ソフトウェア工学 131－5
（2001．6．1）

プロトタイプベースモデリング支援環境の提案

大塚 聖也　岡戸 悟　上田 賀一

茨城大学 工学部 情報工学科

〒316-8511 茨城県日立市中成沢町 4-12-1

本研究では，開発効率の良いプロトタイプベースモデリング支援環境を提案する．本環境はドメイン特化した図式表現を可能にするために，図式表現を意味情報と視覚情報に分け，両側面を編集可能にした．またシステム開発を，対象ドメインの基本モデルであるドメインモデル開発と，対象システムであるアプリケーションモデル開発に分割した．これらは，既存の図式記述法では非効率的記述となるドメインへのドメイン特化な図式記述を可能にし，また，環境側にてユーザを明確に分離することを可能にした．このことはユーザに求められる専門的知識や技術の分割も可能とする．

A prototype-based modeling support environment

Masaya OHTSUKA　Satoru OKADO　Yoshikazu UEDA

Ibaraki University

4-12-1 Naka-Narusawa, Hitachi, Ibaraki, 316-8511 Japan

This paper proposes a prototype-base modeling support environment for efficient software development. To represent the diagram specialized in a domain, both aspects of view information and semantic one of diagrammatic models are made editable. In the environment, software development is divided into domain modeling which makes the base model of a target domain and application modeling which makes the target models. This environment leads user to make domain specific diagrammatic representation of target software system and distinguishes between the high and low-level users of special knowledge and techniques.

1　はじめに

近年のソフトウェアシステムに対する要求は大規模化・複雑化の一途を辿っている．背景にはハードウェア性能の向上により，ユーザの高度な要求受け入れが可能になったことや，ソフトウェアの適用分野の拡大などがある．

これらの状況下において，対象物を簡単化・抽象化することで関係者相互のコミュニケーションを円滑にするER図やデータフロー図など図式表現によるモデル化は，その有効性を認められ様々なソフトウェア開発に使用されている．

こういったCASEツールの中には，仕様記述言語を用いることによりシステムのプロトタイプ作成を支援するものがある．しかしそれらの多くが記述様式にデータフロー図や状態遷移図などと図式表現が固定されており，その記述言語の適用性が高いシステムでは有効であるが，適用性が低いシステムでは有効ではない．

本報告では，図式表現を固定せず，システム作成

をドメインを定義する部分と定義されたドメインを用いてシステムを作成する部分に分割することを提案する. 図式表現は, 視覚情報と意味情報にし, システム作成においては, ドメインを定義する部分をドメインモデル開発者が, システム作成をアプリケーションモデル開発者が担当する. 分割・編集することにより柔軟な表現を可能にする. さらにメタ概念 [1][2][3] を導入することにより, 意味情報の編集を可能にする.

はじめにこのプロトタイプベースのモデリング環境の特徴について述べ, そのなかでメタ階層について説明する. 次に本環境の設計・実装について述べる. その本環境におけるモデリング手順を示し, 関連研究との比較を行い, 最後にまとめと今後の課題について述べる.

本章ではメタ階層の概念とモデル開発の分割について述べる.

2 メタ階層

2.1 ERF モデル

ERF モデルとは, 実世界を実体 (エンティティ) と関連 (リレーションシップ) で表現する ER モデルを基本として, それにプロトタイピングにおけるオブジェクト指向の有用性からエンティティとリレーションシップをオブジェクトとして扱うことにし, さらにオブジェクトの集約を扱うために, フィールドの概念を取り入れたものである.

- **エンティティ** モデリングの対象世界を構成する実体オブジェクト.
- **リレーションシップ** 2 つのエンティティ間に存在する関連.
- **フィールド** エンティティ, リレーションシップ, フィールドを集約するオブジェクト.

2.2 メタメタモデル

メタメタモデルの構成要素は ENTITY, RELATIONSHIP,FIELD である.

- **ENTITY** システムを構成する実体オブジェクト
- **RELATIONSHIP** ENTITY や FIELD 間の関連オブジェクト

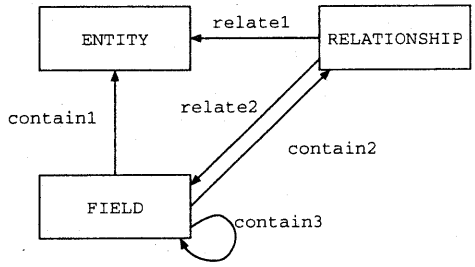

図 1: ERF モデルによる基本概念

- **FIELD** ENTITY,RELATIONSHIP, FIELD の集約オブジェクト

これらのコンポーネント間の関連は図 1のようになる. 図中に示される relate,contain の各コンポーネントはさらに上位のメタモデルにて定義されるものであるが, 本環境ではメタメタモデルが最上位に位置しているので, これらをメタメタモデルに追加している.

- **relate1** RELATIONSHIP から ENTITY への関連である.
- **relate2** RELATIONSHIP から FIELD への関連で,FIELD は ENTITY としてみなすことも可能なのである.
- **contain1** FIELD が ENTITY を内部に持つことが可能であることを意味する.
- **contain2** FIELD が RELATIONSHIP を内部に持つことが可能であることを意味する.
- **contain3** FIELD が FIELD を内部に持つことが可能であることを意味する. これにより FIELD を使用した階層的記述が可能である.

2.3 メタモデル

メタモデルのコンポーネントとその間の関連は, メタメタモデルの定義に基づきドメイン毎に記述される. その際, 各コンポーネントの属性やメソッドについても定義する.

2.4 モデル

モデルはシステムのプロトタイプであり, メタモデルに基づいて記述される. その際, メタモデルで記述されたコンポーネント間の制約や, フィールド

が内部に持つことのできるコンポーネントの制約を満たさねばならない.

3 モデル開発の分割

本モデリング環境では,利用者をドメインモデル開発者と,アプリケーション開発者に分離して考える.ドメインモデル開発者は,各ドメインに特化したベースモデルとメタモデルの組を開発し,アプリケーション開発者は,ドメインモデル開発者によって開発されたベースモデルを利用して実行モデルを開発する.これにより,メタ階層やオブジェクト指向といったより専門的な事項はドメインモデル開発者のみが理解し,使用することになるので,アプリケーション開発者に専門的知識の多くを必要としないという利点が生じる.

さらに,ベースモデルは意味モデルとビューモデルの 2 部から成る.これはベースモデルを既存の特定なモデル記述法 (例えば,UML, ペトリネットなど) に限定せずに,ドメイン独自の記述法の使用を考慮に入れたものである.ビューモデルでモデルの視覚的記述を行い,意味モデルでモデルの各要素の意味や制限を記述する.ビューモデル部が具体的に図形描画 (矩形, 円形, 矢印) を行い,意味モデル部が図形と対応する動作・解釈的意味を行う.例えば,ビューモデル部にて円形を作成し,意味モデル部で円形について名前や動作するメソッドを記述する.

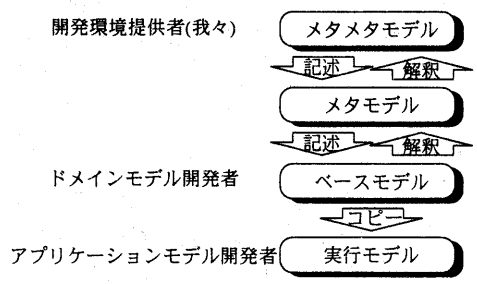

図 2: モデリング環境によるユーザの分離

4 本支援環境

本支援環境では,ソフトウェアシステムに関する要求仕様書を基にドメインモデル開発者がベースモデルを作成し,環境がベースモデルとメタメタモデルからメタモデルを導出する.作成されたベースモデルをコピー・編集してアプリケーションモデル開発者が実行モデルを作成する.ベースモデル,メタモデル,メタメタモデルは意味モデル部とビューモデル部からなり,それぞれを作成・編集することになる.

作成されたモデルは XML によって保存され,また XML によって記述されたモデルの読み込み使用が可能なことを目指す.

また本環境は C++による実装を行い,メソッドの評価・実行には本研究室にて開発・保守されているオブジェクト指向モデル記述言語 Bramble[6] にて行う.

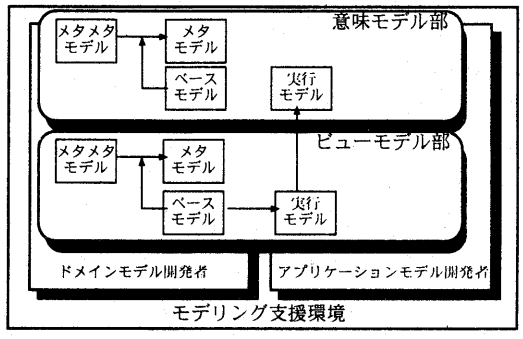

図 3: モデリング環境の概念図

4.1 意味モデル部の設計

意味モデル部はモデルの評価・実行部分である.その際,当研究室にて開発・保守しているコピーベースのオブジェクト指向言語 Bramble とその仮想機械である BrambleVirtualMachine にて実行メソッドの評価・実行を行う.

意味モデル部の目的は,メタ概念に基づいたモデル開発を可能にし,実行メソッドの評価・実行を行う.

本環境を作成し,その際の意味モデル部のクラス図を図 4に示す.

- **SEnvironment** 意味モデル部におけるメインクラス. ビューモデル側のメインとなるVEnvironment と相互にコミュニケーションする.
- **SEntity** 環境にて提供するメタメタモデルクラス.
- **SRelationship** 環境にて提供するメタメタモデルクラス.
- **SField** 環境にて提供するメタメタモデルクラス.
- **SMetaField** メタモデルを集約し, ベースモデルにおけるフィールドのメタとなるクラス.
- **SMetaModel** ベースモデルを記述・解釈するクラス.
- **SBaseField** ベースモデルを集約するクラス.
- **SBaseModel** ベースモデルの基本要素を構成するクラス. このクラスは基本以上のメソッドを持たず, 対応するメタモデルに記述する形となる.
- **SBroker** 実行時はBramble解釈系にて実行されるので,Bramble との仲介を行うクラス.
- **XMLTInterface** XML トランスレータとの通信を行うクラス. ビューモデルと共用する.

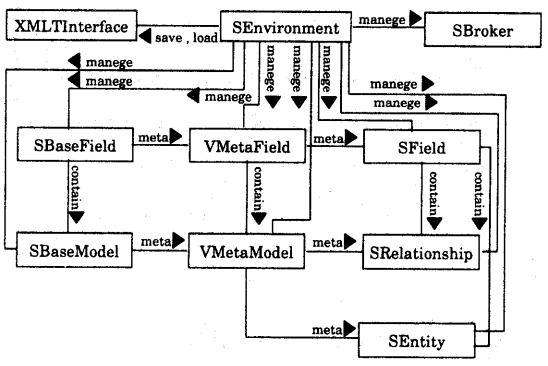

図 4: クラス図 (意味モデル部)

4.2 ビューモデル部の設計

ビューモデル部は環境のユーザインタフェースである. ユーザのモデル入力に対して, その "形","位置" を評価する. その "形","位置" からモデルの視覚的な定義を評価する.

以上から, ビューモデル部の目的は, ユーザフレンドリなモデル定義・記述・動作を可能にすることでプロトタイピングをサポートすることである.

本環境を作成し, その際の意味モデル部のクラス図を図 5に示す.

- **VEnvironment** ビューモデル部におけるメインクラス. 意味モデル部のメインとなるSEnvironment と相互にコミュニケーションする.
- **VBaseEntity** ベースモデルにおける実体を表すクラス.
- **VBaseRelationship** ベースモデルにおける関連を表すクラス.
- **VMetaEntity** メタモデルにおける実体を表すクラス.
- **VMetaRelationship** メタモデルにおける関連を表すクラス.
- **ModelingTool** MainUI,ToolUI, DrawUI,AttributeUI を集約するクラス.GUI とモデルデータの仲介を行う.
- **MainUI** GUI における MainWindow を管理するクラス.
- **ToolUI** GUI における ToolWindow を管理するクラス.
- **DrawUI** GUI における DrawUI を管理するクラス.
- **AttributeUI** GUI における AttributeUI を管理するクラス.

5 本環境におけるモデリング手順

本研究にて提案するモデリング支援環境を用いたモデル開発手順をペトリネットを例に取り上げ, 説明する.

5.1 ペトリネット

ペトリネットとは円形 (プレース), 矩形 (トランジション), 黒点 (トークン), 弧 (アーク) からなる図式的・数学的な有向グラフであり, 並行性, 非同期性, 分散性などの解析に有効である.

ペトリネットを構成する要素は次のようなものである.

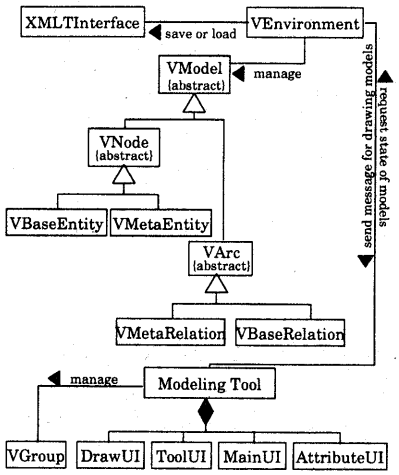

図 5: クラス図 (ビューモデル部)

使用して図形を作成, 関係生成ツールにて 2 図形間の関係を作成する. これを図 6 に示す. ビューモデル・意味モデルの両モデルを作成した後, ビューモデルと意味モデルの 1 対 1 の対応を決定する. またモデルの属性を決定する.

これの実行画面が図 7である.

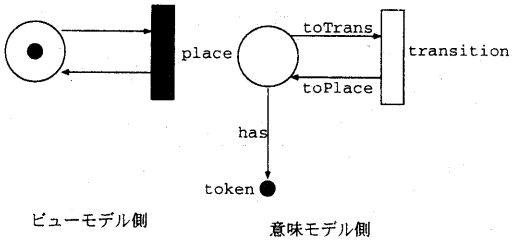

図 6: ペトリネットにおけるベースモデル

- **プレース**
  プレースはモデルにおける条件として捉えることが可能である. プレースにトークンが存在している時, そのプレースと関連づけられている条件が成立したとの解釈が可能である. プレースはまたデータ, バッファなどと捉えることも可能である. プレースは円として描画される.
- **トランジション**
  トランジションはモデルにおける事象と捉えることが可能である. その際, 事前条件と事後条件を表す入力プレースと出力プレースを持つ. トランジションは長方形として描画される.
- **トークン**
  プレースにのみ所有され, そのプレースに関連づけられている条件が成立していることを示す. トークンはプレースの中に黒点として描画される.
- **アーク**
  プレースとトランジションを結ぶ有向パスである. プレースからトランジションへとアークによって結ばれた時, プレースはトランジションの事前条件であるとみなせ, トランジションからプレースへと結ばれた時はトランジションの事後条件とみなす. アークは方向付きの線分として描画される.

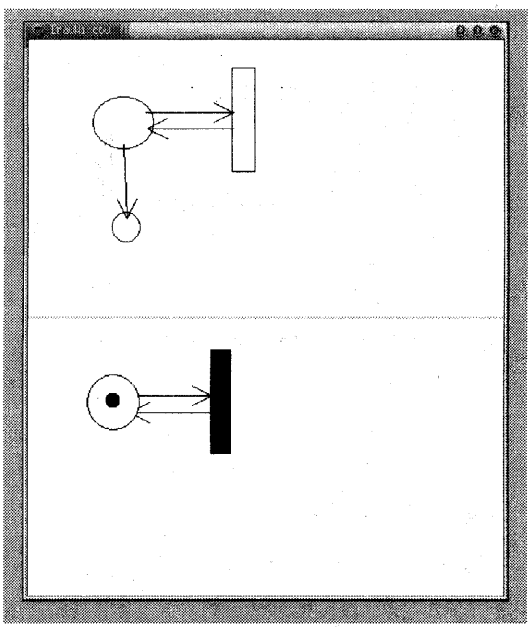

図 7: ベースモデル実行画面

5.2 ベースモデル

以上の事項を基にペトリネットのベースモデルを作成する. ユーザは GUI 上にて作成し, その際, キャンパスに円形描画ツールや, 矩形描画ツールを

5.3 メタモデル

ベースモデル作成後, ユーザによるメタモデル生成依頼によって, 環境がメタメタモデルとベースモ

デルから, メタモデルを導出する. ただし導出されたメタモデルはドメイン用の属性やメソッドの一部が空であったり, ビューモデルでは, ベースモデルにおける不可視な関係はメタモデルとして導出されないので, それらをユーザが確認・編集する. これを図 8, 図 9に示す.

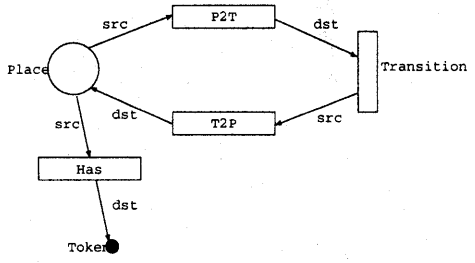

図 8: ペトリネットにおけるメタモデル (意味モデル側)

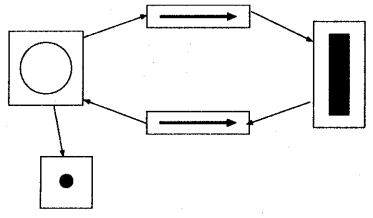

図 9: ペトリネットにおけるメタモデル (ビューモデル側)

5.4 実行モデル

ドメインモデルであるベースモデルとメタモデルの作成された後, 実際にアプリケーションモデルとなる実行モデルを作成する. この時, 環境ではドメインモデル開発フェーズからアプリケーションモデル開発フェーズへと移行しており, ユーザの交代が行われることもある.

図 10が実行モデルである. 実行モデルの作成は, ベースモデルのコピー・編集にて行われる. その際, メタモデルや, ベースモデルの意味は編集せず, ベースモデル中の要素の関連などが編集されるの

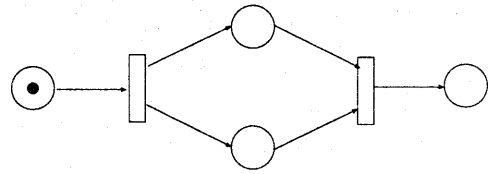

図 10: ペトリネットにおける実行モデル

で作成作業はビューモデル側でのみ行い, 意味モデル側にビューモデル側にて行われた作業を逐次反映していく. これの実行画面が図 11である.

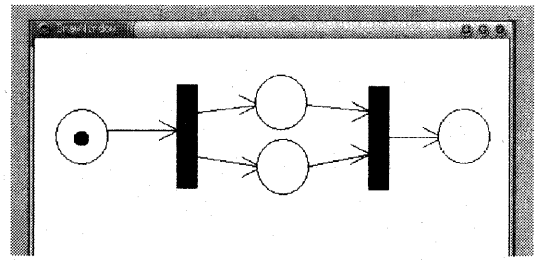

図 11: 実行画面 (実行モデル)

実行モデルが作成されたら, ユーザは実行モデルを実行する. モデル実行のメッセージを意味モデル側環境が受け取ったら, 実行モデルのフィールドにモデル実行のメッセージを送り, メタモデルに記述されたメソッドが評価・実行されていく.

実行後の結果が以下の図 12である. また, 実行画面が図 13, 図 14である.

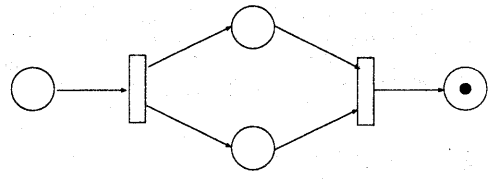

図 12: 実行後の実行モデル

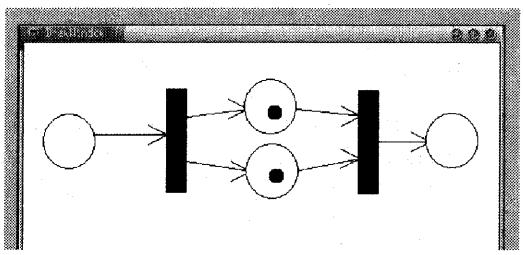

図 13: 実行画面 (実行結果その 1)

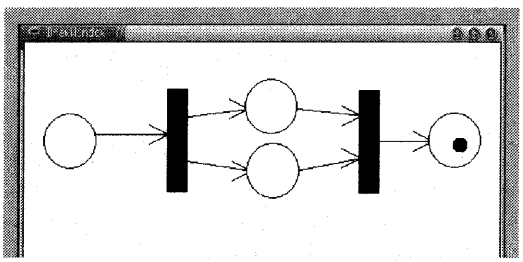

図 14: 実行画面 (実行結果その 2)

6 関連研究

本報告にて提案したモデリング環境との関連研究について比較・検討する.

6.1 庄司による研究

本研究室の庄司による研究 [4] では, ユーザによるメタモデル作成を含むアプリケーション開発環境の提案がなされた.

この環境もメタメタモデル, メタモデル, ベースモデルの 3 階層から構築され, メタメタモデルは環境側で提供する. しかし, 本報告にて提案した環境とは異なり, ソフトウェアシステム開発をドメインモデル開発とアプリケーションモデル開発には分割せず, アプリケーション開発者は, まず, メタモデルを作成し, 続いてベースモデルを開発する. メタメタモデルは実体・関連・フィールドからなる ERF で構成され, アプリケーション開発者はメタエンティティ, メタリレーション, メタフィールドを作成する. また GUI を持たず,Bramble 言語にてモデルの記述を行う.

この環境では, ユーザの分離が行われておらず, ソフトウェアシステム開発時に, アプリケーション開発についての専門性の低いユーザを含んでプロトタイピングを行う場合, 開発の全工程でメタ概念や Bramble 言語, プログラミングに対しての高い専門性が必要とされてしまうため, 開発がスムーズに進展しないことも予想される.

6.2 小林らによる研究

小林らによる研究 [5] では, オブジェクト指向プロダクト生成環境の提案がなされた.

この環境はソフトウェア開発の各工程をオブジェクトとして捉え, 前の工程による成果物に対して, 分析, 設計, 実装, メンテナンスに適用できるパターンを用い, 次の工程へ譲渡する成果物の生成を試みるものである.

ユーザが要求仕様書を入力すると, まず要求分析に適用可能なパターンが使用され, 要求分析される. 次に設計に適用可能なパターンが使用され, さらに実装に適用可能なパターンが使用されるという順序で実行される. ただし, 適用パターンは既存のものをそのまま使用するのではなく, 対象ドメインに特化したものを使用する. それゆえ前工程での適用パターンにおける特性を発展させ, 次工程での適用パターンを生成する. 適用パターンにはメタ階層概念が導入されており, ベースモデルを適用パターン, メタモデルをパターン中に用いられる図, メタメタモデルを図にて用いられるコンポーネントとしている. このうちメタメタモデルであるコンポーネントは, オブジェクト図ならばオブジェクトと連結, クラス図ならばクラス, メソッド, 属性値, 関連などと予め定義されいる. これにより適用パターンのより詳細な発展が可能である.

この環境にて使用される適用パターンは, メタモデルにて記述されている必要があり, 適用パターンがなくユーザが適用パターンの作成を試みた場合, ユーザはメタメタモデルである図のコンポーネントを使用して, メタモデルである適用パターンに用いられる図を作成することになる. これはユーザにメタ階層概念とソフトウェア開発に用いられるパターンについての高い専門性を要求しており, 一般的なユーザが適用パターンを作成することは困難であると思われる.

本研究ではベースモデルを基としたメタモデル作成を支援しており, この点で優れていると考える. また, 本研究にて提案した環境では, ユーザの分離を図っており, この点でも異なっている.

7 まとめ

本報告では, 柔軟な図式表現を可能にし, ソフトウェア開発を分割したプロトタイプベースのモデリング環境を提案した.

今後, より充実した環境となるための課題として, 以下のようなものが挙げられる.

- 階層的なモデルへの対応
  本報告中ではモデルを集約するものとしてフィールドを定義している. 定義的にはフィールドがフィールドを含むことは可能であるが, 実装時に, フィールドの持つ属性値として自分のフィールド情報がない. また, 仮にこの属性値を持たせ, フィールドが自分の親フィールド上にあるエンティティと関連を持ったとしても, 自分の親フィールド上のモデルへのメッセージ送信に環境が対応していないのである. そこを対応させたい.

- メソッド評価の方法
  今回のメソッド評価方法は必要とされる全てのモデル情報を Bramble へ送り, 評価・実行した結果を受け取るというものである. その際, 制約条件として, メソッド中には他のモデルへのメッセージ送信を含んではならないことが挙げられる. これを許可すると,Bramble 側から環境上のモデルへのメッセージ送信が発生することになるが, その時点で, 環境側へさらに必要なモデル情報を要求することとなる. 今回使用する BrambleVirtualMachine は, 自身を呼び出した関数へ結果を返すことは可能だが, それ以外の関数へのメッセージ送信は不可能となっている.

これらについて, より良い方法を検討中である.

参考文献

[1] 安間その美 : "モデルのメタ階層に基づいたソフトウェアプロトタイピング基盤機構の開発", 情報処理学会, 第 52 回全国大会, No.5,pp.65-66(1996,3)

[2] 高橋大輔 : "メタアーキテクチャに基づいたモデル構築支援環境の開発", ソフトウェア工学の基礎 III, pp.146-149, 近代科学社 (1996,12)

[3] 中野喜之 : "メタ階層アーキテクチャに基づくモデリング環境の設計と実現", ソフトウェア工学の基礎 IV, pp.71-74, 近代科学社 (1997,12)

[4] 庄司龍一 : "メタアーキテクチャに基づくプロトタイプモデル生成・解釈機構の実現", ソフトウェア工学の基礎 VI, pp.212-219, 近代科学社 (1999,11)

[5] Takashi Kobayashi, Motoshi Saeki : "Software Development Based on Software Pattern Evolution", Asia-Pacific Software Engineering Conference99(1999,12)

[6] 上田 賀一, 中野 喜之, 金村 星吉, 高橋 大輔: "オブジェクト指向モデル記述言語 Bramble の開発", 情報処理学会, 研究報告 (SE), Vol.96, No.32, pp.65-72 (1996,3)