ソフトウエア工学 93−10
（1993．7．8）

協調開発プロセスモデルを用いたプロジェクト管理手法について

飯田 元† 三村圭一† 井上克郎† 鳥居宏次†‡

†大阪大学基礎工学部 情報工学科

‡奈良先端科学技術大学院大学 情報科学研究科

本研究では，多人数による大規模なソフトウェア開発におけるプロジェクト管理を支援するための複合プロセス記述モデルと，それを用いたプロジェクト管理支援方法を提案する．提案するプロセス記述モデルは，多人数による協調作業やプロダクト間の関係，開発要員の組織構成を表す複数のモデルから構成される．本稿ではさらに，提案する手法に基づいた協調開発支援モニタ「はこにわ」バージョン２の設計についても述べる．

Development Management based on Cooperative Software Process Model

Hajimu IIDA†, Kei-ichi MIMURA†, Katsuro INOUE† and Koji TORII†‡

†Faculty of Engineering Science, Osaka University

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

This paper describes a cooperative process model of software development and the development support/monitor system "Hakoniwa" (version 2) which is based on the model. The proposed model is a kind of concurrent and communicating sequential processes. This model has been extended from the previous version of it to allow hierarchical process description and dynamic task instance creation/deletion. Hakoniwa system has been extended also to support these features with excellent graphical interface, which displays the status of software process in real time.

1 はじめに

ソフトウェアの大規模化や人件費の増大につれて，一箇所での集中したソフトウェア開発が非常に困難になりつつある．その結果，地域分散開発などの支援に関する研究が盛んに行なわれるようになってきた．

分散開発は「処理の分散」と「組織の分散」の二つの観点から捉えることができ [1]，従って，分散開発において生じる問題点も次の二点に分類できよう．

- 物理的な距離に起因する問題（通信速度やコストの問題）
- 意思疎通や情報管理の未熟さに起因する問題

前者の問題に対しては，分散処理の基盤技術開発や高速回線の敷設やテレビ会議システムなどの利用といった面から解決がはかられている．一方，後者は多人数で開発を行う際に存在する問題で，物理的な分散とは直接は無関係である．つまり，一拠点での集中開発の場合には開発要員同士が非形式に直接対話可能であったものが，物理的に分散した環境においては不可能になったために表面化した問題であると言えよう．

本研究は，ソフトウェア開発プロセスモデルを基礎にした情報管理を用いて，後者の問題の解決を図るものであり，分散環境を特には前提とせずに「協調開発」のモデル化を行う．

2 協調開発プロセスモデル

一般に，ソフトウェア開発プロセスとは，企画・設計段階から保守段階までがどのような工程で行なわれるかといったことや，各工程でどのようなドキュメントを参照し，どのような作業を行い，どのような成果物を作成するかといった事項を定めたものをいう [11]．

従来，多くのソフトウェア開発会社では標準開発手順と呼ばれる，プロセスモデルの一種を用いてきた．これらの標準開発手順は，製品の品質向上と工程管理の効率化を主な目的として用いられているが，そのほとんどは上流工程から下流工程への一方通行的な進行を仮定した「ウォーターフォール・モデル」に基づくものである．従って，設計ミスによる上流工程のやり直しや複数のグループによる並行した開発などを十分反映することができない．そこで，協調分散開発支援を考慮した新たなプロセスモデルが必要であると考えられる．

本研究では，特に多人数による並行した開発プロセス（協調プロセス）を意識したモデルを新たに定義し，それに基づいた記述を用いて進捗状況の把握や各要員の誘導，プロジェクト管理・制御情報の自動伝達を行うことを考える．

2.1 複合プロセスモデル

本稿で提案するプロセスモデルは，筆者らが [5] で提案した複合ソフトウェアプロセス・モデルの概念に基づくものである．複合ソフトウェアプロセスとは，ソフトウェアプロセスの持つ様々な要素を，個々の要素毎に単純なコンポーネントモデルとして定義し，それらを組み合わせることで複雑なプロセスを構成するものである（図1）．

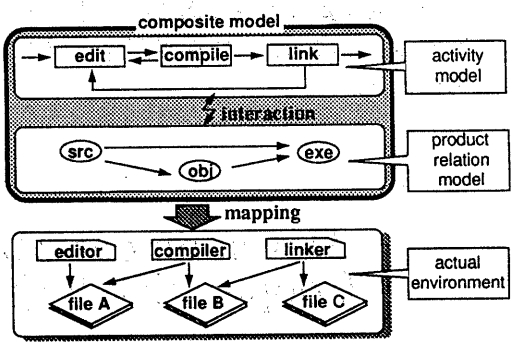

図 1: 複合ソフトウェアプロセスモデル

ここでは，まず協調開発における工程間の相互作用に視点をおいたコンポーネントモデル CTM/H (Cooperative Task Model with Hierarchy) を提案する．これはチーム内の協調開発を表すためのモデル「作業モデル」[4] を拡張したものである．本節では，まず「作業モデル」について述べ，続いて今回の拡張点について述べる．

2.2 協調開発用コンポーネントモデル（作業モデル）

実際の開発作業では複数の工程がそれぞれ別の開発者によって並列に行なわれることが多く，開発者は互いに連絡をとりながら工程間の進行の調整や他の工程への制御を行う．

このとき，一人の開発者が互いにあまり関係のない複数の工程を担当することも考えられる．そのため，開発

者が協調開発の基本構成要素であるような記述では工程間の関係が明確に表せない．そこで，ここでは工程を主体にモデルを構築し，開発者は単にいくつかの工程の担当者としてとらえる．

タスク

各開発者は複数の工程を並行して行なっても良いが，それぞれの工程の内部は本質的に逐次に行なわれるものでなくてはならない．もし，ある工程が逐次的に表せない場合には，それをいくつかの逐次的な工程の並列集合に分解する．このようにして得られる逐次的な工程のことをここでは「タスク」と呼ぶことにする．

各タスクは逐次的な動作によって表されるが，タスクを表現するアルファベットに相当するものを「基本作業」と呼ぶ.「作業」という言葉のとらえ方の単位には,「ファイルを編集する」といった小規模のものから「システムの仕様を変更する」といった大規模かつ多人数で行うものまである．ここでは，最小の作業単位 (ツールの実行や人間による判断など) を「基本作業」として定義する．タスクの持つ逐次的な制約は「基本作業」の集合をアルファベットとした形式文法（ここでは正規文法）で定義する．

以上をまとめると，

- 「タスク」とは「基本作業」の系列（正規表現）として定義され，
- 「開発プロセス」は，並列に実行される「タスク」の集合として定義される．

タスク間の通信

このとき，タスク間の同期をとったり他のタスクを制御したりするために，基本作業の一種として同期／非同期の通信動作（文字列の送受信）をタスクに持たせることができる．各タスクに送られたメッセージは受信用ポート（キュー）に書き込まれ，メッセージを送信する際には送り先のタスクとその受信ポートを指定する．一つのタスクは複数の受信ポートを持つことができる．基本的な通信動作を表 1に示す．

更に，タスクの開始，終了の記述を簡略化するために，表 2に示すようなタスク制御用オペレーションが用意されている．これらは基本通信動作を用いて実現されており，すべてのタスクに制御通信専用のポートが定義されている．

表 1: 基本通信操作

操作	引数	動作	戻り値
send	task,port	文字列を送信する（非同期）	——
recv	port	ポートから文字列を読みとる（同期）	文字列
peek	port	ポート内のメッセージの有無を調べる（非同期）	論理値

表 2: タスク制御用通信操作

操作	引数	動作
start	task	他のタスクに開始要請を送る
wait	task	他のタスクの終了を待つ
exit	task	自タスクの終了を通知する

「作業モデル」におけるタスク間の関係をより直観的に表現するために，図 2に示すような図式表現を用いる．丸みを帯びた四角がそれぞれタスクを表し，矢印がタスク間でやりとりされるメッセージを示している．メッセージの種類は矢印の横に自然語で記述する．

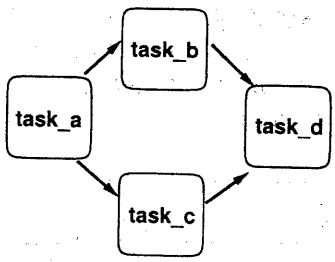

図 2: 作業モデル

2.3 協調タスクモデル CTM/H: 作業モデルの拡張

前節で述べた「作業モデル」は，比較的小規模なグループ内での協調開発プロセスの表現は行なえたが，多人数による大規模な開発プロセスや，動的な変動要素を持つプロセスを表現することは困難であった．

そこで，CTM/H では更に

- プロセスの階層的表現
- プロセスの動的な変更

を可能とするよう，モデルの拡張を行なった．

並列プロセスの階層化

大規模な開発プロセスは，抽象的な工程から具体的な作業へと順次階層化していくことで明確で理解しやすい定義が可能となる．また，それに対応して，人員配置も階層的に編成された複数のチームとして行なわれることが多い．階層的なプロセスモデルとしては HFSP[7] をはじめとして，様々なモデルが提案されている．

一方,「作業モデル」では，個々のタスクは基本作業の系列（正規表現）で表せるような小規模なものを対象としていたため，多人数による大規模な開発プロセスを記述する際に非常に数多くのタスクの記述を平面的に行なうことが必要となり，記述が繁雑で内容の把握も困難になる．そこでタスクのグループ化を用いて階層的プロセスモデルを実現する．

ここでは，並列な複数のタスクをグループ化したものをコンポジット・タスク（複合タスク）として定義する．コンポジット・タスクはお互いに並列に実行されるタスクの無順序集合として定義される．コンポジット・タスクもまたタスクであるので，別のコンポジット・タスクの構成要素となることができる．このようにして，階層的なプロセス構造が定義できる．なお，タスク間の通信は同一グループ（コンポジット・タスクのメンバ同士）内に限られ，他のグループとの通信は親タスクのポートを一度経由して行なわれる．

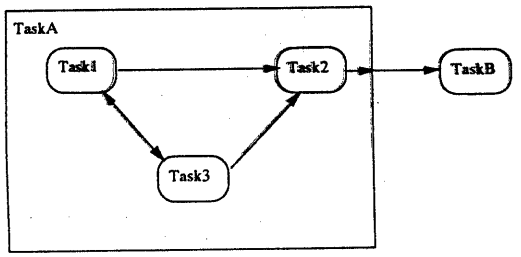

図 3: コンポジット・タスク

この階層化は単純なグループ化を用いたもので，下部階層をサブルーチンのように共有することも無い，階層的に定義されたコンポジットタスクを展開することによって，もとの「作業モデル」と同等の記述が得られる．従って，この階層化モデルはもとの「作業モデル」と同一の意味定義を持つと言える．

プロセスの動的な変更

プロセスの動的な変更には，例えば次のように様々なレベルがある．

(1) タスクの定義内容を変更するもの
(2) プロセスの基本的な構造を変更するもの（複数のタスク間の関係や構成を変更するもの）
(3) タスクのインスタンスの動的な生成・削除

(1),(2) のようにタスクの定義内容を実行中に変更すると，変更の結果が予測が困難であり，また，変更後の整合性の問題なども存在する．ここでは比較的弱い変更として，(3) のタスクのインスタンスの動的な生成・削除を考える．

まず，関連するプロダクトの指定を抽象化したタスクの記述をテンプレートとしてクラス化しておく．そしてテンプレートに実プロダクトへの参照をパラメータとして与えることで，タスクのインスタンスを生成する．この操作は従来の「作業モデル」ではプロセス実行前に行なわれていたが，今回の拡張では，関連プロダクトが確定した時点で生成が行なわれ，関連プロダクトが消滅した時点でタスクインスタンスも削除される（図 4）．このとき，タスクの構造自体はテンプレートに定義されているので変化しない．

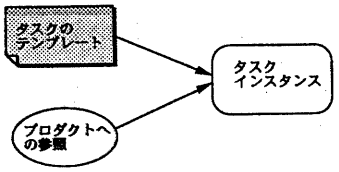

図 4: タスクインスタンスの生成

例えば，ソフトウェアを複数のモジュールに分割して開発するような場合，分割されたモジュールの個数はプロセスの実行の途中まで確定しない．しかし，個々のモジュール開発のためのタスクの構造は共通のものを用いることができるので，これらのタスクは，テンプレートを定義しておいて動的に生成・削除することが考えられる．

この機構を用いると，共通のテンプレートを持つ個数不定のタスクの集合は，その時点でのプロダクト集合への参照から生成することで対応できる．例えば,「デザイン作成」タスクでプログラムを複数のモジュールに分割した後，モジュール毎に「モジュールの作成」タスクを作成するようなプロセスでは，図 5に示すようなタスク生成が行なわれる．

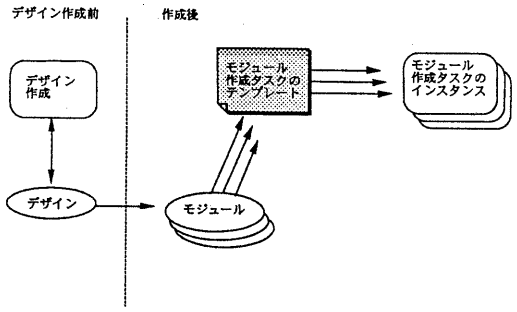

図 5: 不定個数のタスクの生成

3 他のコンポーネントモデルとの結合

以上述べたような拡張を行なったモデルを実開発に適用する場合，ソフトウェアプロセスにおける開発要員の組織構成といった人的な要素や，プロダクトの構成や管理といった要素を考慮する必要がある．これらの要素を協調タスクモデルに直接導入することはモデルの複雑化を招き，タスク間の協調関係を不明確にする．

そこで，これらの要素はそれぞれ独立したコンポーネントモデルとして導入し，協調タスクモデルとのインターフェースを定める．

3.1 PRM/G: プロダクトモデル

プロダクトとは，ソフトウェアプロセスの各工程によって参照・生成されるファイルやドキュメントのことを指す．ここでいうプロダクトモデルとは各プロダクトの構成やプロダクト間の関係を定義するものをいい，以下に挙げるようなオブジェクト指向の特性を持つものとする．

- プロダクトモデルはプロダクトの集合 $\{P\}$ とその上での関係 $\{R\}$ の集合として定義される．
- プロダクトの型（クラス）を定義でき，実プロダクトはそのインスタンスである．
- 様々なサブプロダクトを構成要素として持つようなプロダクトクラスを定義できる．
- 各インスタンス固有の値を持つ属性フィールドを定義できる．
- クラス固有の操作（メソッド）を定義できる．

このような要件を満たすオブジェクトモデルは数多く存在するが，ここでは我々が [3] で提案した関数型言語に基づいたプロダクト・リレーション・モデル PRM/G (Product Relation Model based on Grouping) を用いる．このモデルでは上に示したような要素はすべてプロダクトのメンバ関数として定義され，クラス定義はこれらの関数の集合として表される．

3.2 OSM/T: 組織モデル

CTM/H では，人員割当はタスク集合 $\{T\}$ から開発要員の集合 $\{M\}$ への写像 Rとして定義される．しかし，ある時点 t で，あるタスク T_iに対して割り当て可能な人員の集合 $\{M_{ti}\}$ を求めるに当たっては，組織の構成など $\{M\}$ のもつ性質をより詳細に定めておく必要がある．

組織モデル OSM/T (Organization Structure Model based on Team) は，このような情報を記述・定義するための枠組で，基本的に以下のような情報が定められるものとする．組織モデルの基本構成要素は開発要員（人間）である．また，人間の集合としてチームが定義される．このモデルでは階層的なチーム構成を仮定するので，チーム内にサブチームを定義することも可能である．

3.3 モデル間の相互作用

モデルを分離することによってそれぞれの視点からプロセスを分かりやすく捉えることができる．これらを実際に組み合わせて用いる場合，各モデルがお互いにどのように作用し合うかを定める必要がある．本稿で用いた3つのモデルでは，以下に示すような相互作用を行なう（図 6）．

- 協調タスクモデルでタスク T の生成プロダクトとして P が定義されているとする．T が実行されると，プロダクト関係モデルに対して P の生成要求が送られる．プロダクト関係モデル（厳密にはモデル記述に基づいて動作するプロダクトサーバ）は，P の実体を生成する．
- プロダクト関係モデルで P のサブプロダクトとして不定個数の p_i が定義され，P の実体が生成されると p_iの個数が決定され，実体が生成されるとする．この時，p_i の生成が協調タスクモデルに通知され，各 p_iに対応したタスクが生成される．
- 協調タスクモデルでタスクを要員に割り当てる時には，親タスクの担当者がリーダーを勤めるチームのメンバから選ばれる．この情報は組織モデルによる記述から提供される．
- 組織モデルにおいて，あるメンバーの現在のタスクの担当状況は，協調タスクモデルでのタスクの割当情報から得られる．

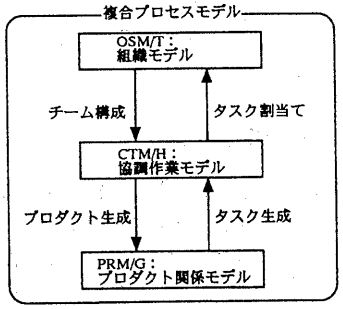

図 6: モデル間の相互作用

4 協調プロセスモデルを用いたマネジメント

前節までで示したモデルに基づいてプロセスを予め記述することによって，以下のような用途に役立てることができる．

人員配置 分散開発の場合は特にコミュニケーションのコストを押えることが重要である．我々のモデルではタスク間のコミュニケーションが明確に記述されているので，できるだけ遠隔地間のメッセージが少なくなるようにタスクのグループ化を行うことが容易になる．

進捗管理 現在のプロセスの状態を常に記述と対応づけておくことによってプロジェクトの状態をより細かく把握することができる．特に，ウォーターフォールベースのモデルと異なり，繰り返しや並列が採り入れられているので，行詰まり（異常な多い繰り返し回数や異常に長い実行時間）やデッドロックなどを容易に発見することができる．

コスト見積り タスクの種類毎に，どの程度の時間やリソースを消費するかといったことを見積もったり，タスクの多重度（ある時点でどれだけのタスクが並行して行なわれるか）といったことを予め知ることで，必要なコストやリソースを知ることができる．

作業支援 各開発要員への仕事の割り当てはタスク単位で行なわれる．従って，自分の担当する仕事が複数ある場合でも，何が割り当てられていてそれぞれ現在どのような状態にあるのかを明確に知ることが可能となる．さらに，タスクの定義から作業誘導システムを容易に生成することができる [3]．

5 はこにわ Version 2

拡張したタスクモデルに基づいたプロセス記述に基づいて開発を支援しモニタするシステム「はこにわ」バージョン 2 の設計を行なった．本節ではこのシステムの機能を紹介する．

5.1 システムの構成

はこにわでは，プロセス記述を入力として，その記述が表すプロジェクトの進行を誘導し，同時にそれをモニタする．システムは誘導・モニタ用の作業インタフェースとモニタ情報を収集するはこにわサーバから構成される（図 7）．

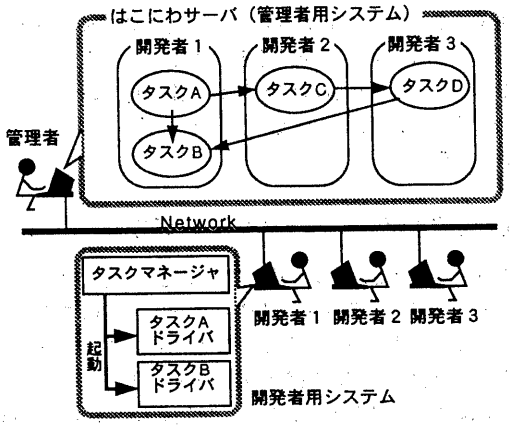

図 7: はこにわ支援システムの概容

5.2 タスクインスタンスの生成と割当

プロジェクト開始時のタスク割当は管理者によってサーバを通じて行なわれる．サーバは，割り当て情報に基づいた作業誘導プランを作業インターフェースに通知する．バージョン 1 ではプロセスの実行前にすべてのタスクインスタンスの生成を行ない，開発者への割りつけを行なった後，各開発者用の支援環境を生成していた．バージョン 2 ではタスクの階層化とタスクインスタンスの動的な生成を採り入れた結果，コンポジットタスクを割り当て

られた開発者がその内部タスクのインスタンスの生成と人員割当を行うように変更された．

プロセス実行前に関連プロダクトが未決定であるタスクは，それが決定されるまでインスタンスは存在しない．関連プロダクトが決定した時点で，親タスクの担当者がサーバに対してインスタンスの生成と担当者を指示する．生成・割当が行なわれると，そのタスクはネットワークを通じて割り当てられた開発者に送信される．

5.3 既存の機能

その他，既存のものとして以下のような機能がある．

誘導：誘導はタスク単位で行なわれる．開発者支援ナビゲータはメニューによる作業誘導を行う．メニューは2階層に分かれており，現在担当しているタスクの一覧が表示されるタスクメニューが現れる．新しく生成されたタスクは，開発者に割り当てられた時点でその開発者のタスクメニューに現れる．開発者がその中からタスクを選択すると，次にそのタスクに関する作業メニューが現れる．作業メニューには，現在そのタスクで行うべき基本作業一覧が表示される．現在行うべき作業の集合は，タスクで定義されている作業系列と，各作業を行うための前提条件から求められる．

モニタ：開発者用インターフェースはメニュー選択によって進行して行く作業の履歴をサーバに通知する．これらの情報はサーバによってリアルタイムに表示され，管理者がプロジェクト全体の状態を把握するのに役立つ．

タスクの階層ごとに，タスク間のメッセージの受渡し関係が表示される．また，各タスクを表す楕円形には，タスクの担当者，実行時間，現在の状態（活性，非活性，停止）などが示される（図8）．さらに，開発者毎のタスクの実行状態の一覧および基本作業の実行履歴（図9），コンポジット・タスク以外のタスクの内部状態（図10）が表示される．

タスクの自動協調：タスク間のコミュニケーションはナビゲータによって自動的に行なわれる．通信はすべてサーバを介して行なわれ，記録される．

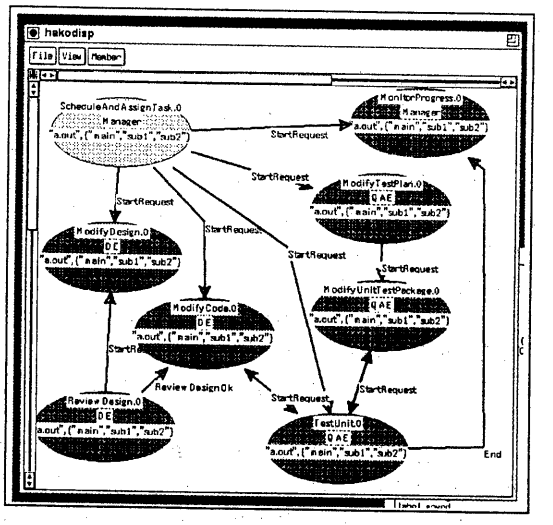

図 8: タスク関係図

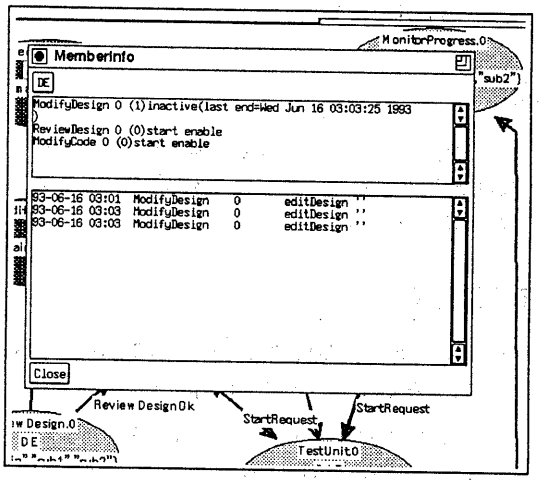

図 9: 開発者毎の情報

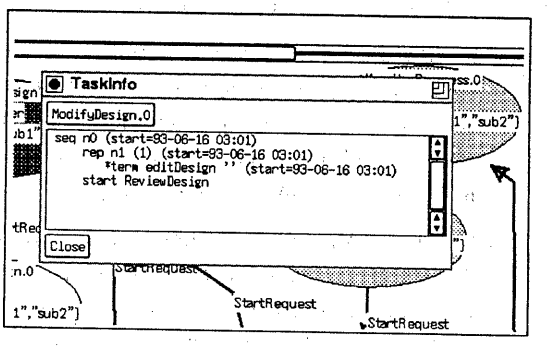

図 10: タスクの内部状態表示

6 おわりに

協調作業を考慮したプロセスモデルを提案した．このモデルを用いることにより協調開発をより詳しく分析することが可能である．また，提案したモデルに基づくプロジェクト管理手法について考察した．さらに，この手法を支援するシステム，はこにわバージョン２の設計を行ない，現在実装作業を行なっている．

問題点としては，ウォーターフォールモデルやPERT図などに比較してモデルが詳細過ぎるため，逆に進捗の大雑把な推定が困難になることが挙げられる．このモデルに基づいて進捗の度合を定量化できるようなプロセス・メトリクスを定めることが望ましい．

今後，作成したシステムを用いた実プロジェクト応用への実験を予定しており，さらに，はこにわをプロダクト・マネジメントを行う本格的なサーバシステムと結合することにより，統合支援・計測環境として拡張していくことも検討中である．

謝辞

はこにわシステムの表示部は日本電気（株）より提供を受けた鼎ツールキットを用いて作成した．日本電気のご厚意に感謝します．

参考文献

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

[2] Iida,H., Nishimura,Y., Inoue,K., Torii,K.: *Generating software development environment from the descriptions of product relations,* in Proc. COMPSAC'91, Tokyo, Japan, pp.487-492 (1991).

[3] Iida,H., Ogihara,T., InoueK., Torii,K.: *Generating a menu-oriented navigation system from formal descriptions of software development activity sequence,* in Proc. 1st Int. Conf. Software Process, Redondo Beach, CA, pp.45-57, (1991).

[4] Iida,H., Mimura,K., InoueK., Torii,K.: *HAKONIWA: Monitor and navigation system for cooperative development based on activity sequence model,* in Proc. 2nd Int. Conf. Software Process, Berlin, Germany, pp.64-74, (1993).

[5] 飯田，荻原，井上，鳥居: ソフトウェア開発作業系列の形式的定義と誘導システムの生成, 情報処理学会論文誌, Vol.34, No.3, pp.523-531 (1993).

[6] Inoue,K., Ogihara,T., Kikuno,T., and Torii,K.: *A formal adaptation method for process descriptions,* in Proc. 11th Int. Conf. on Software Engineering, Pittsburg, PA, pp.145-153(1989).

[7] Katayama,T.: *A hierarchical and functional software process description and its enaction,* in Proc. 11th ICSE, Pittsburg, PA, pp.343-352 (1989).

[8] Kellner,M.: *Software process modeling example problem,* in Proc. 1st Int. Conf. on Software Process, Redondo Beach, CA, pp.176-186, (1991).

[9] 中山高史，東野輝夫，谷口健一: LOTOSによるソフトウェアプロセスの全体記述と開発者個人毎のプロセス記述の導出, 信学技法 SS91-22, pp.59-67 (1991).

[10] 荻原剛志，井上克郎，鳥居宏次: ソフトウェア開発を支援するツール起動自動制御システム, 信学論 (D-1), **J72-D-I**,10, pp.742-749 (1989).

[11] Osterweil,L. : *Software processes are software too,* in Proc. 9th ICSE, Monterey, CA, pp.2-13 (1987).

[12] 落水浩一郎: ソフトウェアプロセスモデルに基づくソフトウェア開発支援環境 Vela, 日本ソフトウェア科学会第７回大会論文集, pp.205-208, (1990).

[13] Saeki,M., Kaneko,T., Sakamoto,M. : *A Method for software process modeling and description using LOTOS,* Proc. 1st Int. Conf. on Software Process, Redondo Beach, CA, pp.90-104 (1991).

[14] Williams,L.G. : *Software process modeling: A behavioral approach,* Proc. 10th ICSE, Singapore, pp.174-186 (1988).