ソフトウェア工学 102-11
（1995. 1.26）

オブジェクト指向仕様記述の実行・検証系：ROAD/EE

田村直樹、中島毅、柳生理子、萩原正敏
三菱電機（株） 情報システム研究所

要求分析作業では、顧客の要求を漏れなく正確にしかも早期に仕様化することが求められる。我々は、プロトタイピング手法とオブジェクト指向分析・設計手法を融合することで、この要求抽出と仕様化の作業を支援するROAD/EE（ Real-time Object-oriented Analysis and Design / Execution Environment）の開発を進めている。本環境では、仕様記述モデルとしてRumbaughらが提案するOMT法を用いる。OMT法で利用される複数の仕様記述モデル間の整合性をチェックすると共に、仕様記述を直接実行し、仕様の静的検証、動的検証を支援する。さらに、仕様記述を基にオブジェクトを動作させることで、仕様記述と一体化したプロトタイピングを可能としている。

本報告では、ROAD/EEの基本的アプローチと機能について述べる。

Prototyping Environment for Object-Oriented Specifications: ROAD/EE

Naoki TAMURA, Tsuyoshi NAKAJIMA, Riko YAGIU, Masatoshi HAGIWARA

Computer & Information Systems Laboratory
Mitsubishi Electric Corporation
5-1-1 Ofuna, Kamakura, Kanagawa, 247 Japan

The goal of requirement analysis is identifying all of customers' requirements and describing a consistent specification of those requirements. To archive these goals, we developed a prototyping environment named ROAD/EE (Real-Time Object-Oriented Analysis and Design / Execution Environment). In this environment, requirement specification is described with Object-Oriented specification models of the OMT method. The environment provides the facility of checking the consistency of specification descriptions and testing the behavior of the specifications with executing the specification descriptions directly .Also the environment supports building prototypes from given specifications, to visualize the behavior of the systems.

In this paper, we explain the concept of this prototyping environment and its facilities.

1．はじめに

ソフトウェアシステムに求められる要求は、近年ますます高度化、複雑化している。このため、システムに対する顧客の要求の仕様化を、いかに漏れなく正確にしかも早期に行えるかは、プロジェクトの成否を決定する大きな要因となりつつある。

要求分析・定義の段階の作業は、顧客の要求を抽出する作業と抽出された要求を仕様化する作業からなる。前者には様々なインタビュー法[1]やプロトタイピング手法が、後者にはある程度形式的な仕様記述モデルを用いる分析・設計手法がある。

プロトタイピング手法は、顧客の要求を素早く抽出する手法としてその有効性が広く認識されている。しかしプロトタイプを作成する上で、特殊なプログラミング言語を用いる場合が多く、その場合プロトタイプ開発が実システムの開発に対して付加的な作業となるため、開発現場からは敬遠されがちである。

一方、仕様記述モデルを用いる手法として、近年オブジェクト指向分析・設計手法[2,3]が注目されている。これらの手法が提供する仕様記述モデルは、複数の抽象化の視点からシステムの仕様を記述でき、またダイアグラムであるため比較的記述が容易である点で、仕様を記述する良い枠組みを提供する。しかし、一般にこれらの仕様記述モデルからシステムの動作イメージを理解することが難しいため、顧客参加の仕様のレビューに利用するにはで大きな壁が存在する。

我々は、プロトタイピング手法とオブジェクト指向分析・設計手法を融合し、上に示したような各々の手法の問題点を解決することを狙っている。具体的には、仕様記述モデルとして、Rumbaughらが提案するOMT法を用い、モデルの入力と実行とプロトタイピングを支援する環境 ROAD/EE （Real-time Object-oriented Analysis and Design / Execution Environment）の開発を進めている。本稿では、現状の仕様化技術が持つ問題を明らかにしROAD/EEの基本的アプローチと機能について述べる。

2．現状の仕様化技術が持つ問題

2.1 仕様記述モデルが持つべき要件

要求仕様を作成するプロセスは、図1に示すように仕様の記述・レビュー・検証の3つの作業の繰り返しとなる。仕様の記述では、システム分析者が顧客から抽出した要求仕様を仕様記述としてまとめる。仕様のレビューでは、顧客がシステム分析者がまとめた仕様

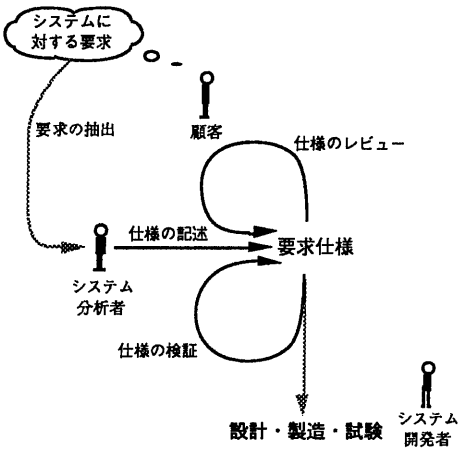

図1　要求の仕様化作業

を見て自分の求める要件との相違点を指摘する。仕様の検証では、システム分析者が最終的に得られた要求仕様の仕様記述に曖昧さや矛盾、抜けが無いことを確認してシステム設計者に渡す。

仕様記述モデルは上記の作業を媒介する役割を持つ。従って、次の要件を備えていることが要求される。

(1) 顧客の様々な要求を適切に表現するための高い記述能力と、徐々に仕様を詳細化したり簡単に修正することが可能な記述容易性

(2) 仕様の計算機の非専門家である顧客でも理解できる理解容易性

(3) 仕様の抜けや矛盾がないことを検証することが出来る検証容易性

2.2 現状の仕様化技術の課題

現在広く利用されている仕様化技術としては下記のものがある。各々の方法の利点と欠点を、上記の要件に照らして検討してみる。

(1) 自然言語を主体とした仕様記述

自然言語による仕様記述は、現在多くのプロジェクトで採用されている。自然言語を用いた仕様記述は、特別な知識を前提とせずに理解することが出来るという特長を持つが、自然言語の持つ曖昧さから仕様の漏れや矛盾、人による解釈の相違を取り除くことが難しい。

(2) 形式的な仕様記述

ある程度形式的な仕様記述を採用することにより、できるだけ正確にシステムの仕様を作成しようとするアプローチがある。例えばオブジェクト指向分析・設計手法として広く着目されているOMT法[3]がそのひとつである。OMT法が提供する仕様記述モデルは、構造的側面、動的側面及び機能的側面の複数の側面

からシステムを捉えることで仕様の記述能力も高く、また図的な記法であるので比較的記述も容易である。さらにCASEツールを用いれば、各モデル記述内での文法チェックも容易となる。

しかし、OMT法では、得られた仕様記述からシステムの具体的な動作イメージを把握することが難しいため、この仕様記述モデルを顧客がレビューをする目的で利用することは困難である。

(3) プロトタイピング手法

プロトタイピング手法は顧客の要求を漏れなく抽出する有効な手法である。プロトタイプは、顧客に対して実際のシステム動作イメージを与え、顧客の潜在的な要求を顕在化するからである。

しかし、一般にプロトタイプの作成には、特殊なプロトタイピング言語が用いられるため、プロトタイピングの結果をそのまま後工程へのベースラインドキュメントとして利用できない。この結果、プロトタイプの作成は本来の仕様化作業に対して付加的な作業となり、このため現場では敬遠されがちとなっている。

3．オブジェクト指向仕様記述の実行検証系ROAD/EE

3.1 ROAD/EEのアプローチ

我々は、記述能力と記述容易性を両立させた形式的な仕様記述モデルから直接プロトタイプを作成することによって、要求の抽出と仕様化の作業を融合し、要求分析作業の品質を向上させることを狙っている。このための支援系としてROAD/EE の開発を進めている。

ROAD/EEでは、仕様記述モデルとしてOMT法の記述モデルを用いる。またOMT法では複数の仕様記述モデルを用いるが、ROAD/EEではモデル間の整合性チェックを行うと共に、これらのモデル間にいくつかの仮定をおくことによってモデルを実行可能としている。これによって、仕様の静的・動的検証を可能にしている。さらに、プロトタイピングを行うために仕様記述に現れるオブジェクトのアイコンを定義し、これを画面上で動作させる支援環境を提供している。

以下、次節ではROAD/EEの各構成要素がどのような機能を表すかについて述べる。

3.2 ROAD/EEの構成

ROAD/EEは、次の構成要素からなる（図2）。

(1) 仕様記述エディタ群
(2) ツールマネージャ
(3) 仕様記述実行系
(4) プロトタイパ

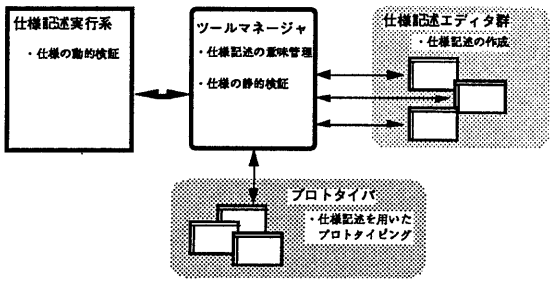

図2　ROAD/EEのシステム構成

これらは、環境の機能拡張を容易にする目的で、複の独立したツールとして構成されている。

(1) 仕様記述エディタ群

ROAD/EEでは、OMT法の仕様記述モデルに対応した仕様入力用エディタを提供している。現在、オブジェクト図エディタ、状態図エディタ、イベントフロー図エディタ等を用意している。なお、これらのエディタは全てCASE環境構築用ツールキットCASEmaker[4]を用いて開発されており、仕様記述モデルの拡張にも容易に応えられる構成となっている。

(2) ツールマネージャ

ツールマネージャは、複数の仕様記述エディタから入力される仕様記述から意味データを抽出し管理する。また、エディタから登録される意味データを基に、複数の仕様記述モデルにまたがる整合性のチェックを行う。さらに、エディタやプロトタイパからの様々な問い合わせを処理したり、仕様記述実行系の実行を制御し、実行結果をエディタ、プロトタイパに通知する。

(3) 仕様記述実行系

仕様記述実行系は、ツールマネージャによって意味的に統合化された仕様を実行する部分である。実行系は、入力された仕様記述モデルを基に、インスタンスを作成し、各々のインスタンスを状態図に基づいて動作させる。この際、状態図中に現れるアクション、アクティビティ等の記述を基に各種演算も行う。更に、複数のインスタンスが並行に状態遷移を起こした時の非決定性の検出（複数の可能な状態遷移の発生、変数の同時更新）といった動的検証も行う。

(4) プロトタイパ

ツールマネージャが管理する仕様記述を基にプロトタイピングを行うツールである。プロトタイプは、仕様記述エディタから入力されたクラスや状態に対してアイコンを与え、これらのアイコンを仕様記述

実行系の実行結果に基づいて動かすことで実現する。また、動作中のインスタンスに対して属性値を編集したり、外部イベントを入力するためのコントロールパネルをクラス毎に定義でき、インスタンスに直接働きかけることを可能としている。プロトタイパは、これらの機能を提供するため、アイコンの編集定義機能、コントロールパネルの編集機能を用意している。また、仕様記述実行系の実行を制御する機能も提供し、仕様記述の詳細な検証を行うことも可能である。

4．仕様実行のメカニズム

4.1 仕様記述の取り扱い

ROAD/EEでは、OMT法の記述モデルを実行可能とするために、いくつかの仮定を設けている。ここではそれらの仮定について述べる。基本的には、インスタンスを属性値と状態値を持つ独立した状態機械として考える。ここでは、OMT法の仕様記述モデルとインスタンスとの対応関係を示す。

(1) クラスと状態

クラスは、論理的にひとつだけ状態図を持つことが出来る。ひとつのクラスのインスタンスは、そのクラスが持つ状態図に従って状態遷移を起こす。我々のひとつの仮定は、継承関係にある2つのクラス間で共に状態図が定義されている場合、状態図の継承は行わないものとすることである。これは、例えば図3のように、下位クラスでは新たな状態遷移が追加されるなどして、動的モデル中の状態の意味が異なってしまうためである[5]。

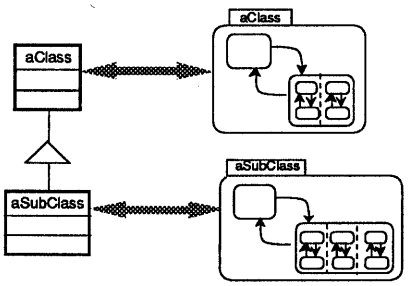

図3　クラス階層と状態図の定義の関係

(2)インスタンシエーション

インスタンスは、各々個別に状態値と属性値を持つ。状態値はそのインスタンスのクラスの状態図上でのアクティブ状態を一意に決めるものであり、属性値はそのインスタンスのクラスとそのスーパークラスの全ての属性に対する値スロットである。

またROAD/EEでは、関連のインスタンスであるリンクの生成は、インスタンスの生成と独立に行う。これは、あるクラスのインスタンスは複数同時に存在し、例え1:1の関連であろうとリンクの接続先インスタンスを一意に決定できないためである。また、インスタンス間でのイベントの送信はリンクをアクセスパスとして行われる（(3)参照）。

(3) イベントの取り扱い

イベントは、基本的にブロードキャストされるものとして扱う。特定のインスタンスに対してメッセージを与えたい場合は、明示的に送り先インスタンスを指定する。この送り先の指定には、役割名を用いることも可能である。この場合、その役割名で接続されているインスタンス群にのみイベントを配送する。

(4) アクションとアクティビティ

ROAD/EEでは、アクティビティには実行に時間を消費する処理を、一方アクションには実行に時間を消費しない処理が割り当てられることを仮定する。また、時間に対して連続的に変化する属性値の計算はアクティビティ内で記述することとしている。

(5) クラスのサービス

状態図中のアクションやアクティビティからは、クラスのサービスを起動することが出来るとする。起動されるサービスは通常のオブジェクト指向言語と同様のスコープ規則で実行時に決定される。

4.2 連続時間の実行

我々は、ROAD/EEの支援対象のひとつとして、組み込み系制御システムを想定している。この種のシステムは、時間経過に伴ってシステムの外部環境が変化し、この変化を受けてシステムが動作する。ROAD/EEではこのような時間によって変化するシステムの動作をモデリングし、これを実行することが要求される。

このため、ROAD/EEでは図4に示す離散時間モードと連続時間モードという2つの実行制御モードを備えている。仕様記述の実行は、この2つのモードを切り替えながら行われる。この実行制御は、O.Malerらが提案する数学モデル[6]に基づいている。

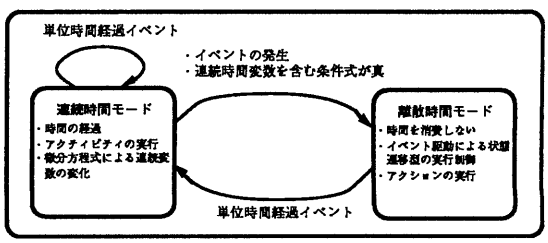

図4 ROAD/EEの実行モード

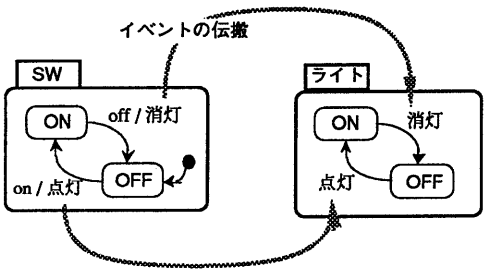

図5 状態遷移の連鎖

・離散時間実行モード

図5に示す状態図は、SWオブジェクトが「消灯」イベントを上げ、それを受けてライトオブジェクトが状態遷移を起こす様子を示している。離散時間実行モードは、このような状態遷移の連鎖を実行するモードである。ROAD/EEは、SWオブジェクトにoffイベントを与える等の外部からイベントが入力された場合、あるいは連続値がある敷居値を越えて状態遷移を起こす条件が満たされた場合に、この実行モードへ遷移する。

状態遷移の連鎖の過程で起動されるアクションの実行には時間を消費しないという仮定を置いている。

・連続時間実行モード

図6に示す状態図では、「自動車」オブジェクトが加速度中にあることを示している。この時、自動車の速度や現在位置は時間によって刻々と変化していく。連続時間実行モードはこのような時間に従ったオブジェクトの振舞いを実行するモードである。ROAD/EEは一定時間の経過を示す単位時間経過イベントが発生することにより、この実行モードに遷移する。外部から新たにイベントが入力されない限り、このモードでの実行が続く。ここでは、状態遷移は起きず特定の状態に留まっており、この状態に割り当てられたアクティビティが継続的に実行される。この過程で、オブジェクトの属性値が変更される。

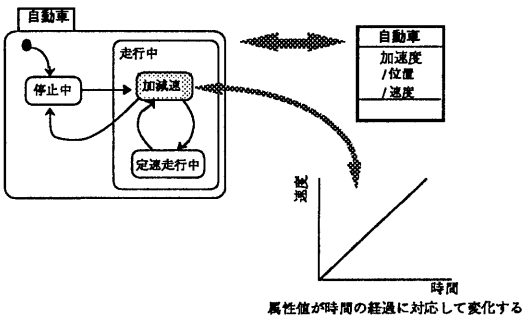

図6 連続値変数の変化

5．ROAD/EE上でのプロトタイピング

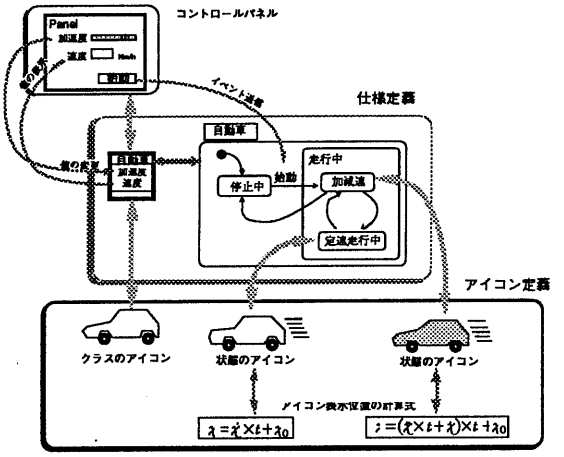

図7 アイコン、コントロールパネルと仕様記述の対応

5.1 アイコンとコントロールパネルの作成

ROAD/EEのプロトタイピング機能の基本的なアイデアは、形式的な仕様記述モデルに対してその動作を表現する「アイコン」を与えることで、システムの動作イメージを表現しようというものである。

ROAD/EEでは、図7に示すような対応関係で仕様記述モデルに対してアイコンを設定可能としている。

・*クラス、状態に対応したアイコンを作成する*

プロトタイパ上で動作を確認したいオブジェクトに対してアイコンを設定する。例えば、自動車クラスに対して1個のアイコンを設定する。また、自動車クラスの状態「定速走行中」及び「加減速」の各々にそれを視覚的に表現するアイコンを定義する。

また、各アイコンに対して表示位置を決定する計算式を定義する。プロトタイプ実行時にはこの計算式を用いてアイコンの表示位置を決定する。

・*クラスに対応するコントロールパネル画面を定義する*

プロトタイプでは、システムの動作に対してユーザが与える入力の影響を知ることも重要である。このため、ROAD/EEでは「コントロールパネル」と呼ぶユーザインタフェースを用意する。個々のクラスによって利用できる属性値が異なるため、このコントロールパネルは各クラスに対応して定義する。ユーザは、コントロールパネルを通じて、このクラスのインスタンスの属性値を参照、設定し、またインスタンスに対してイベントを与える。

さらに、クラス定義を中心に、状態図、アイコンの定義、コントロールパネルの定義をまとめて、蓄積することで、仕様記述の部品化を図ることが可能となる。ROAD/EEでは、各クラスの仕様記述を仕様ライブラ

リとして蓄積して行くことによって、仕様記述の再利用を図ることを狙っている。特定の業務分野（ドメイン）に対して複数のシステムの分析を繰り返す過程でこの仕様記述ライブラリを充実させることで、ドメイン知識の集積を行ない、顧客の業務知識を反映した要求仕様化の作業をより円滑に進められると考えている。

5.2 プロトタイプの作成

仕様に対してアイコンとコントロールパネルを作成した後、システムの動作を模擬するプロトタイプの作成を次の手順で行う。

- *プロトタイパ上でクラス定義を基に対応するインスタンスを作成し、初期値、初期状態の設定を行う。*
  作成されたアイコンを選択し、プロトタイパの実行画面上に配置することで、インスタンスが生成される。これに伴って、インスタンスに対応したコントロールパネルが生成されるので、この上で各々のインスタンスの初期値、初期状態を設定する。
- *プロトタイプを実行する*
  インスタンスの実行は、対応するクラスの状態図に沿って実行を行う。インスタンスの状態値が変更されれば、インスタンスのアイコンはその状態に対応するアイコンに置き換えられる。これによって、インスタンスの状態をプロトタイパ画面上で確認することが出来る。状態にアイコンが設定されていない場合は、クラスに設定されたアイコンが表示される。またインスタンスの属性値が変更されれば、これに対応してアイコンの表示位置やコントロールパネル上での表示内容が更新される。これによって、システムの具体的な動作イメージを顧客に伝えることが可能となる。
  また、実行の途中で個々のインスタンスのコントロールパネルを通じて個々のインスタンスの属性値を任意に変更することも可能である。これにより、ユーザとのインタラクションによってシステムがどう動作するかを検討することも可能としている。

6．STATEMATEとの比較

Statechartを図的仕様記述言語とするプロトタイピング環境としては、STATEMATE[7]がある。STATEMATEは、ROAD/EEと同様、状態図をベースに仕様記述を実行し、その結果をアニメーションとして表示する機能を備えている。

STATEMATEとROAD/EEの最大の相違は、STATEMATEではインスタンスレベルの個々の状態に対してアイコンを設定するのに対して、ROAD/EEではアイコンをオブジェクトに対応して設定している点にある。このため、ROAD/EEではオブジェクト単位で仕様を部品化することが可能になる。例えば、複数の信号機をモデル化する作業が、信号機クラスをインスタンシエーションすることで簡略化される。

また、STATEMATEでは、基本的にタイムアウトを含む離散時間での動作しか扱うことが出来ない。これに対して、ROAD/EEでは連続的な値の変化を扱うことが出来る。これによって、システム外部の環境を数学モデルで表現したり、アイコンの連続的な運動を表現することが出来るようになっている。

7．まとめ

本報告では、現在開発を進めているオブジェクト指向仕様記述の実行・検証系ROAD/EEの狙いと機能概要を示した。

今後は、様々な分野での適用を試み、プロトタイピング環境としての機能の充実を図るとともに、特定ドメインに対応した仕様記述部品ライブラリの充実化を図り、仕様レベルでの部品化・再利用の効果を検討して行く必要があると考えている。また、仕様記述部品を充実することにより、プロトタイプの作成から始めて仕様記述モデルを作成するという利用方法も可能となると考えており、このための機能の拡充を図っていく方針である。

参考文献

[1] C.Potts, et.al, "Inquiry - Based Requirements Analysis," IEEE Software Vol.11 No.2, pp.21-32 (1994-3).

[2] G.Booch, "Object-Oriented Analysis and Design with Applications," The Benjamin/Cummings Publishing (1994).

[3] J.Rumbough, et.al, "Object-Oriented Modeling and Design," Prentice-Hall (1991).

[4] 田村他, "CASE環境構築用ツールキットの開発," 第44回情報処理学会全国大会 3J-3 (1992).

[5] D.Coleman, et. al, "Introducing Object5charts or How to Use Statecharts in Object-Oriented Design," IEEE Trans.on Software Engineering Vol.18, No.1 (1992-1).

[6] O.Maler, et.al, "From Timed to Hybrid Systems," Real-Time: Theory in Practice, pp.447-484 Springer-Verlag (1991-11).

[7] "First Annual i-Logix User Group Meeting," i-Logix (1992-10).