ソフトウェア工学 94－7
（1993. 9. 21）

プロトタイピングのための モデル記述言語と支援環境

清水俊吾　上田賀一

茨城大学

〒３１６ 茨城県日立市中成沢町４－１２－１

あらまし

ソフトウェア開発規模の拡大に伴いそれに携わる人員の数も増大し、正確かつ迅速な情報伝達の方法がより一層重要となってきた。本研究では、これを実現する手段として共通のモデルを介した情報伝達を行なうことを考える。モデルを統一的に扱い、また自由に作成することを可能とするためのモデル記述言語として、ＯＥＲ（Objective Entity Relationship）モデルとＭＶＣ(Model View Controller)モデルによる上位表現系を開発する。さらに、作成したモデルを実行させプロトタイピングを容易に行なうための、実行支援環境を構築する。

和文キーワード　ＯＥＲモデル、 ＭＶＣモデル、 上位表現系、 ＣＡＳＥ、 プロトタイピング

A Model Description Language and its Supporting System for Prototyping

Syungo SHIMIZU　Yoshikazu UEDA

Ibaraki University

4-12-1, Nakanarusawa, Hitachi, Ibaraki, 316 JAPAN

Abstract

As the scale of the software development becomes larger, the number of technical experts who are engaged in it increase, and the correct and rapid communication methods become more important. Now we consider using a common model to communicate. In order to enable to treat various models consistently and to make them freely, we develop the meta model based on Objective Entity-Relationship model and Model-View-Controller model as the executable model description language. Additionally, we build the supporting environment to make and simulate models for prototyping.

英文 key words　OER model, MVC model, Meta model, CASE, Prototyping

1．はじめに

今日のソフトウェア産業はあらゆる面において巨大化しつつある。これは、ハードウェア性能の向上に伴うユーザニーズやソフトウェア適用分野の増大に起因する。これによるソフトウェア開発規模の拡大とともにそれに携わる人の数も増大し、正確かつ迅速な情報伝達の方法がより一層重要となってくる。

従来より、ソフトウェア開発過程の各段階において様々なモデルを作成し、そのモデルを介して情報伝達を行なう方法が採られてきた。モデルを作成することで各段階で必要な側面のみを捉えることが可能となり、複雑性を減少させ問題領域の全体構造の把握が容易にできるという点からモデルを用いることは非常に有効な手段であるといえる。

モデルの表記法としては言語表記や図式表記などが存在するが、ソフトウェア開発においては図式表記、特にグラフが多く用いられている[5]。これは、言語表記よりも直感的、論理的に全体構造を把握しやすいためである。状態遷移図やデータフロー図などがこれにあたる。本研究では、これらのグラフを統一的に扱うこと及び自由に個々のグラフを作り出せることを考える。

さらに、記述したモデルを実行可能とし、モデルをアニメーション表示させるなどしてプロトタイピング[3]が容易にできることを考える。プロトタイピングを実施することで開発対象システムの妥当性の確認や実現性の評価を行なうことができ、これによりソフトウェア開発の支援につながると思われる。本研究では、これを実現するためにオブジェクト指向の概念を導入し、実行可能なモデル記述言語を開発する。また、実行時の視覚的表現を支援するものとしてＭＶＣモデル[4]の概念を取り入れ、充実したモデルの構築・実行支援環境を開発する。

2．グラフベースなモデル

状態遷移図やデータフロー図などのグラフベースなモデルは、形状や線の種類に多少の違いはあるが、Node（節点）とArc（弧）の２種類の要素から構成されているといえる。本研究ではこれらのモデルを総称してＮＡモデル（Node Arc Model）[1][2]と呼ぶことにする。図１に、ソフトウェア開発で多く用いられる代表的なＮＡモデルを示す。

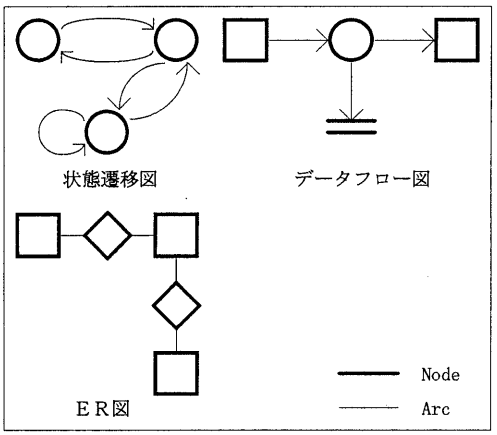

図１．ＮＡモデルの例

3．ＯＥＲモデル

ＮＡモデルを実際に計算機上で実現するためには、ＮＡモデルを表現するためのモデル（記述言語）が必要である。本研究では、ＮＡモデルのNodeとArcに対応づけることが可能であり、複雑な概念を必要としないＥＲモデル（Entity Relationship Model）の導入を考えた。

ＥＲモデルは、実世界に存在するEntity（実体）とEntity間に存在するRelationship（関連）という２つの概念を用いて実世界の意味構造や制約条件をより直感的かつ自然に表現するモデルである。また、ＥＲ図により視覚的な表現が可能である。

しかし、ＥＲモデルには動的側面を表現する能力が無く、本研究の目標である実行可能なモデル記述が行なえない。そこで、ＥＲモデルにオブジェクト指向の概念を取り入れ、動的側面を内包するモデルに拡張し、ＯＥＲモデル（Objective Entity Relationship Model）[1][2]とした。オブジェクト指向によりデータ抽象化と継承という概念が利用できるので、モデル構築の労力を減らすことが可能である。

ＯＥＲモデルでは、EntityおよびRelationshipがそれぞれ属性（attribute）と動作（action）を

持つオブジェクトである。されに特殊な属性として、Entityはアイコン（icon）をRelationshipは線種（line style）を持ち、ＮＡモデルを作成する際の視覚的な識別の役割を果たす。

４．ＭＶＣモデル

プロトタイピングを行なうことを念頭に置いてモデルを記述する時、EntityやRelationshipには記述対象となるモデルの属性や動作に加え、モデル実行時にのみ必要となる属性や動作も記述する。これは、モデル全体としての構造や動作の確認を行なう際には問題はないが、モデル内の個々のEntityやRelationshipに視点を移すと属性や動作が記述対象となるモデルのものなのかモデル実行時にのみ必要なものなのかが分かりにくく、モデル化の意味がなくなってしまう。さらに、EntityやRelationshipに対する変更の数も増大し、モデル記述に対して大きな負担となる。

本研究では、この問題を回避するためにＭＶＣモデル（Model-View-Controller Model）の概念を取り入れ、記述対象となるモデルをModel, View, Controllerの３つの観点からそれぞれのオブジェクトを定義し、３つのオブジェクトが協調しあう形で表現する。

ＭＶＣモデルの３つの観点の目的については以下のとおりである。

- Model
  記述対象となるモデルそのものの構造や動作を表す。
- View
  モデルの実行時に、視覚的表現によってユーザへ情報を提供する。
- Controller
  ユーザからの入力を解釈し、Modelあるいはviewにメッセージを送り、適切な調整を施す。

図２は、Model, View, Controllerの関係を示している。基本的にModelは視覚的な表現のための情報を持たず、ユーザからの入力のために特化されたメッセージも受け付けない。また、ＭＶＣのControllerのみを他のControllerに置き換えれば、表示内容はそのままでユーザとのインタラクションだけを変更することができる。さらに、同じModelを異なった形式で表示したり操作したりするために、違ったViewとControllerのペアを追加することも容易である。

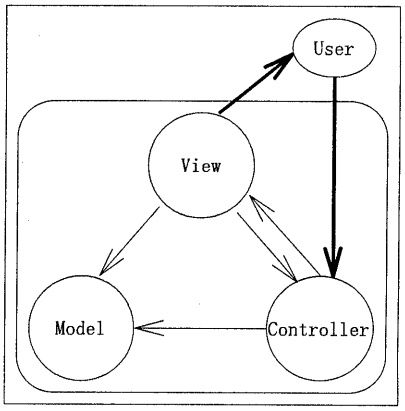

図２．ＭＶＣモデルの枠組み

本研究では先のＯＥＲモデルとＭＶＣモデルを融合してモデルを記述するためのモデルすなわち上位表現系とし、Model, View, Controllerをそれぞれ上位表現系より記述したＮＡモデルを用いて記述する方式を採る。

ＮＡモデル，ＯＥＲモデル，ＭＶＣモデルの関係を図３に示す。

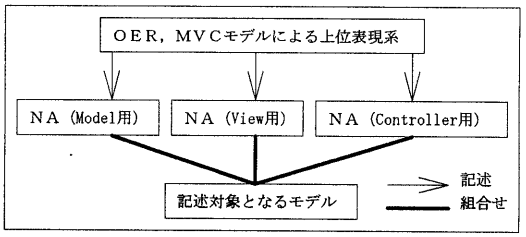

図３．ＮＡ，ＯＥＲ，ＭＶＣモデルの関係

５．環境オブジェクト

記述したモデルを実行する際に、時間や空間、またオブジェクト間の通信を管理する機能が必要で

ある。本研究ではこれらを実現するために、環境という観点で動作するオブジェクトを考え、これらのオブジェクトを用いて実行時の支援を行なう。

5．1 時間管理オブジェクト

モデルの実行時には、時間に関する情報が必要である場合が考えられる。例えば、処理時間を考慮に入れたモデルを作成する場合や、ある周期ごとに動作を行なうオブジェクト、ある時間に起動するオブジェクトなどを作成する場合である。

本研究では、時間を管理する環境オブジェクトとしてＴＭＯ（Time Managing Object）を設計し、オブジェクトからの問い合わせに対して時間に関する情報を提供する機能を実現した。

5．2 メッセージ管理オブジェクト

モデルの実行は、オブジェクト間のメッセージ通信により行なわれる。メッセージ通信には同期通信と非同期通信の２つの種類がある。同期通信はメッセージの送り先の動作を直接呼び出すという形で実現できるが、非同期通信はメッセージを保持しておく必要がある。このために、非同期通信のメッセージ用のメッセージキューを備えた環境オブジェクトＳＶＯ（Supervisor Object）を設計した。

また、メッセージに優先順位を付けることでより柔軟な記述が行なえると考え、ＳＶＯにはメッセージの優先順位に従いメッセージキュー内のメッセージをソートする機能を付加した。

5．3 空間管理オブジェクト

モデル実行時においてオブジェクトがメッセージ通信を行なう際、メッセージの送り先の情報が必要となる。このために、モデル内の全オブジェクトの接続情報を保持する環境オブジェクトが必要である。本研究ではこのオブジェクトをＳＭＯ（Space Managing Object）とた。これは、オブジェクト間の接続情報を管理し、モデル内のオブジェクトからの問い合わせに対してオブジェクトの接続情報を提供するオブジェクトである。

5．4 プラグ

Model, View, Controllerを記述するＮＡモデルにおいて、外部とメッセージの送受信を行なう必要のあるNodeが存在する場合がある。このようなNodeに対してはプラグという概念を用いて、メッセージの送受信を行なう。プラグはModel, View, Controllerのそれぞれに１つずつ存在し、これらの間のメッセージ通信のためのインタフェースとなる。本研究では、内部（ＮＡモデル）と外部（ＭＶＣモデル）間の通信を中継する環境オブジェクトとして、Ｐｌｕｇを設計した。Model, View, Controller間のメッセージ通信は全てこのＰｌｕｇを介して行なわれる。図４にＰｌｕｇの概念図を示す。

また、オブジェクト間の接続情報には、Model, View, Controller間と、これらを記述するのに用いられるＮＡモデルのNode, Arc間の２つの異なるレベルの接続情報が存在する。そこでＳＭＯをこれらのために特化し、Ｐｌｕｇ－ＳＭＯとＮＡ－ＳＭＯの２つを設計した。

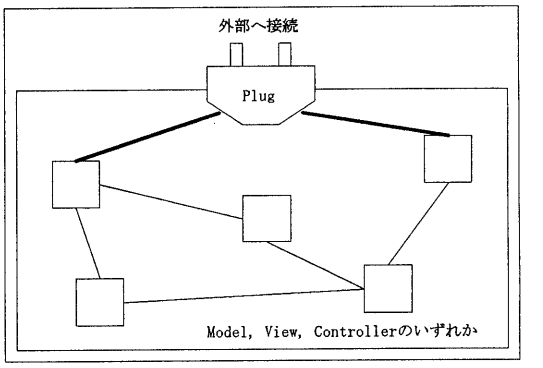

図４．Ｐｌｕｇの概念図

6．実行の仕組み

モデルの実行は、オブジェクト間のメッセージ通信によりなされることは先に述べたとおりである。従って、メッセージに注目してモデルの実行の仕組みを説明する。

6．1 同期通信の場合

同期通信は、メッセージの送り先となるオブジェクト内の動作をあらかじめ知っていなければならないので、あまり汎用的な通信方式ではない。しかし、即座に効果を要求するような場合には有効である。

動作の手順は以下のとおりである。

● Node, Arc間及びＰｌｕｇ とNode, Arc間

1）ＮＡ－ＳＭＯに接続情報を要求する
2）接続情報に従い、メッセージの送り先となるオブジェクトを決定する
3）送り先のオブジェクトに存在する動作を直接呼び出す

● Ｐｌｕｇ間

1）Ｐｌｕｇ－ＳＭＯに接続情報を要求する
2）接続情報に従い、メッセージの送り先となるオブジェクトを決定する
3）送り先のオブジェクトに存在する動作を直接呼び出す

6．2 非同期通信の場合

非同期通信では、メッセージがいったんＳＶＯのメッセージキューに格納され、ＳＶＯによりメッセージが送り先のオブジェクトに送られる。メッセージには優先順位を付けることができるので、例外処理などの緊急なメッセージも実現可能である。

動作の手順は以下のとおりである。

● Node, Arc間及びＰｌｕｇ とNode, Arc間

1）ＮＡ－ＳＭＯに接続情報を要求する
2）接続情報に従い、メッセージの送り先となるオブジェクトを決定する
3）ＳＶＯに対して、メッセージの送り先オブジェクトとメッセージの種類及び優先順位を伝える
4）ＳＶＯはメッセージの優先順位に従いメッセージキューをソートする
5）ＳＶＯはメッセージキューの先頭にあるメッセージを送り先のオブジェクトに対して送信する
6）メッセージを受信したオブジェクトはメッセージの種類にしたがって処理を行なう

● Ｐｌｕｇ間

1）Ｐｌｕｇ－ＳＭＯに接続情報を要求する
2）接続情報に従い、メッセージの送り先となるＰｌｕｇを決定する
3）Ｐｌｕｇ用のＳＶＯに対して、メッセージの送り先Ｐｌｕｇとメッセージの種類及び優先順位を伝える
4）Ｐｌｕｇ用ＳＶＯはメッセージの優先順位に従いメッセージキューをソートする
5）Ｐｌｕｇ用ＳＶＯはメッセージキューの先頭にあるメッセージを送り先のＰｌｕｇに対して送信する
6）メッセージを受信したＰｌｕｇはメッセージの種類にしたがって処理を行なう

図５に実行時のメッセージ通信の様子を示す。

7．システム構成

本研究では、図６に示すようなシステムを考えている。

本システムには、５つのツールと４つのデータファイル，そしてアクションパートライブラリが存在する。

次にシステム構成要素のそれぞれについて解説を行なう。

● ＮＡモデルデータファイル

EntityとRelationshipのデータを格納する。
このファイルは、モデル構築環境中だた一つ存在する。

● メタモデルデータファイル

メタモデルに存在するEntity, Relationshipの集合を格納する。
このファイルは、メタモデルごとに一つずつ存在する。

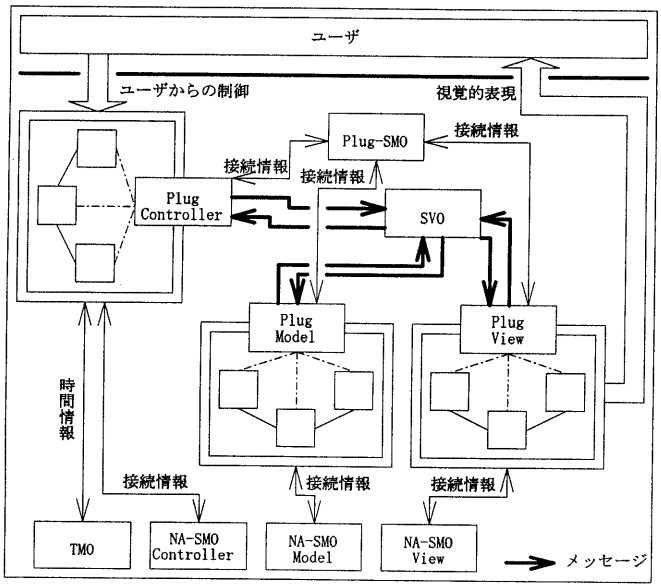

図５．実行時のメッセージ通信

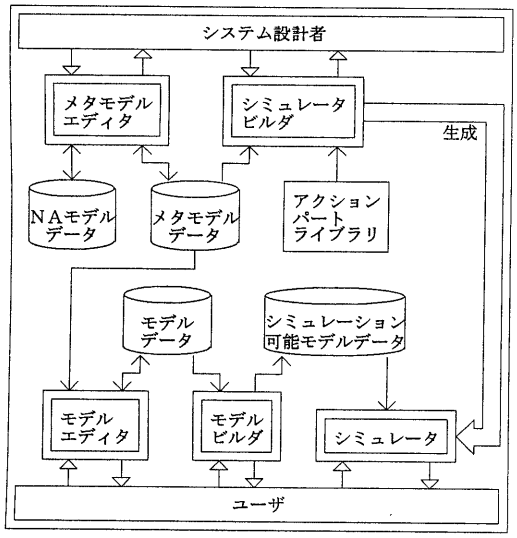

図６．システムの構成

● モデルデータファイル

作成されたモデルのデータを格納する。

Model, View, Controllerのそれぞれに必要となる。

このファイルは、作成したモデルごとに一つずつ存在する。

● シミュレーション可能モデルデータファイル

シミュレーション可能となったモデルのデータを格納する。

このファイルは、シミュレーション可能なモデルごとに一つずつ存在する。

● メタモデルエディタ

モデル記述に必要なEntity, Relationshipの作成を支援する。

作成されたデータは、ＮＡモデルデータファイル及びメタモデルデータファイルに格納される。

● シミュレータビルダ

ある特定のメタモデルに対応するシミュレータを生成する。この時にアクションパートライブラリと結合する。

● モデルエディタ

Model, View, Controllerのそれぞれを、メタモデルエディタにより作成されたメタモデルを用いて記述するための支援を行なう。

作成されたModel, View, Controllerはモデ

ルデータファイルに格納される。

● モデルビルダ

Model, View, Controllerを組み合わせ、シミュレーション可能なモデルの作成を支援する。

作成されたデータは、シミュレーション可能モデルデータファイルに格納される。

● シミュレータ

シミュレーション可能なモデルのデータを読込み、実行させてシミュレーションを支援する。

また、モデルの属性の参照・変更や実行の制御（始動・停止・一時停止など）を可能とすることでシミュレーションの充実を図る。

● アクションパートライブラリ

シミュレータを生成する際に結合される、モデル記述に必要なデータ型やデータ構造などのオブジェクトを集めたクラスライブラリである。

具体的には、以下の４つに大別できる。

- 上位表現系
  ＯＥＲモデル，ＭＶＣモデル，環境オブジェクトを提供する部分。
- 基本データ型
  基本となるデータ型を提供する部分。
- 構造データ型
  データ構造を提供する部分。
- グラフィック
  グラフィック表示やユーザインタフェースに必要なオブジェクトを提供する部分。

8．モデル記述の手順

本システムでは、システム設計者とユーザの作業を切り分けている。こうすることで、お互いが未開の領域に足を踏みいれることなく自分の担当の作業に専念でき、混乱を招くようなことがなくなると考えられる。

次に、システム設計者とユーザの作業手順を示す。

● システム設計者の作業

1）メタモデルエディタにより、ＭＶＣモデルのModel, View, Controllerのために特化されたEntity, Relationshipを作成し、またそれらの中からいくつかを選択してＮＡモデルを作成する。ここにおけるＮＡモデルが、メタモデルとなる。

2）シミュレータビルダを使いある特定のメタモデルに対するシミュレータを生成する。

● ユーザの作業

1）モデルエディタを使い、作成されたメタモデルを用いてModel, View, Controllerごとのモデルを作成する。

2）モデルビルダで、Model, View, Controllerのそれぞれのモデルから１つずつ選択し、シミュレーション可能なモデルを生成する。

3）必要に応じてモデルをシミュレータにかけることにより、その動作をシミュレーションやアニメーションして確認することができる。

9．まとめ

本研究では、様々なグラフを統一的に扱え、汎用性を有したＮＡモデル，ＯＥＲモデル，ＭＶＣモデルを示し、またモデルの記述・実行を支援する環境を示した。以下に、本研究の主要な特徴を挙げる。

● 種々のモデルを統一的に扱える

ソフトウェア開発過程の各段階ではその段階に適したモデルを用いるので、開発の段階が移る際にモデルの変換が必要となり、共通で

あるべき認識や理解に違いが生じてしまう可能性がある。本研究で作成したシステムでは、種々のモデルを統一的に扱うことを考えているので、この問題を回避することが可能である。

- モデル記述がＭＶＣモデルに基づいている
  モデルの本質的な部分とモデル実行時にのみ必要となる部分を同じ場所に記述した場合、必要以上にモデル内部の変更がなされる恐れがある。そこでＭＶＣモデルの概念を導入し、モデルの本質的な部分と実行時にのみ必要となる部分を分離して記述し、モデルを実行する際にそれぞれを組み合わせることとする。これにより効率的なモデル記述が行なえる。また組合せを変えることで実行時の見え方を変えることができるので、充実した実行の支援を行なうことができる。

- プロトタイピングが容易に行なえる
  作成したモデルを実行させることで、詳細設計段階や実現段階まで移行する前に動作の確認を行なうことが可能であり、本研究で作成したシステムでプロトタイピングを支援することが可能である。

以上のように、ソフトウェア開発に携わる人員間の認識や概念の一致を図ることを主眼において研究を進めてきた。本研究では、モデルを記述しそのモデルを実行することで、これを実現することを考えた。従って、効率の良いモデル記述が行なえることが必要条件となるが、本研究では不十分であると思われる。研究の今後の課題として、より良いモデル記述の効率を目指すために、以下に示すような考えを本システムに取り入れることを考える。

① 継承を用いたＮＡモデルの記述

現段階でのＮＡモデルの記述には、オブジェクト指向という観点から見るとカプセル化の機能しか用いられてない。これに加えて、継承の概念を取り入れ、差分を記述するだけで新しいNodeやArcを作りだすことを考える。継承により、ＮＡモデルを作成する際に記述する項目が少なくなり、モデルの効率的な記述につながると思われる。

② モデルの部品化

NodeやArcをつなげてモデルを作成する際、ある機能や特徴でまとめられたモデルの部品が存在すると便利である。このモデルの部品を特殊なNodeとして考えると、モデルの作成に対しては新しい知識を必要とせず、またNodeやArcの数が少なくなるのでモデルの全貌がつかみやすくなり、モデル作成時の負担を軽減すると思われる。

参考文献

[1] 大野賢二，清水俊吾，上田賀一：メタモデルに基づくモデル記述とシミュレーション，情報処理学会第４４回全国大会 4J-8 (1992).

[2] 清水俊吾，丸山健，上田賀一：実行可能なモデル記述のための上位表現系の開発，情報処理学会第４６回全国大会 5J-3 (1993).

[3] 竹下　亨：ＣＡＳＥ概説，共立出版 (1990).

[4] 青木　淳：オブジェクト指向システム分析設計入門，ソフト・リサーチ・センター (1993).

[5] 河村一樹：ソフトウェア工学入門第２版，啓学出版 (1992).