ない，かつ，利用者に対してある種の定型作業を強いるものとなってしまうと考えられる．

本節では，このような近年の開発形態に即した開発環境の構築に関する研究について述べる．[26, 25, 27, 34]．本研究で提案する環境は，開発時に生成されるプロダクトや，開発環境中のさまざまな資源を構成要素とするソフトウェアプロセスモデルを用いることによって，プロセスの記述，プロセスの実行，実行された作業の計測と，計測したデータに

基づく進捗状況の把握を行うことを目的としている．

本節ではまず，本論文で提案するオブジェクト中心型ソフトウェアプロセスのモデル化手法である MonoProcess について述べる．MonoProcess では上記で述べた目的を達成するために，開発環境中のさまざまな要素を構成要素として開発環境全体を表現することができる．また，記述されたオブジェクトを用いて，ソフトウェアプロセスを表現し，かつ管理するための枠組を提供する．

3.1 プロセスモデル MonoProcess

本節では，本論文にて提案するソフトウェアプロセスのモデル化手法について説明する．最初に，MonoProcess 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．また，MonoProcess のもつ機能についても紹介する．

3.1.1 概要

MonoProcess 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」を単位としたモデル化を行う．オブジェクトは属性とメソドから構成され，属性によってオブジェクトのもつ特性を，メソドによってオブジェクトに適用する操作を定義する．MonoProcess は任意のオブジェクトを集合化する機能を提供しており，これによって複数のオブジェクトを 1 つのオブジェクトとして扱える．オブジェクトの継承機構を用いることにより，複数のオブジェクト間で情報の共有を行える．また，属性の参照等のオブジェクトへのメッセージはオブジェクトの操作履歴として自動的に保存される．これらの履歴は作業状況の把握等を行う際に利用される．

3.1.2 MonoProcess 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 3.1 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．

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

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

図 3.1: Object 記述例

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3.1では5つの属性が定義されており，@Owner, @Type, @Filename, @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もつリスト

例えば，図3.1で定義されている属性 @Owner とそれに対する属性値 (.Person.Matsushita) は「このオブジェクトの管理者が .Person.Matsushita というラベル名で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3.1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 ではメソド関数がもつべき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．なお，オブジェクトの記述を解釈する実行系は，これらのメソド記述言語を解釈するためのアプリケーションインターフェイスをもつ．新たなメソド記述言語を追加する場合には，それを解釈するモジュールを利用者が追加することによって行う．図3.1では，現在試作中である Perl 風の簡単な記述言語を解釈するモジュールを用いてメソド内の記述を行っている．

3.1.3 MonoProcess オブジェクトを用いたソフトウェアプロセスの記述

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 のもつラベルを接頭語として持ち，O のもつ属性やメソドの部分集合をもつものとする．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．本研究では，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．

簡単な例を用いて部分オブジェクトと状態オブジェクトを説明する．まず，.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，を表す）を仮定する．これらの各オブジェクトは同一の属性ラベル @FINISHED を持ち，この属性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．

まず，.SPEC, .CODE, .TEST それぞれのオブジェクトから @FINISHED だけを抜きだした物を，それぞれ部分オブジェクトとして定義する．定義した部分オブジェクトをそれぞれ .SPEC.FIN, .CODE.FIN, .TEST.FIN とする．そして，これらのオブジェクトで表現される開発の進行状況を示す状態オブジェクト .Dev.Status を，上記 3 つの部分オブジェクトの集合として定義する．

このとき，作業が進行するに従って .Dev.Status は次のような状態の変化を行う（図 3.2）．

1. まず，開発開始時には .SAMPLEOBJ のもつ 3 つの状態オブジェクトがもつ @FINISHED 属性がすべて “false” となっている．
2. 仕様書の作成が終了した時点で，.SPEC.FIN のもつ属性 @FINISHED が “true” へ変化する．
3. コーディングの作業が終了すると，.CODE.FIN のもつ属性 @FINISHED が “true” へと変化する．つまり，.TEST.FIN だけが “false” の状態である．
4. 作業が終了すると，すべてが “true” に変化した状態となる．

上記のような .Dev.Status の状態遷移は，開発の進行状況と直接対応するものとなって

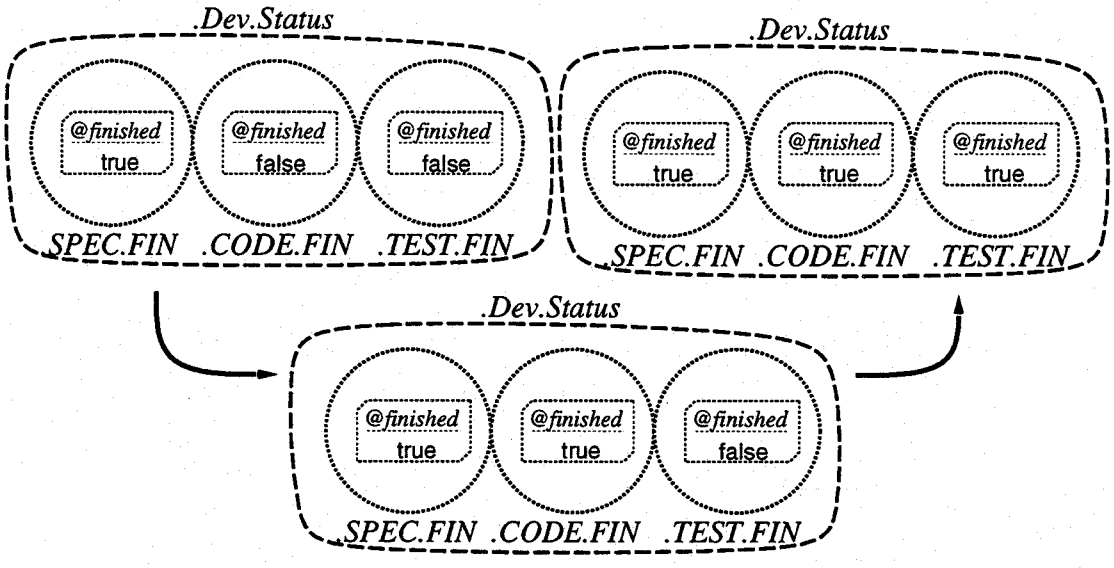

図 3.2: 状態オブジェクトの遷移

いる．MonoProcess では，必要に応じて各状況に対応した状態オブジェクトを定義し，その遷移系列を定義することによってソフトウェアプロセスを表現することができる．

3.1.4 MonoProcess オブジェクトの機能

MonoProcess は，先に述べた開発環境の構成要素のオブジェクトによるモデル化とそれを用いたソフトウェアプロセスの表現の他に，プロジェクト管理，ソフトウェア開発，開発者の協調作業を支援するためにさまざまな付加機能を提供する．以下，MonoProcess の提供する付加機能について説明する．

オブジェクトの参照範囲とアクセス制御

一般的には，すべての各オブジェクトは他のオブジェクトから参照され，また他のオブジェクトを参照することができる．しかし，各オブジェクトはそれ自身の他からの参照範囲を自分自身の属性として定義することができる．実際には，この参照範囲は他のオブジェクトからのアクセス制御として定義している．従って，以下ではオブジェクトのアクセス制御についてのみ述べる．

アクセス制御は，オブジェクトが含まれるグループの指定と，そのグループに対する制

御内容として示される．

- グループによる範囲の指定

  グループはグループの名前によって特定されるオブジェクトの集合である．各オブジェクトはあらかじめ決められたグループに所属できる．所属したグループの名前は，@GROUP 属性にて表される．@GROUP オブジェクトを持っていないか，持っていても値が定義されていないオブジェクトはどのグループにも属していないものとする．

- 範囲/範囲外から行える操作の指定

  @ACCESS 属性によって，オブジェクトに対して行える/行えない操作を定義する．操作には 4 つの種類があり，それぞれ「属性値/メソドの参照」「属性値/メソド関数の変更」「メソドの実行」「継承の許可」である．この 4 種類の操作について,「同一のグループに所属しているオブジェクトから」および「同じグループに所属していないオブジェクトから」行われた際に許可するかどうかを @ACCESS に保存する．これにより，参照範囲の制限は属性値/メソドの参照の可否によって行われる．

MonoProcess が一般的なオブジェクト指向の考え方に従っているならば，これらの機能はクラス構造やクラス間の関連付け等によって実現されるであろう．しかし，時間の経過と共にその構造が変化し続けることが前提となるソフトウェアプロセスのモデル化にそのようなオブジェクト指向の方法をそのまま適用するのは困難ではないかと考えている．MonoProcess では，ソフトウェア開発環境中の要素をオブジェクトとして定義することができる．このため，開発環境の変化に対応して，自由にその構成を変更することが可能である．また，オブジェクトは単純な構造を用いており，より複雑な要素についても複数のオブジェクト等の組みあわせとして表現することが可能となっている．これにより，MonoProcess を用いることにより，複雑でかつ多様な形態を取る実際の開発環境をより正確に記述できると考えられる．

オブジェクトへの操作履歴

属性値の参照等，オブジェクトに対する操作は，対象となるオブジェクトに対してメッセージを送り必要な処理を要求することによって行われる．MonoProcess では，すべてのオブジェクトに対する操作は，履歴を保持する属性に記録される．一般にすべての操作に対する履歴の収集は非常に膨大かつ頻繁に行われる物となるが，MonoProcess では，各オ

ブジェクトの粒度がある程度大きい単位(ファイル等)であり，オブジェクトのもつ属性，メソドを単位とした履歴の収集を行うため，このような履歴の収集は現実的であると考えている．履歴の収集は，オブジェクトに対する変更操作や，ユーザ側で動作する履歴収集等を目的としたエージェント等によって行われる．

属性およびメソドに対する操作の履歴として，以下のような情報が記録される．

- 属性に対する操作

  操作したオブジェクト，操作した時間，操作の内容．

- メソドに対する操作

  操作を行ったオブジェクト，操作の開始時間，操作の終了時間，実行結果の戻り値．

また，これらの履歴は自動的に該当する属性へ記録される．すなわち，ある属性 @X の値の参照を行う際には，暗示的に @X.HISTORY という属性に対しての参照履歴の追加操作を伴ない，あるメソド &Y を実行する際には，その操作内容が属性 &Y.HISTORY に追加される．

オブジェクトの生成と継承

オブジェクトの生成は .OBJECT というあらかじめ定義されたオブジェクトの雛型か，既に存在しているオブジェクトからの属性/メソドの継承によって行われる．すべてのオブジェクト生成はオブジェクトからの継承と等価であるため，以下の説明ではオブジェクトの継承に着目して説明する．

既に存在しているオブジェクト .A から，新たにオブジェクト .B を継承によって作成する場合，まず .B を .A の複製として作成する．このときに，.A にて設定されたアクセス制限によって，継承を禁止された属性とメソドについては省かれる．次に，.B のもつ属性とメソドに対して新たに追加および変更する必要があればその操作を行う．つまり，MonoProcess ではオブジェクトの継承は既存のオブジェクトの複製を作成することによって行う．

なお，オブジェクトの継承は，オブジェクトの雛型で定義されている &FORK メソドの実行によって行われる．このメソドは引数として新しく生成されるオブジェクトで用いられる追加定義の情報を取る．また，&FORK を再定義することによって，あるオブジェクトに固有の継承方法を定義することができる．

3.2 MonoProcess による記述例

ここでは，いくつかの MonoProcess 記述の例を用いて，我々の目標がどのようにして実現されているかを示す．

3.2.1 オブジェクト指向分析を用いた記述

あるソフトウェアの開発を行なう際に利用される開発環境に対するオブジェクト指向分析の結果，以下のような結論になったとする．

> 文書やソースコードはオブジェクトとして考える．オブジェクト間の関連(包含関係等)はオブジェクトの継承や何らかの構造を用いて定義する．ソフトウェアツールやコンピュータもオブジェクトとする．

MonoProcess では,「システム全体の仕様書」などの文書,「コンパイラ」などのツール等は図 3.3 のように記述される．仕様書を表わすオブジェクト (.Doc.Specification) では，文書の形式や文書の場所を属性として表し，文書を閲覧する作業がメソドとして定義されている．コンパイラを表わすオブジェクト (.Tool.Compiler) では，対象言語を表わす属性やツールの場所を属性として表しており，ツールを起動するためのメソドが定義されている．

このように，MonoProcess ではオブジェクト指向分析で抽出されたオブジェクトをそのままの形でオブジェクトとして表現することができる．

3.2.2 さまざまな粒度を持つオブジェクトの記述

この例 (図 3.4) では，1つ1つのソースコードがオブジェクトとして定義されており (.SRC1 and .SRC2)，全体として 1 つのモジュール (.MODULESRC) を構成しているとする．

.SRC1 および .SRC2 と .MODULESRC は共通のラベルが付けられた &MAKEOBJ というメソドを持っている．しかしながら，両者の振舞いは異なっている．.SRC1&MAKEOBJ は単純に自分自身のコンパイル作業を行なうだけだが，.MODULESRC&MAKEOBJ はモジュールに含まれている各ソースコードをそれぞれコンパイルした後，リンク作業を行なってモジュールの生成を行なう．このように，MonoProcess では，同じラベルが付けられていても，オブ

```
Object .Doc.Specification def
        Attribute @DocType "plaintext";
        Attribute @Location "host/path/to/document";
        Method &View def
                $tool = .caller&GetViewer(@DocType);
                if ($tool) {
                        invoke($tool, @Location);
                } else {
                        @Location;
                }
        endMethod
endObject

Object .Tool.Compiler def
        Attribute @Language "C, C++";
        Attribute @Localtion "arch:/path/to/cc";
        Method &do def
                invoke(@Location, $args);
        endMethod
endObject
```

図 3.3: OOA の例

```
Object .SRC1 def
        ...
        Attribute @Location "host/path/to/src1";
        Method &MAKEOBJ def
                invoke(.COMPILER@Location, @CompileFlags,
.SRC1.OBJ@Components)
        endMethod
        ...
endObject

Object .SRC2 def
        ...
        Attribute @Components "path/to/src2";
        Method &MAKEOBJ def
                invoke(.COMPILER@Location, @CompileFlags,
.SRC2.OBJ@Components)
        endMethod
        ...
endObject

Object .MODULESRC def
        ...
        Attribute @Components (.SRC1 .SRC2);
        Attribute @Location "host/path/to/module";
        Method &MAKEOBJ def
                foreach $Component (@Components) {
                        $Component&MAKEOBJ;
                }
                invoke(.LINKER@Location, @Components, @Location);
        endMethod
        ...
endObject
```

図 3.4: 異なる粒度におけるオブジェクトの例

ジェクトごとに違う動作を定義することができる．これによって，それぞれの粒度に応じた操作を独自に定義できる．

もし仮に，開発者が .SRC1 を 2 つのソースコードに分割することにしたとする．この場合には，MonoProcess では特別なメソドである.SRC1&FORK を用いて，.SRC1 から .SRC1.MAIN と .SRC1.SUB という 2 つのオブジェクトを継承によって生成する．また，メソド .SRC1&MAKEOBJ がこの 2 つを管理するように再定義する．このように，MonoProcess では開発の状況に応じて柔軟にオブジェクトを定義し直すことによって，さまざまな粒度のオブジェクトを自由に定義できる．

3.2.3 オブジェクト中の情報を抽出するための記述

この例では，オブジェクト .FOO に属性 .FOO@MAINTAINER が定義されており，このオブジェクトの担当者を表現しているとする．ある時，担当者が別の人間に変更になった (.FOO@MAINTAINER の値が変更された) とする．その後，オブジェクト .FOO に関するトラブルが発生した際に「原因と思われる事象が起きた当時，誰が担当者であったのか」を調べることを考える．

MonoProcess では，.FOO@MAINTAINER.HISTORY 属性を参照することによって，.FOO@MAINTAINER 属性の変化を記録している．属性値の変更等，オブジェクトに対する操作の履歴は保存されているため，この場合 .FOO@MAINTAINER.HISTORY を参照することによって，当時誰が担当者であったのかを調べることができる．

また，.BAR という，.FOO と同種で異なるオブジェクトがあったとする．もし，担当者の推移に関するプロセスを観測したい場合には，次のような記述によって定義することができる．

```
StatusObject .MAINTAINER_STATUS def
        PObject .FOO.MTSTATUS;
        PObject .BAR.MTSTATUS;
endObject
PObject .FOO.MTSTATUS def
        Attribute @MAINTAINER;
endObject
PObject .BAR.MTSTATUS def
        Attribute @MAINTAINER;
endObject
```

この場合，2 つの部分オブジェクト (.FOO.MTSTATUS と .BAR.MTSTATUS) を定義し，これ

を用いて状態オブジェクト .MAINTAINER_STATUS を定義している．.MAINTAINER_STATUS の状態変化を追うことによって，担当者の変化に関するプロセスを観察することができる．

3.3 MonoProcess を用いた開発管理システム

3.3.1 予備実験

MonoProcess を実装するにあたり，近年の分散開発環境下を想定した，既存の開発環境と共存するプロセスの実行環境の有効性を確認するための予備実験を行った．この予備実験のために，Web を用いた開発支援環境を構築した．ここでは，予備実験で用いた簡単な開発支援環境と実験の内容について述べる．

プロセスモデル

分散開発環境下で，直観的にその開発プロジェクトの状況を示すものとして，以下の 3 つの要素とその間の関係を考える．

- **タスク**
  ソフトウェア開発におけるひとまとまりの作業．1 つのタスクには必ず 1 人以上の開発者が割り当てられ，その関係を破線の有向辺で示す．
- **プロダクト**
  タスクの入出力となりうる成果物．タスクとの入出力関係を有向辺で示す．
- **開発者**
  タスクに割り当てられる作業者（1 人の開発者が複数のタスクに割り当てられる場合もある）．

これらの要素の状態を知ることにより，全体の進捗状況が把握できる．そして，これらの要素に対して以下のような属性値を付加することにより，プロセスモデルとして用いることにする（図 3.5）．

- **タスクに関係する属性値**
  スケジュール，作業内容，開始/終了制限，入出力プロダクト名，進捗状況，連絡事項

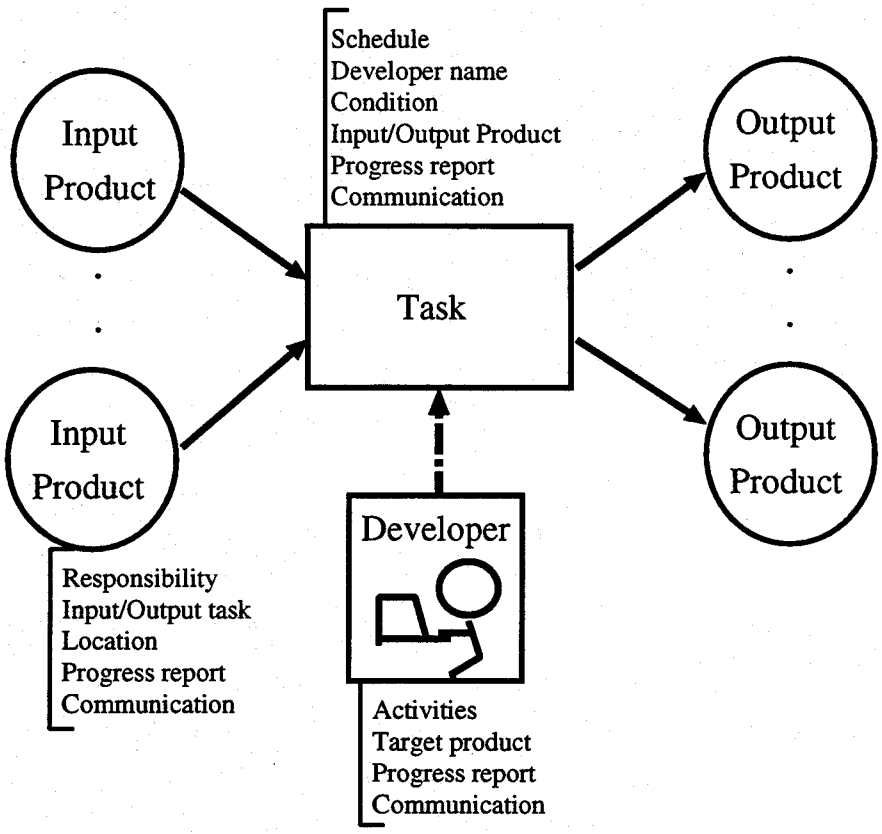
Schedule
Developer name
Condition
Input/Output Product
Progress report
Communication
Input Product
Input Product
Task
Output Product
Output Product
Developer
Responsibility
Input/Output task
Location
Progress report
Communication
Activities
Target product
Progress report
Communication

図 3.5: プロセスモデル

- **プロダクトに関係する属性値**
  責任者，プロダクトを生成/使用するタスク名，プロダクトの所在，進捗状況，連絡事項

- **開発者に関係する属性値**
  作業内容，作業の対象となるプロダクト名，進捗状況，連絡事項

Web システムの構築方法

プロセスモデルに基づくページの生成　前述のプロセスモデルの各要素に対応して，Web ページを設ける．また，全体のプロセスを表示するためのページ（プロセスページ）を設ける．具体例として，2つのタスク（ModifyCode, Test），2つのプロダクト（spec, module1.c），2 人の開発者（A, B）から成るプロセスモデルに対応したページを図 3.6 に示す．ここで，module1.c はタスクに対する入出力となるため 2 つのページを設け，後述する部品等を利用することにより内容の一貫性を保つ．各ページは HyperText Markup Language（HTML）で記述されており，各ページの参照は Web ブラウザによって行う．

上述の方法で生成したページにはプロセスモデルで記述された要素間の関係を HTML のリンクで記述する．たとえば，プロセスページから各タスクページへのリンクや，タスクページからタスクに割り当てられたプロダクトページ，および開発者ページへのリンクが存在する．

部品によるページの構成　これら 4 種類のページの HTML 記述のうち，各ページに共通な部分をまとめ，以下のような部品として提供する．

- 属性値入力部品
  文章や数値で示される属性値を入力することができる．入力された値の履歴はデータベースに保存され，表示の際に利用される．たとえば，開発者ページを作成する際，数値入力を行える部品を用いることによって，各作業ごとのテスト実行回数を入力する部分を簡単に作成することができる．

- 属性値表示部品
  属性値入力部品を用いて収集された情報を表示することができる．たとえば，タスクページを作成する際，データを集計して表を作成する部品を用いることにより，デー

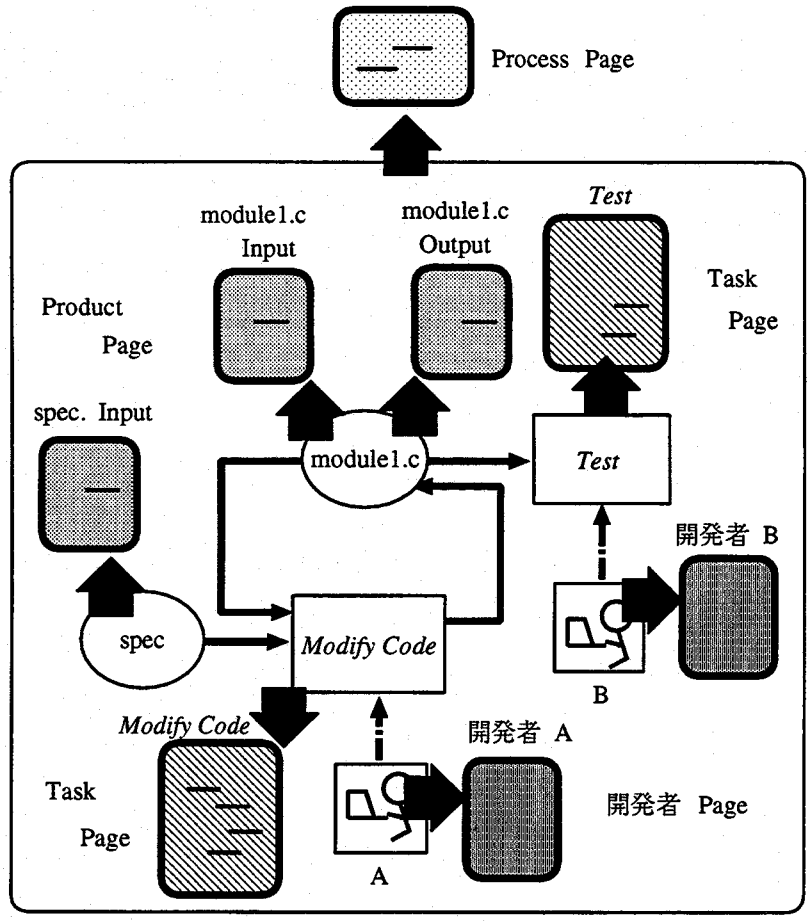
Process Page
module1.c
Input
module1.c
Output
Test
Task
Page
Product
Page
spec. Input
module1.c
Test
開発者 B
spec
Modify Code
B
Modify Code
開発者 A
Task
Page
開発者 Page
A

図 3.6: ページの生成

タベースに保存されたテスト実行回数を各作業者ごとに集計して，表の形で表示する部分を簡単に作成することができる．

試作環境を用いた実験

事前にデータベースに登録されたデータを，HTML で記述された表に整形するプログラム開発を行い，この開発プロセスの進捗管理を行った．そのために大阪大学と奈良先端科学技術大学院大学間に分散した開発環境を設定し，それをモデル化して Web システムを作成した（図3.7）．本実験では，各開発者から2つのタスクについて，作業時間，コード行数，テスト回数，発見された誤り数，文章による進捗状況報告の計5種類の進捗状況を収集，表示した．

この適用実験により，モデル化や Web システム化について次のことが分かった．

1. プロセスのモデル化を容易に行うことができた．
2. 通常用いられていた開発者の環境に影響を与えることなく，簡単にシステムの導入を行うことができた．
3. 開発者，プロダクト，タスクの追加，削除などのプロセスモデルの変更には，ページやリンクを追加や削除することで容易に対応することができた．

3.3.2 開発管理システムの試作

MonoProcess/SME は以下の 6 つより構成される (図 3.8).

- オブジェクトリポジトリ
- オブジェクトアクセスライブラリ
- メソドエンジン
- オブジェクトブラウザ
- HTML/リポジトリ変換器
- ユーザ側エージェント

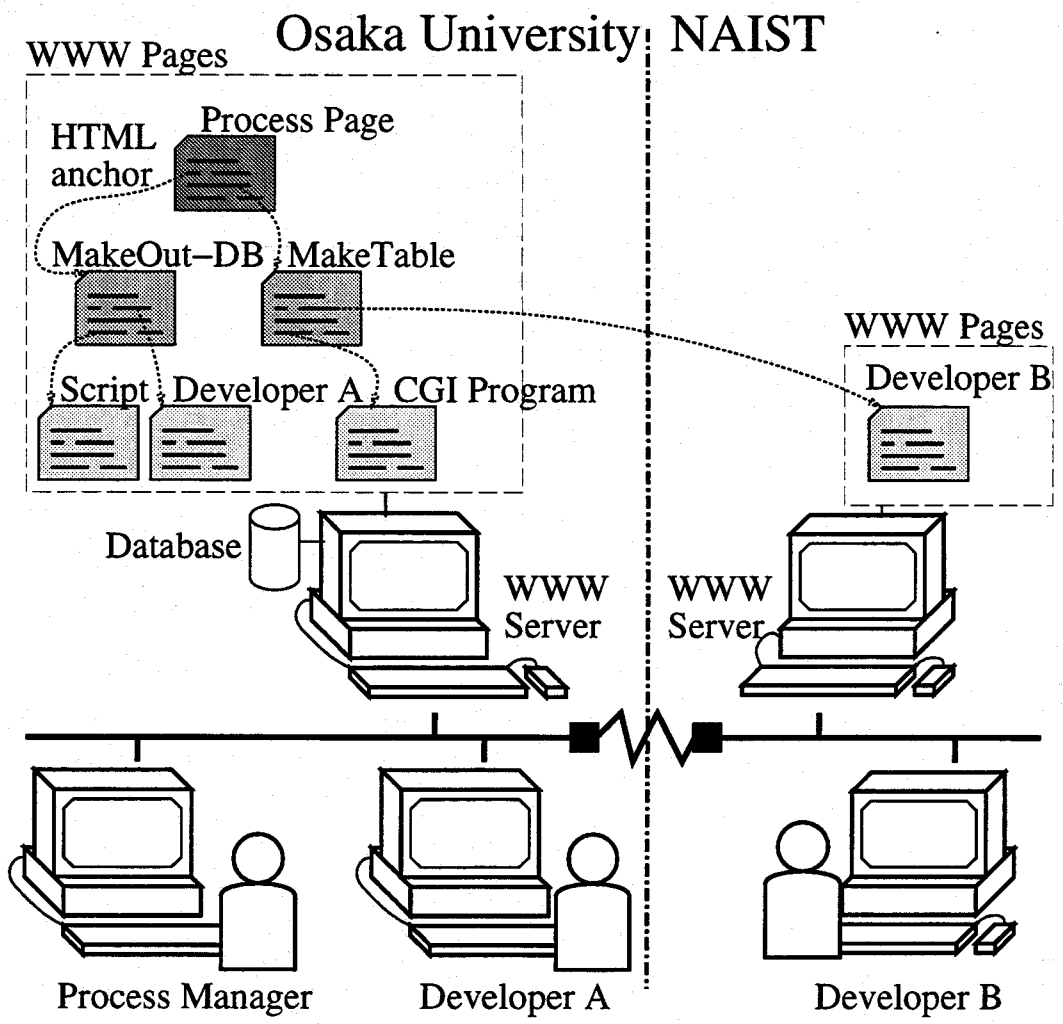
Osaka University
NAIST
WWW Pages
HTML
anchor
Process Page
MakeOut–DB
MakeTable
Script
Developer A
CGI Program
WWW Pages
Developer B
Database
WWW
Server
WWW
Server
Process Manager
Developer A
Developer B

図 3.7: 実験環境

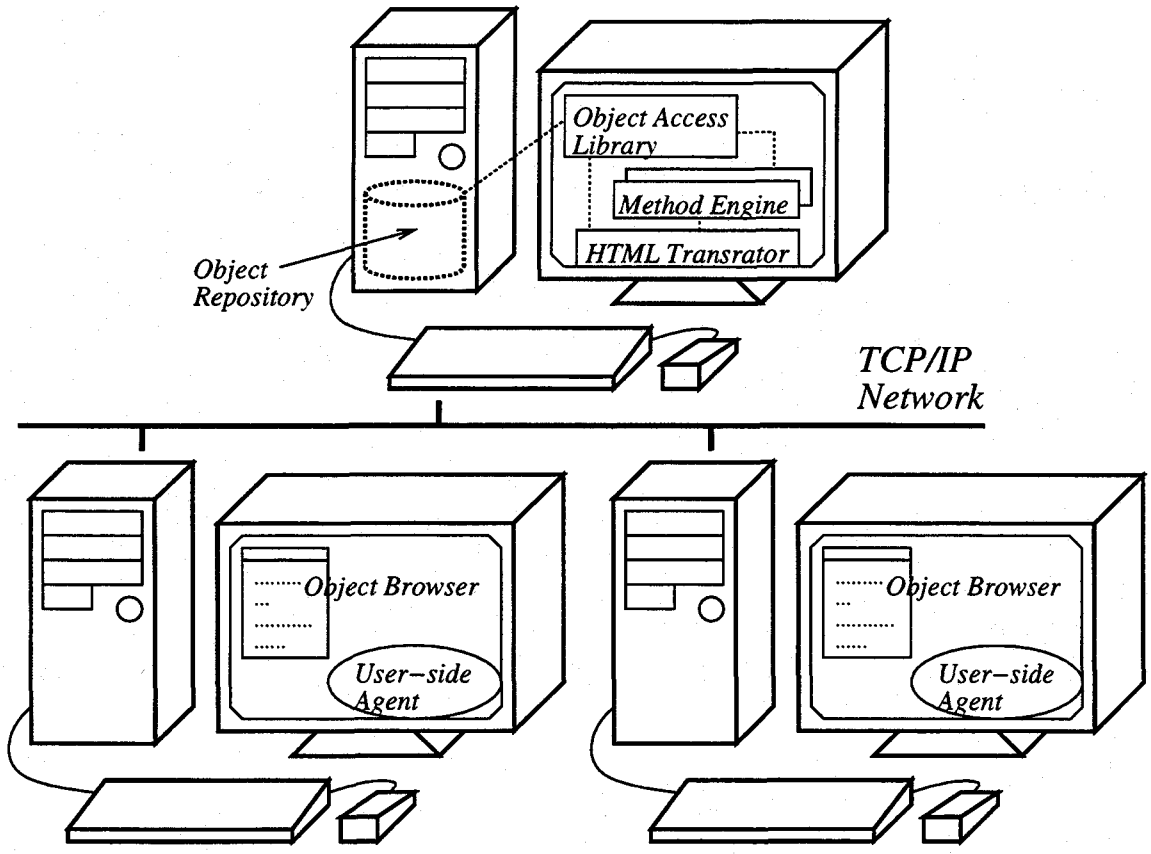

図 3.8: MonoProcess/SME の構成

オブジェクトリポジトリは定義されたオブジェクトの貯蔵庫としての役割を持つ．オブジェクトリポジトリには，オブジェクトの構造定義，属性値，メソド記述が含まれる．オブジェクトリポジトリを中心にして考えた場合，MonoProcess/SME はオブジェクト指向データベースのように見えるであろう．

オブジェクトアクセスライブラリはオブジェクトリポジトリを容易に扱うためのプログラミングインターフェースとなる．MonoProcess は複数の記述言語をメソド記述として許しているため，オブジェクトアクセスライブラリは各記述言語毎に用意される．我々は現在 Perl や UNIX shell をベースとした記述言語の実装を進めている．

メソドエンジンはメソドの解釈実行の中心的役割を果たすプログラムである．オブジェクトアクセスライブラリと同様，メソドエンジンはメソド記述言語に依存した部分であり，各メソド実行ごとに起動される．メソドエンジンはオブジェクトアクセスライブラリを用いてオブジェクトリポジトリの参照や変更を行なう．

オブジェクトブラウザは MonoProcess/SME におけるユーザインターフェースである．我々は WWW で用いられているブラウザをオブジェクトブラウザとして採用した．これによって，プラットフォームに依存しないユーザインターフェースを容易に実現することができる．オブジェクトリポジトリ等の内容をオブジェクトブラウザ側で参照できる形に変換するために，HTML[5]/リポジトリ変換器を用意する．HTML/リポジトリ変換器は WWW サーバに組みこまれて，システムの他の部分との間で情報の中継を行なう．

ユーザ側エージェントはユーザごとに用意され，各ユーザに対するバックエンド処理を担当する．ユーザ側環境の情報収集や，ツールの起動等，各ユーザに依存した処理はこのユーザ側エージェントで行なわれる．

我々は MonoProcess/SME の実現可能性と有効性を事前に確認するために，既に MonoProcess/SME の試作を行なっている．この試作では，Perl を元にした簡単な記述言語をメソド記述のために用い，これを解釈するメソドエンジンとオブジェクトアクセスライブラリを実装した．変換器は Apache WWW サーバの組み込みモジュールとして実装した．また，このプロトタイプを用いて ISPW-6 のプロセス例題 [18] を記述し動作させた．これについては 3.4 節で詳しく述べる．

3.4 ISPW-6 例題の適用

本節では，試作した MonoProcess/SME を用いた ISPW-6 の例題プロセスの記述について述べる．

3.4.1 ISPW-6 例題

ISPW-6 の例題は，ソフトウェアプロセスモデルの理解や比較を目的として注意深く設定されており，ソフトウェア変更プロセスの比較的限定された作業を定義している．なお，本例題の完全な記述は [18] にある．例題は核問題 (core problem) と拡張問題 (optional extension) から構成されているが，本論文では核問題の部分だけを取りあげる．

核問題はソフトウェア変更プロセスの比較的限定した部分に着目しており，ソフトウェアの設計，コーディング，単体テスト，局所的な管理作業を定義している．プロジェクトマネージャが変更作業計画を立てて仕事を割り振った時点から，単体テストが終了するまでの一連の作業が核問題の対象となっている (図 3.9)．

3.4.2 MonoProcess による ISPW-6 例題の記述

ISPW-6 の例題を MonoProcess で記述を行なうための手順は，大きくわけると以下のようになる．

1. 例題の記述中からオブジェクトとなるものを抽出する．
2. 抽出した対象をそれぞれオブジェクトとして記述する．
3. 着目したい点を整理して状態オブジェクトを定義する．

以下では，各項目についてどのような作業を行なったかを述べる．

オブジェクトの抽出

例題では，例題が定義している作業を次のような形で説明している．まず，例題全般に渡る話として，例題プロセスの組織的な構造について説明している．また，作業の流れを主要なタスク (作業全体は 8 つの細かい作業に分解されている) を中心にして述べている．各タスクの説明では，タスクの入出力，責任や制限について具体的に説明が行なわれている．

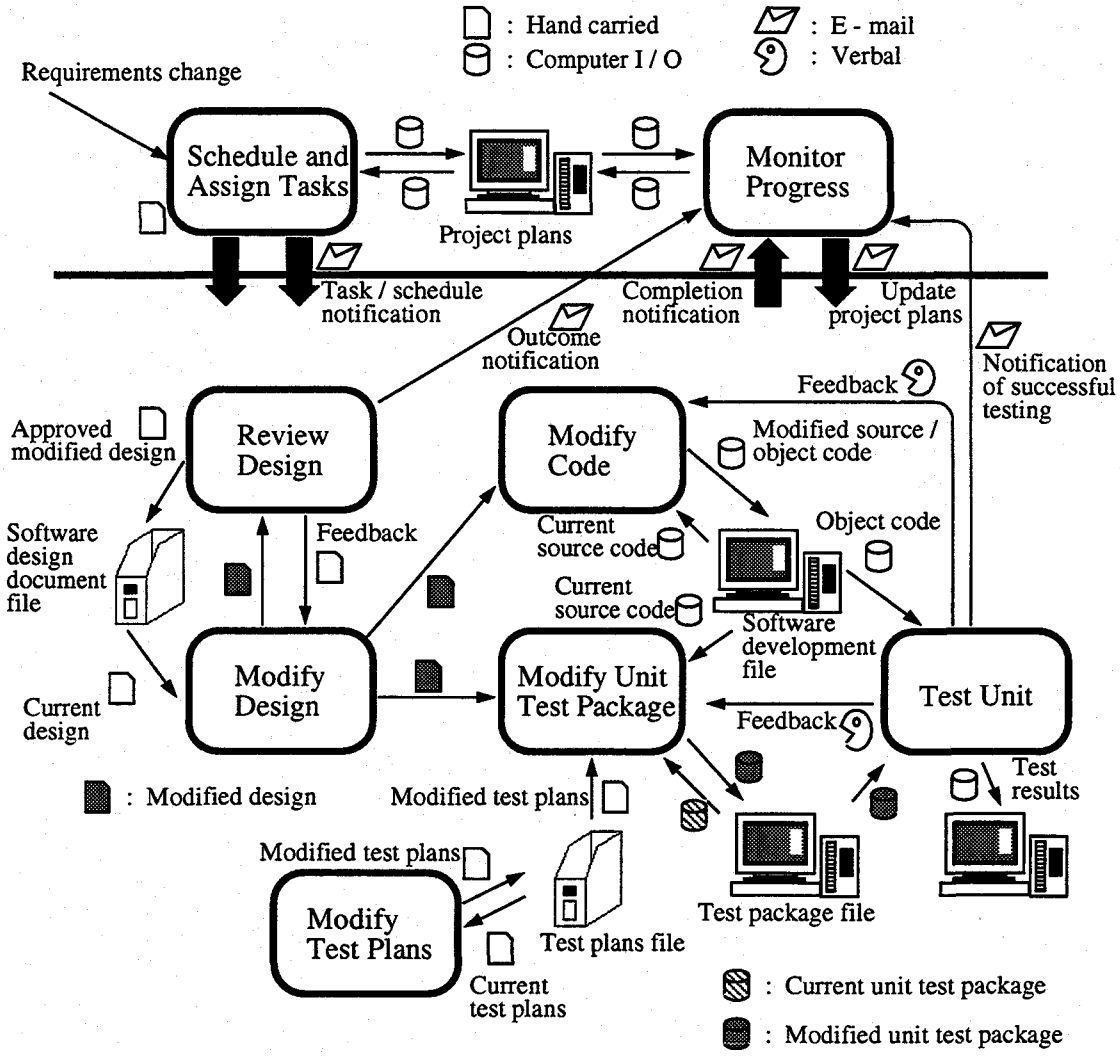

図 3.9: ISPW-6 例題 (核問題)

従来のプロセス記述では，例題によって分解されている各部分ステップそれぞれについて作業を記述するのが一般的である．この場合，作業の内容を記述するのは容易であるが，それ以外の部分は他の作業との関連を考慮しながら決定する必要がある．しかし，MonoProcessでは，この例題で述べられているプロセスの各項目をそのままオブジェクトとして定義すれば良い．すなわち，

- 各部分ステップにおいて入出力として扱われている生成物
  要求変更文書，スケジュール，タスク割り当て表，デザイン，デザインレビュー結果，ソースコード，コンパイル結果，オブジェクトコード，テスト結果，テストパッケージ，テストファイル，テスト計画書．

- 開発組織における作業者の担当する役割
  プロジェクトマネージャ，設計エンジニア，品質保証エンジニア，コンフィギュレーション管理委員会．

- 作業者それ自身

がそれぞれオブジェクトとなる．これは，例題プロセスの記述から容易に抽出することができる．MonoProcessでは実際の開発作業を記述することを目的としているため，実際には例題で記述されていない具体的な内容も決定しておく必要がある．ここでは，テストファイルが2つ存在しているとし，作業者としては1名のプロジェクトマネージャ，2名の設計エンジニア，2名の品質保証エンジニア，1名のコンフィギュレーション管理委員会の計6人を想定した．これにより，例題から23 (プロダクト13，役割4，作業者6) のオブジェクトを抽出できる．

抽出した各オブジェクトの記述

次に，抽出したオブジェクトのそれぞれについて，実際に記述を行なった．抽出したオブジェクトは3種類に分類できるため，それぞれの種類について，属性とメソドそれぞれがどう定義できるかを説明する．

入出力として扱われている生成物　例題中では，入出力として用いられる項目が1つ1つ挙げられており，それらは紙面で与えられる物と計算機上に存在している物の2種類があ

る．ここでは，入出力として用いられる文書は全て計算機上に存在しているものと仮定する．この時，例題から以下のような属性が必要であることがわかる．

- 計算機上での場所

  文書は全て計算機上にあるが，具体的に計算機上でどの場所 (ディレクトリ/ファイル) にあるかを知る必要がある．

- 直接対応する他の生成物

  例えば，修正されたソースコードは再度コンパイルされるため，あるソースコードに対応するオブジェクトコードが存在することになる．そのため，ソースコードは対応するオブジェクトコードの置き場所を知る必要がある．このような情報は属性として保持する必要がある．

また，各生成物に対して行なわれる作業は，その生成物を表すオブジェクトにおけるメソドとして定義される．よって，例題では以下のようなメソドが必要であることがわかる．

- スケジューリング/割り当て作業

  例題はまず最初に各作業のスケジューリングを行ない，作業者に作業を割り振るところから開始される．よって，これらの作業はスケジュール日程や作業計画を表現するオブジェクトに対するメソドとして表現される

- 文書の参照

  例題の実行中，各生成物は何らかの形で参照され得る．よって，生成物を参照するためのメソドが各生成物に必要となる．

- 文書の変更

  デザイン，ソースコード，単体試験パッケージ等は例題による作業の中で変更される．よって，この作業を表すメソドが必要となる．

- レビュー

  例題では，要求を受けて変更されたデザインはレビュー作業にかけられる．このため，デザインを表わすオブジェクトにはレビュー作業を表わすメソドが必要となる．

- コンパイル作業

  例題では，変更されたコードはコンパイルされ，後の単体試験に備えられる．この作業は，ソースコードを表わすオブジェクトに対するメソドとなる．

- テスト作業

  例題では，テスト作業を行ない，一定の基準を満した時点で作業全体が終了となる．この時に行なわれるテスト作業は，オブジェクトコードを表わすオブジェクトに対するメソドとなる．

- モニタリング

  例題には，それぞれの作業に対する終了通知等を管理する作業が記述されている．この作業はスケジュールが予定通り行なわれているかを見るための作業であるため，ここではスケジュール日程を表すオブジェクトに対するメソドとして表現する．

作業者の役割　例題では，例題で述べれれている作業を遂行するにあたって組織されるプロジェクトチームに関して記述が行なわれている．それぞれの役割をオブジェクトとすることによって，同じ役割を果している作業者に共通した内容を纏めて記述することができる．具体的には，例題で述べられている「各役割に割りあてられている作業で用いられる作業物」を各役割を表現するオブジェクトの属性として定義する．

作業者自身　例題では，各作業者に関する情報が具体的に詳しく述べられているわけではない．しかし，実際に開発作業を行なうにあたって，各作業者に依存した情報や作業内容をそれぞれのオブジェクトとして定義する．これにより，他のオブジェクトにおいてメソドが実行される際，各作業者に依存した実行を行なうことが可能となる．今回は，次の属性やメソドを定義した．

- 作業環境

  文書の変更や参照にどのようなツールを用いるかを，それぞれ属性として定義した．また，主に作業している計算機の名前や，各作業者が担当している生成物の名前も属性として定義する．また，各作業者に割り当てられた役割も属性として定義しておく．

- 電子的な連絡

  例題では，作業者が他の作業者に連絡を行なう場合には電子メール等が用いられている．よって，各作業者ごとにメールを読み書きするためのメソドを用意する．また，作業者の電子メールアドレスは属性として用意する．

このようにして，各オブジェクトに対して属性やメソドをそれぞれ記述していった．記述された内容は主に「例題の記述から自動的に導出されるもの」でありい，それ以外には「例題には記載されていないが，実際の開発を行なう上で必要となるもの」を記述した．

状態オブジェクトの定義

このようにして定義されたオブジェクトを用いることによって，着目したい開発プロセスや，開発環境中に存在する要素を自由に表現することができる．以下，いくつかの状況において実際にどう記述されるか説明する．

作業結果の閲覧　例題で行なわれる作業の結果はファイルとして出力されている．この結果を纏めることによって，実際に行なわれた作業の内容をふりかえることが可能である．このことを MonoProcess で定義すると以下のようになる．

```
PObject .Doc.Reviewresult.File def
        Attriburte @file;
        Method &view;
endObject

PObject .Doc.Compilerresult.File def
        Attriburte @file;
        Method &view;
endObject

PObject .Doc.testresult.File def
        Attriburte @file;
        Method &view;
endObject

StatusObject .Results def
        PObject .Doc.Reviewresult.File
        PObject .Doc.Compilerresult.File
        PObject .Doc.Testresult.File
endObject
```

つまり，デザインレビュー結果，コンパイル結果，テスト結果を示すオブジェクトから，ファイル名とその閲覧方法だけを抜きだして部分オブジェクトとし，それらを纏めて状態オブジェクトとする．

オブジェクト生成過程の定義　ソフトウェア開発における典型的な作業として「ソースコードをコンパイルしてオブジェクトを作成し，それのテストを実行する」ことが挙げられる．

例題でもこの一連の作業について定義されているが，これは MonoProcess を用いることにより1つの状態オブジェクトとして定義することができる．

```
PObject .Doc.Sourcecode.make
        Attribute @file;
        Attribute @compiler;
        Attribute @objectcode;
        Method &compile;
endObject

PObject .Doc.Objectcode.test
        Attribute @file;
        Method &test;
endObject

StatusObject .MakeAndTest def
        PObject .Doc.Sourcecode.make
        PObject .Doc.Objectcode.test
        Method &makeandtest def
                ...
        endMethod
endObject
```

ソースコードとオブジェクトコードを表わすオブジェクトから，必要な属性とメソドを抽出し，それらを用いて状態オブジェクト.MakeAndTestを定義する．また，状態オブジェクト中で新たにメソドを定義することができ，これによって作業の順序関係等を表現することができる．

3.4.3 試作システムにおける記述の実行

上記で用意したオブジェクトの記述を手動でオブジェクトリポジトリで用いられるデータに変換した上で，今回試作したシステムに導入した．各ユーザに対しては，それぞれのオブジェクトで定義されている情報を単に提示するインターフェースを用意した．ISPW-6の例題は，既に存在しているソフトウェアの仕様に対する変更作業という形態であるため，まず最初に簡単な仕様書とそれに対応するソースコード，単体試験パッケージ等を準備した．そして，例題の実行を「用意した仕様書に対して機能の追加を行なう」形で行なった．

試作システムでは，各ユーザに対してオブジェクトの内容を表示するインターフェースを用意した．このインターフェースでは，あるオブジェクトについてその属性と属性値の一覧，およびメソド名の一覧を表示する．メソド名の部分をマウスで操作することにより，

そのメソドの実行を行なうことができる．本試作システムを動作させた結果，各オブジェクトごとに操作履歴が収集することができることがわかった．その中には「ソースコードの改変履歴」や「コンパイル/単体試験の実行回数/実行時間」など，ソフトウェア開発の進捗管理を行なう上で必要となると考えられる情報が含まれた．

3.5 関連研究との比較

プロセスモデルの役割

ソフトウェアプロセスモデルや，それをもとにするソフトウェアプロセス環境にはさまざまな目的がある [11]．Curtis らは，プロセス記述，プロセスモデル，プロセス中心型開発支援環境には，プロセスの理解から自動実行といった幅広い分野において，5 つの基本的な使われ方があるとしている [9]．それぞれは次のように纏めることができる．

- プロセスの理解や伝達を容易にする
- プロセスの進化を支援する
- プロセスの管理を支援する
- 自動的なプロセスの誘導を行う
- 自動的な作業の実行を行う

提案するソフトウェアプロセスのモデル化では，これらの点を次のような形で実現している．

- 開発環境はオブジェクトの集合として表現されており，開発者も管理者も共通の表現を利用している．各オブジェクトに関する情報はそのオブジェクト内部で閉じている．
- MonoProcess で用いるオブジェクトはオブジェクト指向分析等を用いて分析を行うことができる．このため，オブジェクト指向分析の結果から導出されるオブジェクトの構成等に関する改善点を適用することによって，開発環境の構成をより効率的な形へ進化させることが可能である．

- 開発作業にて行われた操作は，履歴として保存されている．これらを利用することによって，進捗管理等を行う枠組を提供する．
- 各オブジェクトの内部状態を属性として管理することにより，そのオブジェクトに対して現在行える作業を明示できる．
- メソド記述により，定型的な作業については自動的に実行させることができる．非定型的な作業についても，その中に含まれる単純な作業を自動的に実行させることが可能である．

以上から MonoProcess は，ソフトウェアプロセスを表現，記述，動作するために必要な機能をすべて持っていると言える．

ソフトウェアプロセスの記述

MonoProcess はソフトウェア開発環境における制御/データ統合を行うために，オブジェクトという単一の構造に情報と作業の両方を記述する．記述されたオブジェクトを用いて「開発作業中で現在何が行われているのか」を表現し，行われた作業を履歴として管理することによって，開発管理等にも用いることができる．これは，他のプロセス中心型ソフトウェア開発環境 [10, 13] で用いられている言語に見られる「何をしなければならないのか/どうしなければいけないのか」に着目して「予想されない例外に対してどう対処を行うかを記述する」こととは異なる物である．

本研究では開発プロセスを開発時の結果として捉えており，それはさまざまな観点から十分に記述することは非常に困難であるという立場を取っている．MonoProcess では，多様な観点から捉えられる作業それ自身ではなく，作業を構成する作業の断片をメソドを記述している．これにより，開発環境中で行われる内容をより正確に，かつわかりやすい単位で記述することが可能となっている．また，メソドを単位として行われた作業の結果を操作履歴として残すことができるため，開発プロセスをより詳細に表現することが可能となっている．これらのことから，MonoProcess はプロセス記述を行う際に現実的な基盤となり得ると考えられる．

あるオブジェクトのもつ内容を継承したオブジェクトを定義することは，MonoProcess 以外のプロセス記述言語でも行うことができる．しかし，それらは全体構造が木構造にな

ることを仮定している物が多い [44]．このため，多重継承を表現することや，全体におけるあるまとまった一部分を変更することが困難である．Marvel は strategy によって作業をグループ化することができるが，strategy 自体は Marvel のもつデータ構造の定義に依存しているため，自由なグループ化が行えるとは言えない．

MonoProcess では，作業の分類は MonoProcess のもつグループ機能によって任意の作業に対して行える．また，継承機能を用いることによって，複数のオブジェクトで利用する属性やメソドを共有できる．更に，さまざまな粒度における資源や作業等を，同一の属性/メソドラベルを用いることによって，一貫した記述が行える．

作業内容の記述

APPL/A[44] 等の手続き型のソフトウェアプロセス記述言語では，ソフトウェア開発作業は構造化に基づいたプロセスの定義が行われる．すなわち，プロセスを記述する際には，作業の構造を静的に決定する必要がある．Marvel[4, 6, 16] などの，ルールベースのソフトウェアプロセス記述では，作業の記述を行う場合に何が前条件や後条件となるのかを決定する必要がある． MonoProcess においては，記述されるオブジェクトをオブジェクト指向分析等を用いて分析を行うことができるため，オブジェクト指向分析の結果から導出されるオブジェクトの構成等に関する改善方法を用いてプロセス記述をより効率良く行うことができる．また，開発環境中に存在する文書，ツール，計算機資源，開発者等の要素を，それぞれオブジェクトとして集中的に記述することが可能である．

MonoProcess を用いた記述では，開発手順はオブジェクトにおけるメソドとなるため，ある一連の作業の流れが複数のオブジェクトの複数のメソドとして記述されてしまう可能性がある．これによって一連の記述が分散してしまい可読性を損ねる危険がある．しかし，MonoProcess のもつグループ化の機能を用いることにより，複数のオブジェクトに分散した情報を単一のオブジェクトへ纏めることが自由に行える．これにより，単一のオブジェクトに着目するだけで必要な情報を集中して扱うことができ，内容を参照する場合にも，単一のオブジェクト内で一連の作業が完結しているため理解しやすくなると考えられる．またこの機能により，参照される範囲を適切に限定することができる．

MonoProcess の枠組では，メソドの記述に関して特定の言語仕様を仮定しておらず，その機能のみを定義している．既存のプロセス記述言語は，言語の機能に強く依存した構造であり，作業の表現方法に何らかの制約を課してしまう形となっている [17]．MonoProcess

をメソド記述言語から独立して定義することによって，メソド記述の際，目的に応じて言語を選択することが可能となる．

作業履歴の管理

MonoProcess のもつ作業履歴の管理は，他には見られない特徴である．既存のオブジェクト指向プログラミング言語やデータベース言語においても，メソドの呼び出し回数や実行を中断して内部状態を観察するための同様の機能が提供されている．しかし，それらは外部ツール等によって実現されているものが多く，言語自体に含まれているわけではない．MonoProcess では，記録されている履歴は属性として扱われているために，他の属性を扱う時と同様にして作業履歴を扱うことができる．

作業履歴の収集は，キーボードやマウスに対する割り込みを利用したり，オペレーティングシステムやアプリケーションに対する追加操作として収集することも考えられる．しかし，このようにして収集される履歴は非常に細かな粒度のデータになってしまうため，これらを用いて「作業者がどの生成物に対してどのような操作を行ったのか」を分析するのは非常に困難であろう．MonoProcess はオブジェクト中の属性やメソドを単位とした履歴の収集を行うため，オブジェクトを収集したいデータに応じて定義することによって，履歴の解析を行いやすいデータを効率良く収集することができる．

3.6 まとめ

本節では，オブジェクト中心型ソフトウェアプロセスモデルのモデル化手法である MonoProcess について述べた．MonoProcess はオブジェクトを用いて開発作業環境中にある生成物や資源を表現する．MonoProcess ではソフトウェアプロセスはオブジェクトの状態遷移として表現される．MonoProcess を用いることにより，ソフトウェアプロセスをよりわかりやすく表現することが可能であり，作業履歴を用いることによってプロセス管理をより効率良く行うことができる．

今後，試作環境を実際の開発環境で運用することによって本モデルの機能に関する評価を行う予定である．具体的には，数人の開発者と管理者で構成される開発チームに利用してもらい，オブジェクトの数やその内容の記述状況，システムの利用パターン等の記録を行い，実験終了後にその結果を分析し，開発者等に対する聞き取り調査を実施する予定で

ある．また，より抽象化されたメタオブジェクトを本モデルに導入することにより，本モデルをソフトウェアプロセスの改善を行う際の枠組として発展させる予定である．

第4章　開発作業者による評価を想定した品質評価規格によるプロセス評価環境

ソフトウェア開発プロセスを改善することは，生産効率の向上やコストの削減などに直接影響を与えるため，非常に大きな意味をもつ．ソフトウェア開発プロセスの改善を行うためには，まず現在のソフトウェア開発がどのように行なわれているかを評価することが必要となる．

このような背景から，近年ソフトウェアプロセスの品質評価/品質保証に関する研究が盛んに行われ，また利用されている [21, 42]．現在までに，ソフトウェアプロセスに対してさまざまな評価方法や参照モデルが提案されてきた．たとえば，SEI（ソフトウェア工学研究所）の CMM （Capability Maturity Model）[39, 40] や，ISO（国際標準化機構）の ISO 9000 シリーズ [54]，SPICE（Software Process Improvement Capability dEtermination）[57] などがあげられる．

品質評価規格に基づいて開発組織のプロセスを評価し，その短所や長所などの特徴を認識した上で，プロセスを改善していくことは非常に重要な課題である．通常，プロセスの評価は，その品質評価規格を熟知している組織外の機関や専門家が，評価対象となる組織の人間とのインタビューや，過去のプロジェクトのドキュメントの閲覧などを通じて行われる．しかし，このような方法では，時間や手間が非常にかかるなど，組織に対する負担が大きく，何回も繰り返し実施することは困難であることが指摘されている [21, 49]．

本研究では，実際の開発現場で作業を行っている開発者やその管理者が自らのプロセスを評価し，改善すべきヒントを得ることを支援するシステムの構築を目指している．このような自己評価では，外部組織から評価結果に対する認証を受けることはないが，プロセス評価を適切な基準を用いて手軽に行うことが出来る．これにより，各組織においてより多くの品質評価作業が行われるようになり，品質評価規格がより広く利用されるようになると考えられる．また，開発現場においては，評価結果をもとにして必要な改善措置を効

率良く早急に行なうことができると考えられる．

このようなシステムを作成するためには，何らかの形で品質評価規格が定義し要求する事項を形式化してシステムの中に組み込むなければならない．一般的には，品質評価規格における各種定義やプロセスに対して要求される事項を整理し，それに基づいて，システムのアーキテクチャを設計し実現する方法[37]が一般的と思われる．この方法では，品質評価規格で定められているさまざまな事項を解釈してシステムのアーキテクチャに変換する必要があるが，対象が長大な規格文書の場合，何がどう整理されたのかが不透明になりやすい．また，その解釈が品質評価規格の想定している内容と一致するか否かが重要な問題になる．

ここでは，品質評価規格に対する独自の解釈をできるだけ小さくなるように，また，できるだけ簡明で直観的に理解できるように，タスク，レベル，プロダクトの 3 種類の要素に対し，その間の関係を定義したものをモデルとして用いる．そして，必要に応じて，品質評価規格の原文も参照できるように，もとの品質評価規格の文書へ SGML （Standard Generalized Markup Language）[53]のタグを，これらの要素や関係が記述されているところに挿入することによってモデルを記述した．これによって，品質評価規格を形式的に扱うことができ，モデルに基づいたプロセスの評価作業が行いやすくなる．また，これを用いて構築されるシステムにおいては，品質評価規格で記述されている内容と共に品質評価規格の原文も容易に参照することができる．

また，実際に品質評価規格文書の参照，評価作業を行うために，関連するタグ間の移動や検索，演算処理などを目的とした独自の支援システムを設計，試作した．ここで提案する方法を用いてモデル化を行うことにより，長大で複雑な品質評価規格文書も比較的容易に形式化でき，支援システムも容易に構築できた．

以下，4.1 節では，本研究にて品質評価規格の例として取り上げる SPICE について説明する．4.2 節では，提案するモデル化手法について述べる．4.3 節では，今回試作したプロセス評価支援システムの機能を述べ，4.4 節では本研究に関して考察を行う．最後に 4.5 節でまとめと今後の課題について述べる．

4.1 品質評価規格 SPICE

4.1.1 概要

SPICE（Software Process Improvement Capability dEtermination）とはプロセス品質評価規格の1つであり，ISO/IEC JTC 1/SC7 の WG10 にて国際標準化作業が行われているものである．SPICE 文書全体は 9 章で構成されており，記述は約 400 ページに渡っている．

4.1.2 開発作業の分類

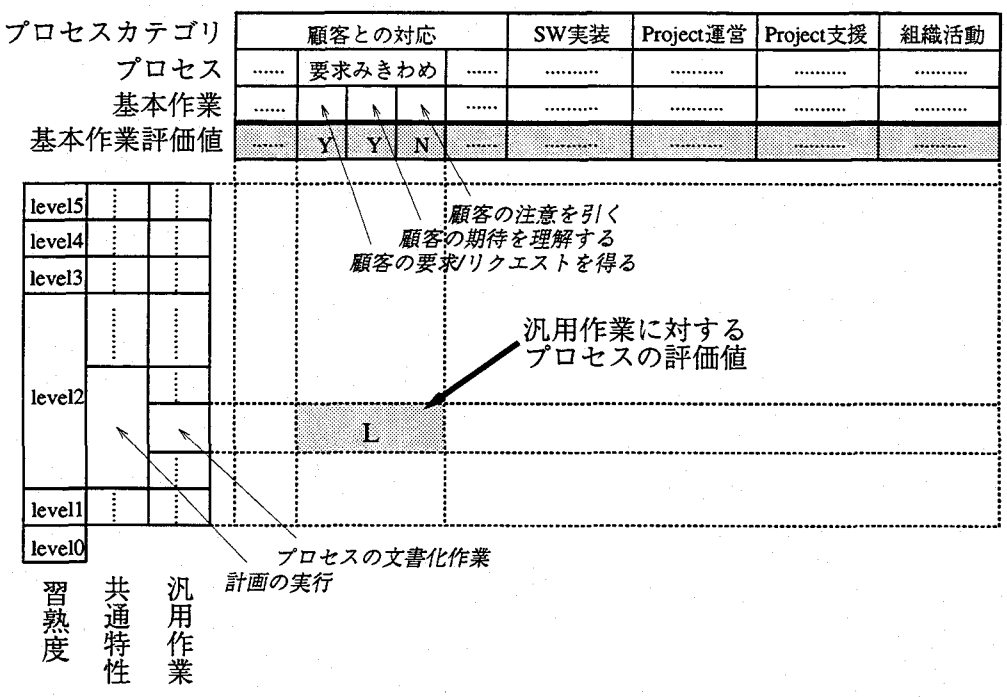

図 4.1: SPICE における開発作業の分類と評価値の決定

SPICE では，ソフトウェア開発におけるさまざまな種類の作業を 1）顧客との対応，2）ソフトウェアの実装，3）プロジェクト運営，4）プロジェクト支援，5）開発組織全体の活動，という5つのプロセスカテゴリ（Process Category）としてグループ化する [58]．各プロセスカテゴリはいくつかのプロセス（Process），各プロセスはいくつかの基本作業（Base Practice）から構成される．

一方，これらの作業の達成度を評価する軸として，レベル 0 からレベル 5 まの習熟度

(Capability Level) がある．各レベルの習熟度はそのレベルの特徴を表すいくつかの共通特性 (Common Feature) から構成される．更に共通特性はいくつかの汎用作業 (General Practice) で構成される (図4.1)．汎用作業においては，作業を遂行するために必要となるプロセス/基本作業や，必要となる作業プロダクトの種類 (Work Product Type) が定められている [61].

4.1.3 評価手順

品質評価を決定する際の手順は，大きくわけて 1) 評価対象の決定，2) 評価情報の収集，3) 評価値の決定と集計，という 3 つの作業で構成される [60].

評価対象の決定においては，評価を実行する範囲，評価を実行する組織の規模や構成について決定を行う．

評価情報の収集においては，SPICE で定義された開発作業と評価対象となる開発作業との対応を把握した上で，評価対象からさまざまな情報の収集等を行う．

評価値の決定と集計においては，収集した情報をもとに各評価項目についての評価値の決定，集計を次のように行う．まず，各基本作業がどの程度実行されているか，を 2 段階 (存在している (Y)，存在していない (N)) または 4 段階 (完全に実行している (F)，大部分実行している (L)，部分的に実行している (P)，全く実行していない (N)) で評価する．また，各プロセスが各汎用作業に対してどの程度効率的に実行されているかを 4 段階 (F, L, P, N) で評価する (図4.1)．これらの結果は最終的に各習熟度ごとに集計される [59].

SPICE では更に，集計された結果を用いたプロセス改善の方法やプロセス能力判定のためのガイドライン，評価作業の実行方法についても規定されており，開発プロセスを改善するための指針として十分な内容となっている．

4.2 提案するモデル

品質評価規格は，一般に開発プロセスやその習熟の度合を定義した上で，実際の開発作業をどのように評価すればよいかを記述していると考えられる．例えば，SPICE 文書では，プロセスや習熟度等の定義や評価指針等は主に 2 つの章に書かれており，これら全部の記述は SPICE 全体のおよそ半分に相当する約 200 ページに渡っている．

本節では，品質評価規格で記述されているこれらの項目のモデル化手法について，SPICEを用いて説明する．

4.2.1 モデル定義

品質評価規格が評価対象とする開発作業全体を以下の3つの要素とその間にある4種類の関係を用いてモデル化する．

（要素）

- タスク[1]
  開発時における作業のうち，開発工程を構成するような作業．SPICEにおけるプロセスカテゴリ，プロセス，基本作業がこれに相当する．
- レベル
  作業の達成度を示す要素．SPICEにおける習熟度，共通特性，汎用作業がこれに相当する．
- プロダクト
  開発時におけるさまざまな生成物，若しくは事前に準備されている成果物．SPICEの評価の際にその存在や内容が調べられる．

（関係）

- タスク - プロダクト間関係
  あるタスクが実行時に必要とするプロダクトを示す
- タスク - タスク間関係
  あるタスクが実行時に必要とするタスクを示す
- レベル - タスク間関係
  あるレベルの遂行にあたって必要とされるタスクを示す
- レベル - プロダクト間関係
  あるレベルの遂行にあたって必要とされるプロダクトを示す

[1] 通常，非可分的な作業をアクティビティ，アクティビティの集合をプロセスと呼ぶ場合が多い．しかし，SPICEが定義するプロセスとの混同を避けるため，ここでは，アクティビティ/プロセス等の総称としてタスクという単語を用いる

SPICE の場合，本モデルにおける要素は，SPICE において定義されている要素を単純に抽象化して得ることができる．また，本モデルにおける構成要素間の関係は，SPICE 文書中に記述されている明示的な関係の記述と対応させることができる．このように，本モデル化は個別の品質評価規格の内容とは独立しているが，SPICE の文書からはほぼ機械的に，必要かつ十分な定義を導出することができる．また，本モデル化は SPICE のような大規模の品質評価規格文書に対しても，その意味を変化させるものではないため，他の品質評価規格に対しても対応できると考えられる．なお，実際のモデルは 4.2.2 節で述べる方法により記述する．

このようなモデルを用いることにより，品質評価規格に記述されている品質評価の判断に必要となる情報を，形式的に扱うことができ，容易に参照することができる．例えば，あるタスクの定義内容や，タスクとタスク - プロダクト間関係をもつプロダクトの一覧等を参照できる．また，あるタスクやレベルで示される作業が実際にどの程度実行されているか調べる際に，該当タスクとタスク - タスク間関係をもつ他のタスクの実行状況や，該当レベルとレベル - タスク間関係をもつタスクおよびレベル - プロダクト間関係をもつプロダクト等を参照できる．

4.2.2 SGML を用いた記述

本研究では，4.2.1 節で定義したモデルを，もとの品質評価規格文書に対して SGML (Standard Generalized Markup Language) 文を追加することにより記述する．SGML とは，ISO によって標準化作業が行われている文書構造記述言語である．通常，定型文書は SGML を用いて構造化することができ，文書処理，文書データベース，文書交換等を行うことができる [46]．SGML を用いてモデルを記述することにより，品質評価規格文書自身の中にモデルを含めることができ，もとの文書とモデルの両方を利用するツールを容易に作成することができる．

ここでは，モデルにおける要素については `ELEMENT` という SGML のタグ（文書中に付加するマーク）を，関係については `RELATION` というタグを用いて記述を行う．各要素，関係のもつ情報は，タグに対する属性として記述することによって，品質評価規格に書かれている情報を的確に表現することができる．

本研究では，SPICE に対して 4.2.1 節で定義したモデルを表現するために，これらのタグを SPICE の文書に付加した．この他，SPICE の文書構造を表現するためのタグを定義

表 4.1: ELEMENT タグの属性

TYPE	要素の種類を表す．TASK, PRODUCT, LEVEL のいずれかの値を取り，それぞれタスク，プロダクト，レベルを表す．
ID	各種類中で一意に決定される文字列．SPICE においてはプロセス，成果物の種類，習熟度に付けられた識別子を値としてもつ．
SUBID	各 ID 中で一意に決定される文字列．SPICE においては基本作業，共通特性に付けられた識別子を値としてもつ．
SUBSUBID	各 SUBID 中で一意に決定される文字列．SPICE においては汎用作業に付けられた識別子を値としてもつ．

し SPICE の文書に付加した．以下，モデルを記述するために定義した 2 種類のタグについて説明する．

要素に関するタグ

要素はすべて ELEMENT という名前のタグで表現する．ELEMENT タグには表 4.1 のような属性を付ける．

関係に関するタグ

関係はすべて RELATION という名前のタグで表現する．RELATION タグには表 4.2 のような属性を付ける．

SGML タグの記述例

前節で定義した SGML タグを，SPICE 文書中に付加する作業を行った結果の例を図 4.2 に示す．

この例は，ソフトウェアの要求仕様を作成する基本作業を定義した文章にタグが付けられている．先に挙げた ELEMENT や RELATION の他，前書き部分を表す PREAMBLE，基本作業の名前を表している TITLE，定義の本文を表す BODY といった文書構造を表現するタグが付けられている．

<ELEMENT TYPE=TASK ID="CUS.1" SUBID="2">
<PREAMBLE>
CUS.1.2
</PREAMBLE>
<TITLE>
Define the requirements.
</TITLE>
<BODY>
Prepare the system and software requirements
to satisfy the need for a new product and/or
service. Note: This definition of the requirements
may be done completely or partially by the supplier.
<RELATION TYPE=TKTK SRC="CUS.1.2"
DST="ENG.1" DST="ENG.2">
See "Develop System Requirements and Design"
ENG.1 and "Develop Software Requirements" ENG.2
</RELATION>
<RELATION TYPE=TKTK SRC="CUS.1.2"
DST="CUS.3.1">
Also see CUS.3.1, "Obtain customer requirements and
requests." CUS 1.2 is focusing on defining requirements
when the software organization is acting as a customer.
CUS.3.1 is focusing on obtaining requirements when the
software organization is acting as a supplier. The primary
difference is one of perspective, the role being preformed.
</RELATION>
</BODY>
</ELEMENT>

図 4.2: SPICE 文書を SGML によってモデル化した例

表 4.2: RELATION タグの属性

TYPE	関係の種類を表す．SPICE においては，TKPD, TKTK, LTP のいずれかの値を取り，それぞれ「タスク-プロダクト間関係」「タスク-タスク間関係」「レベル-タスク間関係若しくはレベル-プロダクト間関係」を表す．
SRC	関係上における参照元を表し，参照元となる要素の識別子を値としてもつ．
DST	関係上における参照先を表し．参照先となる要素の識別子を値としてもつ．
DIRECT	タスク-プロダクト間関係にのみ存在する属性で，プロダクトがタスクへの入力となるか出力となるかを表す．IN 若しくは OUT のいずれかの値を取る．

4.3 支援システムの試作

SGML によってモデル化された SPICE 文書を利用する，品質評価支援システムを試作した．本システムはソフトウェア開発に携わる開発者や管理者自身が自らのプロセスを評価する際に利用することを目的としている．

4.3.1 概要

本システムでは，4.1.3 節で述べた評価手順のうち，主に 2）と 3）を支援する．すなわち，本システムでは，文書の閲覧，文書間関連の検索や演算，SPICE と実際の開発との対応関係の把握，評価結果の導出の際に必要となる情報の提示，評価値の入力や集計といった機能を利用者に提供する．

本システムは，ツールと付随するデータベースによって構成される（図 4.3）．ツールには，文書中に記述されている各要素や，要素間の関連を表示する文書参照ツールと，文書と実際の開発作業とを参照することによって評価支援を行う評価支援ツールの 2 種類が存在する．両ツールとも，モデル化した SPICE の文書を入力とし，SGML タグの解析を行う．解析結果は必要に応じてデータベースとして保存し，ユーザからの要求に備える．過去に入力された品質評価結果は，別途データベースの形で保存される．以下では，試作した 2 種類のツールについて説明する．

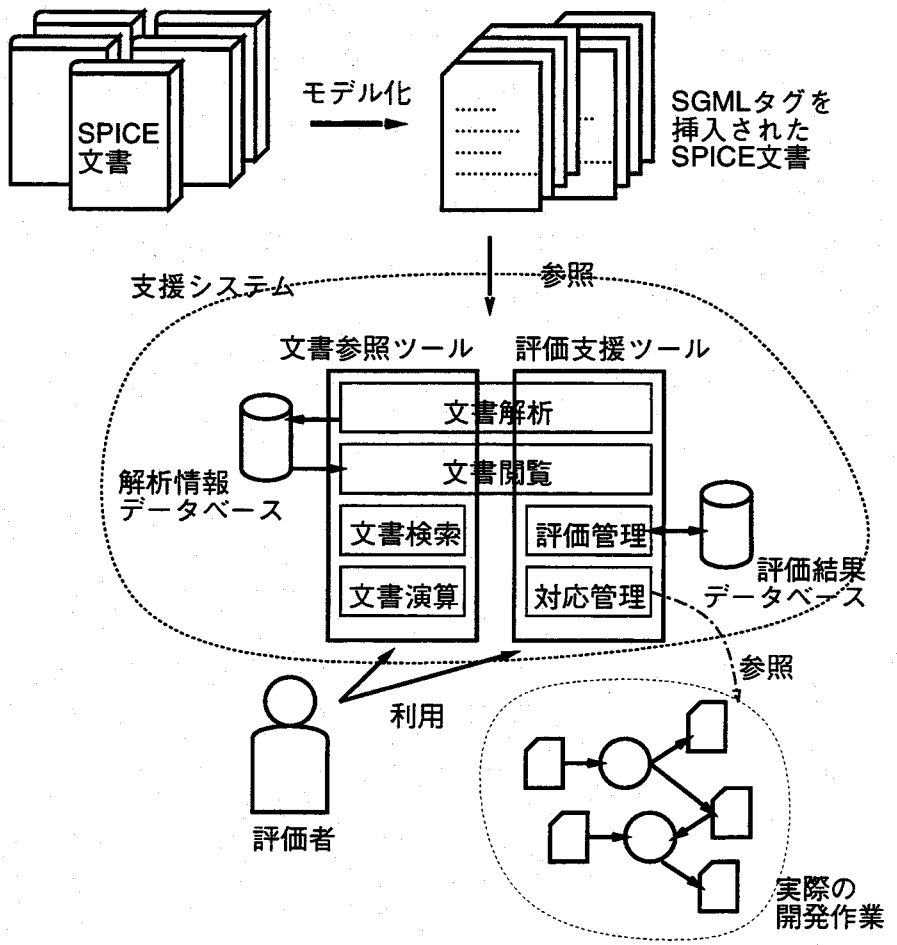
SPICE
文書
モデル化
SGMLタグを
挿入された
SPICE文書
支援システム
参照
文書参照ツール
評価支援ツール
文書解析
文書閲覧
解析情報
データベース
文書検索
文書演算
評価管理
対応管理
評価結果
データベース
参照
利用
評価者
実際の
開発作業

図 4.3: システム概要

4.3.2 文書参照ツール

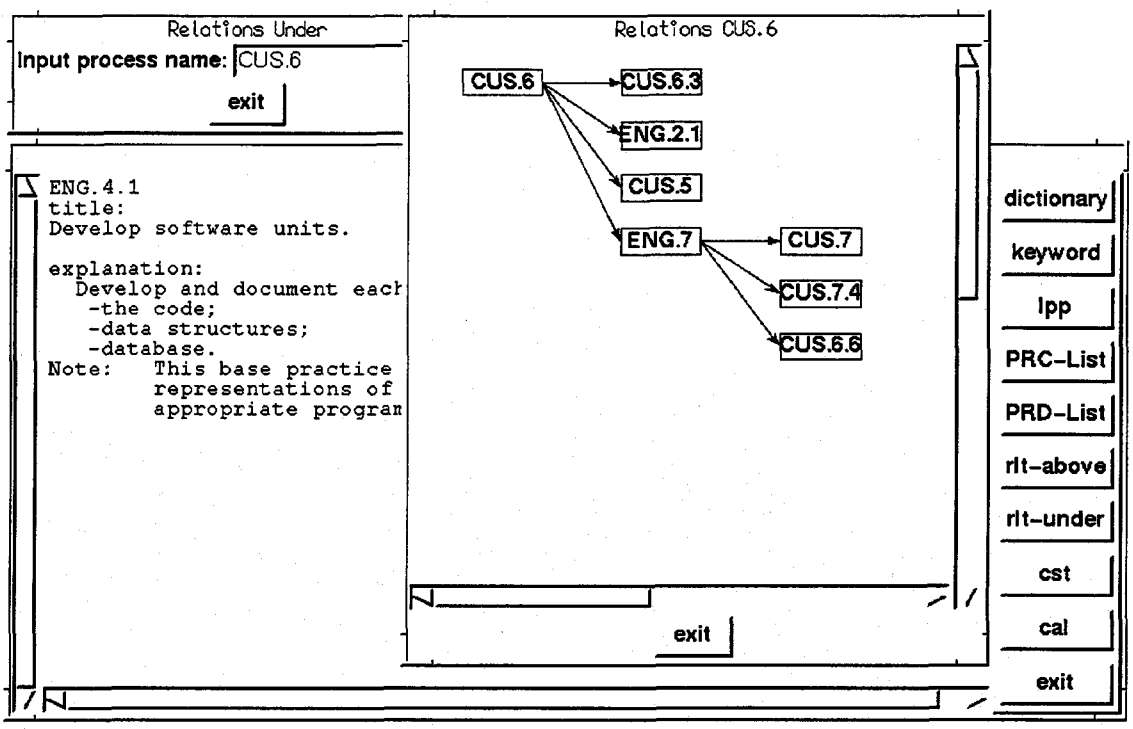

図 4.4: 文書参照ツール

文書参照ツールとは，SPICE 文書中に記述されている各要素に関する内容を表示したり，作業間の参照関係を調べて簡単なグラフ表現にする等，文書中の情報を参照しやすくするためのものである．本ツールの実行画面を図4.4 に示す．

文書参照ツールには次のような機能がある．

- 用語検索

  SPICE が定義している各要素の名前を入力すると，該当する要素が定義されている文章を表示する．

- キーワード検索

  文書中に書かれている単語をキーワードとして与えると，キーワードが含まれる要素を表示する．検索した要素を選択することによって，要素が定義されている文書の表示が行える．

- 要素間関連検索

  モデルの各要素間で定義されている関連の検索を行う．このとき，モデルによって定義されている関連を再帰的に辿ることによって派生した関連の検索も行える．

- 組み合わせ演算

  上記で提供されている検索結果を組合せた演算を行うことができる．複数の検索結果の和集合および積集合の計算や，ある検索結果として得られた複数の要素をそれぞれ別の検索実行に対する入力として用いることができる．

文書参照ツールでは，このような機能の結果を視覚的に表示する．演算の結果が集合となる場合にはリストの形で表示し，木構造となる場合にはその形を表示する．図4.4では，あるタスクを評価する際に参照するべき他のタスクが木構造の形で表示されている．

4.3.3 評価支援ツール

評価支援ツールとは，開発者等が自ら行うプロセス評価作業を支援するためのものである．本ツールの実行画面を図4.5に示す．

評価支援ツールには次のような機能がある．

- 評価に必要とされる情報の閲覧

  評価するタスクやレベルを選択すると，文書中での定義が書かれている文章や，文書中に記述されている関連のあるタスク，プロダクトが表示される．既に評価済みであるタスクの表示の際には，その評価結果もあわせて表示する．

- 評価規格と実際の開発作業との対応関係の入力および表示

  モデルで定義されているタスクやプロダクトと，実際の開発作業において生成されたファイルの組を入力できる．この対応関係を把握することにより，どのタスクやプロダクトが実際の開発環境中に存在しているかを判断することができる．入力された対応関係は表形式で整理，表示する．入力された対応関係はデータベースへ保存する．また，対応関係に登録されたファイルの内容表示も行える．

- 評価結果の入力

  評価したいタスク（SPICE におけるプロセス）を指定すると，それに含

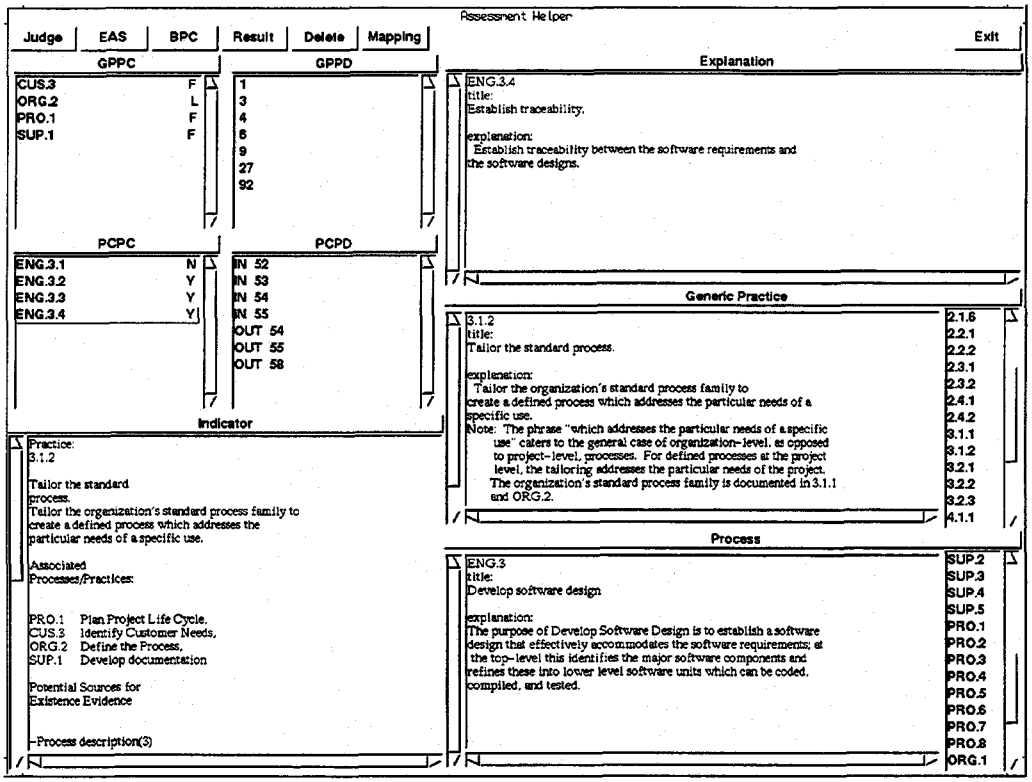
Assessment Helper
Judge
EAS
BPC
Result
Delete
Mapping
Exit
GPPC
CUS.3 F
ORG.2 L
PRO.1 F
SUP.1 F
GPPD
1
3
4
6
9
27
92
Explanation
ENG.3.4
title:
Establish traceability.
explanation:
Establish traceability between the software requirements and the software designs.
PCPC
ENG.3.1 N
ENG.3.2 Y
ENG.3.3 Y
ENG.3.4 Y
PCPD
IN 52
IN 53
IN 54
IN 55
OUT 54
OUT 55
OUT 58
Generic Practice
3.1.2
title:
Tailor the standard process.
explanation:
Tailor the organization's standard process family to create a defined process which addresses the particular needs of a specific use.
Note: The phrase "which addresses the particular needs of a specific use" caters to the general case of organization-level, as opposed to project-level, processes. For defined processes at the project level, the tailoring addresses the particular needs of the project. The organization's standard process family is documented in 3.1.1 and ORG.2.
2.1.6
2.2.1
2.2.2
2.3.1
2.3.2
2.4.1
2.4.2
3.1.1
3.1.2
3.2.1
3.2.2
3.2.3
4.1.1
Indicator
Practice:
3.1.2
Tailor the standard process.
Tailor the organization's standard process family to create a defined process which addresses the particular needs of a specific use.
Associated Processes/Practices:
PRO.1 Plan Project Life Cycle,
CUS.3 Identify Customer Needs,
ORG.2 Define the Process,
SUP.1 Develop documentation
Potential Sources for Existence Evidence
Process description(3)
Process
ENG.3
title:
Develop software design
explanation:
The purpose of Develop Software Design is to establish a software design that effectively accommodates the software requirements; at the top-level this identifies the major software components and refines these into lower level software units which can be coded, compiled, and tested.
SUP.2
SUP.3
SUP.4
SUP.5
PRO.1
PRO.2
PRO.3
PRO.4
PRO.5
PRO.6
PRO.7
PRO.8
ORG.1

図 4.5: 評価支援ツール

まれるタスク（SPICE における基本作業）のそれぞれについて 2 段階による評価の入力を行える．更にレベルを指定すると，タスクとレベルの組に対して 4 段階による評価の入力が行える．評価結果の入力の際には，評価値の候補が一覧として表示され，それを選択することによって行う．入力された評価結果はデータベースへ保存する．

- 評価結果の集計
  入力された評価結果を，プロセスカテゴリごとに，プロセスと汎用作業をそれぞれ縦軸，横軸とした表形式で表示する．また，各習熟度ごとにプロセスおよびプロセスカテゴリ全体の集計を行った結果を百分率で表示する．

実際にこのシステムを用いて「ソフトウェアプロセスモデリングのための例題 [18]」に基づいた開発プロジェクトの評価を行い，ツールの各機能の確認を行った．

4.4 関連研究との比較

モデルの定義

4.2.1 節で述べた，本論文にて提案するモデル化手法は，SPICE のような比較的大きい文書に対してもほぼ機械的に情報を抽出することができる手法である．通常この種のシステムを作るためには，対象となる品質評価規格をシステム設計者が解釈し，それに応じたアーキテクチャを作成する [37] のが一般的であると思われる．この場合，システム設計者の解釈の違いによって，評価方法やその結果が変化する可能性がある．本研究の方法は，原文中にある要素の定義や参照関係などを直接利用しており，システム設計者の解釈による問題が少ないと思われる．

モデルの記述

本研究では，文書のモデル化に SGML を採用し，それに基づいたツールを試作した．SGML は電子文書の再利用，全文データベース等に利用されている例があり [33, 45]，広く使われている手法と言える．しかし，これらのデータベース等では同種類の大量の文書に対する検索を提供しているのに対し，今回試作した文書参照ツールでは一つの長い文書に対する検索等の機能を提供している．

また，[52, 55, 56] をはじめとして，種々の SGML ベースの支援環境が利用できるようになってきている．しかし，これらの環境では，通常文書全体の構造を SGML を用いて定義し，文書構造や表現の変換を目的として設計されているものが多い．本研究にて必要とする機能は，タグ付けされた一部の文書の参照や検索であるため，SGML ベースの支援環境は規模が大きすぎて使いにくいと考えられる．

インターネット上のアプリケーションとして WWW（World-Wide Web）が広く普及している．WWW では，HTML（HyperText Markup Language）[5] と呼ばれる SGML のアプリケーションが用いられている．本研究においても，SGML の代わりとして HTML を用いることが考えられる．HTML を利用することによって既存の豊富な HTML 用ツールを利用することも可能ではあるが，本研究の目的のためには，HTML を一部拡張する必要があり，HTML の汎用性が失われるおそれがある．従って今回はより一般的な SGML を用いたが，HTML を支援システムにおける文書の表示部分に活用することは可能である．

支援システム

通常，実際のプロジェクトに対して品質評価を行うには，長い期間が必要である．試作した評価支援ツールでは，評価結果を途中で保存することができるため，長期間にわたる評価時にも利用することが可能である．本システムは実際に開発作業を行っている作業者が実作業と規格との対応関係の把握や，実際の開発作業の評価を行いながら作業を進めていくことができる．

今回試作したシステムでは，4.1.3 節で述べた SPICE における評価手順のうち，評価情報の収集および評価値の決定と集計という 2 つの作業について支援を行っている．システムが支援するこれらの作業はすべて SPICE で述べらている評価手順 [60] に従っているため，本システムを用いることによって，SPICE に基づいた評価を正しく行うことができる．これによって得られた評価に基づいて改善を行うことにより，開発組織は品質評価規格をより有効に活用することができるであろう．

4.5 まとめ

本節では，品質規格文書を SGML によりモデル化する方法を提案した．また，これに基づいて開発者や管理者が自らのプロセスを評価を行う際に利用するための評価支援シス

テムを試作した．本方法を用いることにより，容易に品質規格がモデル化でき，その支援システムが構築できた．

今回は品質規格文書として SPICE を取り上げたが，今後は本モデル化を他のプロセス品質評価規格へ適用し，より柔軟なシステムを構築していく予定である．

第5章　むすび

5.1　まとめ

本論文では，ソフトウェアプロセスにおける 3 つの目的のそれぞれに則したモデル化について述べた．まず，ソフトウェア開発環境におけるインタラクションを明示的にモデル化することにより，作業者間の連絡作業を確実に把握することを可能とした．次に，分散開発環境を前提としたプロセス実行環境の構築を行った．本環境の構築のために，オブジェクトを用いた実行環境のモデル化を行った．最後に，既存の品質保証規格のモデル化を行い，評価作業を支援するための環境を構築した．

5.2　今後の研究方針

本論文では，ソフトウェアプロセスにおける各目的に応じてそれぞれ個別のモデルを構築した．しかし，これらのモデルを相互に融合させることにより，より高度なソフトウェア開発環境のモデル化を行うことが望まれるであろう．例えば，本研究における品質評価規格文書のモデル化手法をオブジェクト指向のプロセスモデルへ適用することによって，モデルの記述がより可搬性の高い物となり，実行時における支援と，評価時における支援の双方に利用できることが期待される．

さらに，既存のソフトウェア開発環境とプロセス支援環境を融合させることにより，作業者の作業効率をより高め，かつプロセスを用いることによる開発作業の質を向上させることができるであろう．これにより，従来の統合開発環境や独自の形式によるプロセス中心型開発環境とは異なる，真のプロセス中心型開発支援環境が構築できると考えられる．

その他，開発支援環境を用いて収集された作業データを分析することにより，ソフトウェアプロセスに対するメトリクスの確立を行うことが今後重要となると考えられる．これらのメトリクスを確立させることにより，従来のプロセス改善に関する方法論をさらに改良させることができると考えられる．これにより，ソフトウェア工学における重要な課題で

ある，高品質のソフトウェアを短期間に効率よく作成する手段を確立させることができるであろう．

関連図書

[1] 青山幹雄, "分散開発環境: 新しい開発環境像をもとめて", 情報処理, Vol.33, No.1, pp.2-13 (1992).

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

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

[4] N. Barghouti, "Supporting Cooperation in the MARVEL Process-Centered SDE", In Proceedings of the 5th ACM SIGSOFT Symposium on Software Development Environments, pp.21–31 (1992).

[5] T. Berners-Lee, and D.W. Connolly, "Hypertext Markup Language - 2.0", RFC1866, Massachusetts Institute of Technology Laboratory for Computer Science / The World Wide Web Consortium, ⟨URL:ftp://ds.internic.net/rfc/rfc1866.txt⟩ (1995).

[6] I. Ben-Shaul, G. Kaiser, and G. Heineman, "An Architecture for Multi-User Software Development Environments", In Proceeding of 5th ACM SIGSOFT/SIGPLAN Symposium on Software Development Environments, pp.149–158 (1992).

[7] B.W. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, pp.61–72 (1988).

[8] G. Bolcer and R. Taylor, "Endeavors: A Process System Integration Infrastructure", *Proc. 4th International Conference on the Software Process*, pp.76–89 (1996).

[9] B. Curtis, M. Kellner, and J. Over, "Process Modeling", *Comm.ACM*, Vol.35,No.9,pp.75–90 (1995).

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

[11] A. Fugetta, "Functionality and architecture of PSEEs", Information and Software Technology, Vol.38, No.4, pp.289–293 (1996).

[12] A. Fuggetta and A. Wolf, "Software Process", John Wiley & Sons (1996).

[13] H. Iida, K. Mimura, K. Inoue, and K. Torii, "HAKONIWA: Monitor and Navigation System for Cooperative Development Based on Activity Sequence Model", In Proceedings of 2nd International Conference on Software Process, pp.64–74 (1993).

[14] 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ソフトウェアプロセス研究会報告, 95-SP-2-1, pp.1-10 (1995).

[15] Proceedings of 4th International Conference on Software Process, IEEE CS Press (1996).

[16] G. Kaiser, P. Feiler, and S. Popovich, "Intelligent Assistance for Software Development Maintenance", IEEE Software, Vol.5, No.3, pp.40-49 (1988).

[17] T. Katayama, "A Hierarchical and Functional Software Process Description and Its Enaction", In Proceedings of 11th Int. Conf. on Software Eng., pp.343-352 (1989).

[18] M.I. Kellner, P.H. Feiler, A. Finkelstein, T. Katayama, L.J. Osterweil, M.H. Penado, and H.D. Rombach: "Software Process Modeling Example Problem", In Proceedings of the 6th Int. Software Process Workshop, pp.19-29 (1990).

[19] M.I. Kellner and W. Humphrey, "Software Process Modeling: Principles of Entity Process Models", In Proceedings of the 11th International Conference on Software Engineering, pp.331–342 (1989)

[20] K. Lai, and T.W. Malone, "Object Lens: A "Spreadsheet" for Cooperative Work", Proceedings of CSCW'88, pp.115–124 (1988).

[21] F. MacLennan and G. Ostrolenk, "The SPICE Trials: Validating the Framework", In Proceedings of the 2nd International SPICE Symposium, pp.109–118 (1995).

[22] T.W. Malone, et al., "Semistructured Messages Are Surprisingly Useful for Computer-Supported Coordination", ACM Transactions on Office Information Systems, Vol.5, No.2, pp.115–131 (1987).

[23] T.W. Malone, K. Lai, and C. Fry, "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work", Proceedings of CSCW'92, pp.289–297 (1992).

[24] 松下誠, 飯田元, 井上克郎, 鳥居宏次, "開発プロセスを構成する要素間のインタラクションに関する考察", 情処研報, 95–SE–102, pp.165–170 (1995).

[25] 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」, 情報処理学会研究報告, 97-PRO-14-17, pp.97–102 (1997).

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

[27] M. Matsushita, M. Oshita, H. Iida, and K. Inoue, "Distributed Process Management System Based on Object-Centered Process Modeling", In Proceedings of Worldwide Computing & Its Applications '98, pp.108–119 (1998).

[28] M. Matsushita, H. Iida, and K. Inoue, "An Interaction Support Mechanism in Software Development", In Proceedings of 1996 Asia-Pacific Software Engineering Conference, pp.66-73 (1996).

[29] 松下誠, 飯田元, 井上克郎: Web を用いたソフトウェア開発環境のためのプロセスモデリング, 情報処理学会論文誌, 39 巻 3 号, page 830-832 (1998).

[30] M. Matsushita, H. Iida, and K. Inoue, "Web-based Process Management System with Object-Centered Process Modeling", In Proceedings of The International Symposium on Internet Technology 1998, page 92-97 (1998).

[31] 松下誠, 飯田元, 井上克郎: 品質評価規格文書のモデル化とそれに基づく評価支援システムの構成, 電子情報通信学会論文誌 D-I, 採録決定 (1998).

[32] 松下誠, 飯田元, 井上克郎: ソフトウェア開発におけるインタラクションの支援方式, 電子情報通信学会論文誌 D-I, 採録決定 (1998).

[33] 森田歌子, 鈴木政彦, 宮川謹至, 浜中寿, "SGML 方式よる情報管理誌全文データベース化の可能性と HTML による電子情報管理誌の試作", 情処学研報, 95-FI-37-2, pp.7–14 (1995).

[34] 大下誠, 永山裕之, 山本哲男, 松下誠, 楠本真二, 井上克郎:「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」, 電子情報通信学会技術研究報告, SS97-86, pp.71–78 (1998).

[35] 落水浩一郎, "ソフトウェアプロセスに関する研究の現状", ソフトウェア科学会ソフトウェアプロセス研究会, SP93–1–1, pp.1-11 (1993).

[36] 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).

[37] 小元規重, 込山俊博, 藤野喜一, "ソフトウェアプロセス評価支援システム「SPATS」について", 情処学研報, 96-SE-102-28, pp. 159–164 (1995).

[38] L. Osterweil, "Software processes are Software too", In Proceedings of 9th Int. Conf. Software Eng., pp.2–13 (1987).

[39] M. Paulk, B. Curtis, M. Chrissis, and C. Wever, "Capability Maturity Model for Software, Version 1.1", Software Engineering Institute, CMU/SEI-93-TR-24 (1993).

[40] M. Paulk, B. Curtis, M. Chrissis, and C. Wever, "Key Practices of the Capability Maturity Model, Version 1.1" , Software Engineering Institute, CMU/SEI-93-TR-25 (1993).

[41] R.S.Pressman: "Software Engineering A Practitioner's Approach, fourth edition", McGRAW-HILL (1997).

[42] H. Saiedian, and R. Kuzara, "SEI Capability Maturity Model's Impact on Contractors", IEEE Computer, Vol.28, No.1, pp.16–26 (1995).

[43] S. Sutton, D. Heimbigner, and L. Osterweil, "Language Constructs for Managing Change in Proces-Centered Environments", In Proceedings of 9th Int. Conf. Software Eng., pp.328-338 (1987).

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

[45] 高柳由美子，坂田英俊，田中洋一，"SGML による全文データベースシステム"，情処学研報，93-CH-18-5，pp.35–42 (1993).

[46] 田中洋一，"文書記述言語 SGML とその動向"，情報処理，Vol.32，No.10，pp.1118–1125 (1991).

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

[48] L. Wakeman and J. Jowett, "PCTE The standard for open repositores", Prentice Hall International (1993).

[49] I. Woodman, and R. Hunter, "Analysis of Assessment Data from Phase 1 of the SPICE trials", Software Process Newsletter, No.6, pp.1–6 (1996).

[50] ソフトウェアプロセスシンポジウム論文集，情報処理学会 (1994).

[51] 特集 "グループウェアの実現に向けて"，情報処理，Vol.34，No.8 (1993).

[52] DocIntegra, (URL:http://www.hitachi.co.jp/Prod/comp/soft1/open/docint.htm)，日立製作所 (1995).

[53] ISO 8879, "Information Processing - Text and Office System - Standard Generalized Markup Language (SGML)" (1986).

[54] ISO 9000-3 Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software (1991).

[55] OLIAS, ⟨URL:http://www.fujitsu.co.jp/hypertext/openworld/topics/olias/olias.html⟩, 富士通 (1996).

[56] Panorama Pro, ⟨URL:http://www.sq.com/products/panorama/panor-fe.htm⟩, SoftQuad Inc. (1996).

[57] The SPICE Project, "Software Process Assessment – Part 1: Concepts and Introductory Guide", Version 0.02 (1994).

[58] The SPICE Project, "Software Process Assessment – Part 2: A Model for Process Management", Version 0.01 (1994).

[59] The SPICE Project, "Software Process Assessment – Part 3: Rating Process", Version 0.01 (1994).

[60] The SPICE Project, "Software Process Assessment – Part 4: Guide to conducting assessments", Version 0.02 (1994).

[61] The SPICE Project, "Software Process Assessment – Part 5: Construction, Selection and Use of Assessment Instruments and Tools", Version 0.02 (1994).