IIOSSにおけるUMLモデルの振舞い解析

佐野元之[†,††] 疋田輝雄[††]

オブジェクト指向ソフトウェア開発において用いられるモデリング図式UMLには，図式（ダイヤグラム）の文法チェック機能を備えた多くの編集ツールが開発されている．さらにダイヤグラムの振舞いや意味の正しさを調べるために，形式的仕様の上に行う論理的な自動検証の研究がさかんだが，それには論理的に完全な仕様記述が必要であり，モデル開発つまり設計中途における検証は行いにくい．本論文ではソフトウェア設計の中途においても使用しやすい，ダイヤグラムの振舞いチェックを行うために，以下の5つの機能からなる方法を提案する．(i) 図式シミュレーションとして，従来のモデルシミュレーションで行われるステートチャートに対してだけではなく4種類すべての振舞い図で実行可能である．(ii) 他種のダイヤグラムにまたがったコレスポンデンス（対応）機能を提供する．(iii) 同種の複数ダイヤグラムにまたがった，モデル要素間のメッセージ送受信機能を提供する．(iv) モデル要素とJavaプログラムの間のメッセージ送受信機能を提供する．(v) ユーザインタラクティブなデバッガ機能を提供する．これらによりユーザは，複数の異なるビューのダイヤグラムを作成しシミュレーションを行うことで，設計途中でモデルの意味的な間違いを見つけることができる．ここで提案する動的な振舞いチェック機能は，オブジェクト指向統合ケースツールIIOSS（Integrated Inter-exchangeable Object-modeling and Simulation System）のモデルデバッグ機能として実現している．このシステム上でこれらの機能を用いて行った大規模な開発経験の評価についても報告する．

A Global Behavior Analyzer for UML Models in the IIOSS System

Motoyuki Sano[†,††] and Teruo Hikita[††]

Most of UML editors have a capability to check the syntax of models described, but not its semantic behaviors. In order to incorporate semantic checking into UML, we introduce a model simulation mechanism to validate UML models and analyze system specifications. The simulation utilizes information on the models interactively supplied by the modeler. This mechanism has the following five features: (i) The simulator works not only on statecharts which are mostly used to simulate models, but on all four types of behavior diagrams. (ii) A correspondence functionality between model objects in different types of diagrams. (iii) A message passing functionality between model objects. (iv) A functionality that can pass messages between a model object and a Java program during simulation. (v) An interactive debugger that can provide software designers with a functionality to control the way how the simulation ought to proceed. The semantic checking functionalities described in this paper have been realized on the IIOSS (Integrated Inter-exchangeable Object-modeling and Simulation System) project and been enhanced in the IIOSS system as its model debugging facility. The results of evaluation for large or middle-size test software developments are also reported, which confirm that the method described here can be a helpful one in UML tools.

1. はじめに

オブジェクト指向のソフトウェア開発において，要求定義をまとめ，設計をするために，OMG（Object Management Group）が策定したUML（Unified Modeling Language）を利用することが一般的になってきている[4),19)]．システムアナリストやソフトウェア設計者は，グラフィカルなツールを使い，UMLの様々なダイヤグラムを編集することができる．多くのUML編集ツールは，ダイヤグラムの文法チェック機能を備えている．しかしダイヤグラムの振舞いや意味の正しさを調べる機能としては，いくつかのツールが，振舞いをチェックするためにステートチャート図のシミュレーションを行う程度である．UMLには様々な種類のダイヤグラムがある．たとえ大規模なシステムであっても，いろいろな視点から記述することができ

† 株式会社オープンテクノロジーズ
Open Technologies Corporation
†† 明治大学理工学研究科
School of Science and Technology, Meiji University

るが，ダイヤグラム間に矛盾がないことを確認する手段を持つツールは実用化されていない．

本論文では，UML におけるダイヤグラム間にまたがった振舞いを具体化するための，大規模なモデルに対応できるシミュレーションを提案する．具体的には以下の機能である．(i) シミュレーションのフレームワークは，4 種類すべての振舞い図（ステートチャート図，アクティビティ図，コラボレーション図，シーケンス図）で実行できる．(ii) モデル要素と，他種のダイヤグラム間の対応を記述できるコレスポンデンス機能を提供する．あるモデル要素と，違う種類のダイヤグラム上のモデル要素のグループとの対応を記述することにより，シミュレータは対応関係をたどってシミュレーションを行う．(iii) 同種の複数のダイヤグラムにまたがった，モデル要素間のメッセージ送受信機能を提供する．(iv) モデル要素と Java プログラムの間のメッセージ送受信機能を提供する．モデル要素が Java プログラムへのリンクを持つ場合，シミュレータは，その Java プログラムにメッセージを渡して実行し，終了時には戻り値を受け取り，シミュレーションを継続する．(v) インタラクティブなデバッガ機能を提供する．シミュレーションの途中で分岐に到達し，その行き先の判断ができない場合に，ユーザに遷移先を問い合わせる．これは，プログラムにおいてはないような，ダイヤグラムの曖昧さに起因する．外向きの遷移が複数ある場合に，シミュレーションを中断し，ユーザの判断を待つ．

これらの新しい機能を採用する理由は以下のとおりである．これまでの方法では，モデルの意味的なチェック，あるいはモデルのシミュレーションに，ステートチャートだけが使われていたが，これは不十分である．ステートチャートは，システム中のある 1 つのオブジェクトの詳細な振舞いを記述するのに適している．これに対して他の振舞い図は，オブジェクト間やシステム全体の振舞いが記述できる．意味的なチェックには両方の視点が必要である．特にコレスポンデンス機能とメッセージ送受信機能により，こういった他種ダイヤグラム間の整合性や，同種の他のダイヤグラムとの整合性のチェックをすることができる．さらに，ソフトウェア開発工程の初期段階では，モデルの記述は曖昧で不完全である．しかしたとえ不完全であっても，モデルの意味的なチェックやシミュレーションを行うことは，間違いの発見や修正だけでなく，モデルの微細で難しい点を発見するのに役立つ．

ここで提案する動的な意味的チェック機能は IIOSS システムのモデルデバッグ機能として実現している．本論文は，文献 14)，15) を発展させたものである．

2. モデルシミュレーションの技術動向

本章では，モデルシミュレーションと統合開発環境について，現在までの技術動向を述べる．

2.1 モデルシミュレーション

UP（The User Interface Prototyper）は初期のモデルシミュレータである．UP は，ER 図（Entity-Relationship Model）として書かれた状態遷移図と，概念プロセス図のシミュレーションを行う．アクション列は，モデル要素の属性として定義する．

近年，モデルシミュレーションをサポートしているソフトウェアが増えてきたが，その多くはリアルタイムや組み込み系のソフトウェア開発ツールである．これらのシステムでは，タイミングや，リアルタイム性の概念が重要な要素である．ユーザは開発ツールを次のようなステップで利用する．1) 状態遷移図もしくは状態遷移表を詳しく記述する．関連するダイヤグラム（クラス図，コラボレーション図など）も記述する．2) モデルをコンパイルし，実行形式ファイル（プログラム）を作成する．3) そのプログラムを実行し，モデルのシミュレーションを行う．

これらのツールは，独自の方法論に基づき独自のダイヤグラムを記述していたが[16),17)]，最近では UML の記法に移行している．

2.2 シミュレーションと UML

UME/Ruby は，ステートチャートの並列シミュレーションにより，内部および外部の振舞いをチェックできる[5)]．また，シミュレーションのログとして，シーケンス図を出力する機能を持つ．Mocha は，階層構造型の状態遷移グラフを利用したモデルチェッカである．ユーザインタフェースを持ち，内包するスクリプト言語を使って，細かい振舞いを記述することができる[1)]．また，アクティビティ図を拡張したアクティビティ・マシンのシミュレーション[3)] も提案されている．現状で，広く使われている UML（UML 1.3，UML 1.4）は，リアルタイム性の記述能力が低い．ツールベンダは，アクション，並列性，タイミングなどを記述できるように拡張している．例として，Realtime UML[2)]，eUML（Embedded UML）[22)]，Executable UML[18)]，Executable and Translatable UML[7)] などがあげられる．ここにあげた方法は，記述できる項目を拡張する，という静的な方法をとっているが，本論文では動的な方法を用いる．

OMG は，これら問題を解決するために，ツールベンダやユーザとともに作業を行っている．その標準

として，“Action Semantics for the UML” [9),10),20)] がある．この標準は，UML1.4 の拡張として定義され，UML 1.5 [11)] からは標準に含まれている．現在は UML 2.0 の一部として標準化の最終段階にある．現状の “Action Semantics for the UML” は，Semantics（意味）のみであり，言語としての統合とはほど遠い．たとえば，この標準に基づいたツールがあるが，どれもリアルタイム系もしくは組み込み系のソフトウェア開発ツールで，それぞれ別のアクション言語を持っている．これは文法が標準として定義されていないことに起因する[8),23)]．

2.3 統合開発環境 IIOSS

IBM 社（元 Rational Software 社）の Rose は，オブジェクト指向開発ではよく知られた統合開発ツールである[6)]．モデルエディタだけでなく，ソースコード生成機能を持ち，C++，ラウンドトリップをサポートするなど，多機能である．IIOSS は，UML を使ったオブジェクト指向の設計開発ツールである．第 1 著者らが IPA（Information Technology Promotion Agency, Japan，情報処理振興事業協会，現在の独立行政法人情報処理推進機構）の援助で開発しオープンソースで公開されている[12),21)]．IIOSS は，6 つのファシリティと呼ばれる独立した機能からなり，それぞれが関連しながら，要求定義から開発までのソフトウェア開発のライフサイクルをサポートしている．各ファシリティの関係を**図 1** に表す．

Model Editing Facility（MEF）と，Model Debugging Facility（MDF）は，要求定義と設計のフェーズにおけるモデルの作成と正しさの検証に用いられる．IIOSS はほかに，モデルと Java ソースコード間の変換，GUI，データベース，ネットワークのインタフェース定義，オブジェクトデータベースなどを持つ．本論文では，主に，MDF と，ともに利用する MEF について述べる．

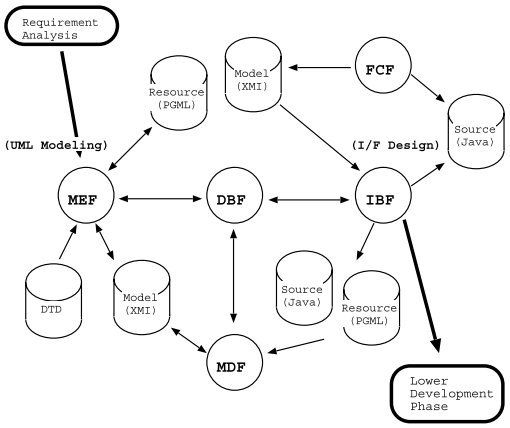

図 1 IIOSS ワークフロー
Fig. 1 IIOSS workflow.

3. オブジェクトとシミュレーションの概念モデル

3.1 オブジェクトの分類

まず，オブジェクトとシミュレーションにおける概念モデルとビューについて定義する．**図 2** に，以下で利用するオブジェクトの分類をあげる．この考え方は，主に，Java 言語のクラスを UML モデルの世界に取り込むために利用している．

1. モデルオブジェクト

モデル要素としてのオブジェクトである．モデルオブジェクトは，UML モデルオブジェクトと，仮想 Java オブジェクトからなる．

1-a. UML モデルオブジェクト

UML モデル要素として表現された直接実行不可能なモデルオブジェクトである．

1-b. 仮想 Java オブジェクト

既製 Java オブジェクト（実装済みの Java クラス）をモデル上で取り扱うために，モデル側に取り込んだ仮想的なモデルオブジェクトである．仮想 Java オブジェクトには，UML モデルオブジェクトと同等の情報に加えて，対応する実装済の Java クラスに関する情報が含まれており，シミュレーション時にはその Java クラスを実行することができる．

2. 実装オブジェクト

特定のプラットフォームで実行可能なオブジェクトを表す．たとえば実行可能なプログラムである．IIOSS においては Java クラス（プログラム）である．

2-a. スケルトンオブジェクト

クラス・ダイヤグラム中の UML モデルオブジェクトから生成される Java クラス・ファイルである．モデリング作業の結果をソフトウェア開発の下流工程に引き継いで行くことができる．

2-b. 既製 Java オブジェクト

すでに存在する Java クラスのこと．既製 Java オブジェクトから，仮想 Java オブジェクトとしてモデルに取り込むことで，モデル側で使用することができ

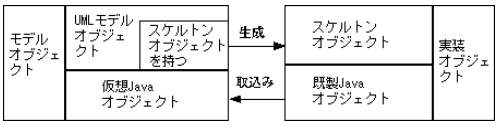

図 2 オブジェクトの種類
Fig. 2 Classification of objects.

る．この結果，シミュレーション中に，システム・ライブラリの使用や，すでに存在するソフトウェアの再利用が可能になる．

3.2 コレスポンデンス

4 種類の振舞い図にまたがったシミュレーションを行うために，新しくコレスポンデンス機能を導入する．これは，ある 1 つのモデル要素と，別の種類のダイヤグラム上にある，モデル要素のグループを関連付ける機能である．モデル要素のグループは，必ず 1 つの開始要素と，1 つ以上の終了要素を含む．コレスポンデンスの意図は，モデル要素の分解，あるいは，詳細化である．

コレスポンデンスの起源要素は，アクティビティ図のアクションステート，ステートチャート図のステート，コラボレーション図，シーケンス図におけるメッセージの両端に定義できる．展開されたダイヤグラムの要素は，起源要素と同じ要素型でなければならない．これらの起源要素と，展開要素とは，ダイヤグラム上で指定される．

シミュレータがコレスポンデンスの起源要素に到達すると，処理は，コレスポンデンスをたどり，展開後のダイヤグラムの導入点に移る．終了要素に到達すると，処理は起源要素に戻る．

以下で IIOSS を使ってコレスポンデンスを紹介する．ここでは，アクティビティ図とステートチャート図による例をあげる．これは現金自動預け払い機（ATM）のモデルで，ユーザからのパスワード入力の認証を行う部分である．**図 3** のモデル要素 “authorize” は，アクションステートを表すモデル要素の右上に四角いマークがついているが，このモデル要素にコレスポンデンスが定義されているという印である．コレスポンデンスは，他種類のダイヤグラム上のモデル要素グループへのリンクである．ここでは，**図 4** にあげるステートチャート図へのリンクになっていて，四角で囲まれた要素がコレスポンデンスとして定義されている．シミュレータの実行が図 3 中の “authorize” に達すると，コレスポンデンス定義を見つけて，図 4 中の “GetCard” に飛び，シミュレーションを継続する．コレスポンデンス定義の終了要素のいずれか（ここでは “Approved” か “Rejected” のいずれか）に着くと，シミュレータはアクティビティ図に処理を戻す．

この例において，コレスポンデンス定義は明らかに実用的な意味があり，アクティビティ図とステートチャート図の相互利用の価値を見出すことができる．

これを 4 種類の振舞い図に適用する場合，展開元と展開先のダイヤグラムの組合せとしては，大きく分けて以下の 4 種類にまとめられる．

i) アクティビティ図ないしステートチャート図から，それらいずれかへの展開
ii) アクティビティ図，ステートチャート図からシーケンス図，コラボレーション図への展開
iii) シーケンス図，コラボレーション図からアクティビティ図，ステートチャート図への展開
iv) シーケンス図ないしコラボレーション図から，それらいずれかへの展開

それぞれの場合について，利用可能性を**表 1** にあ

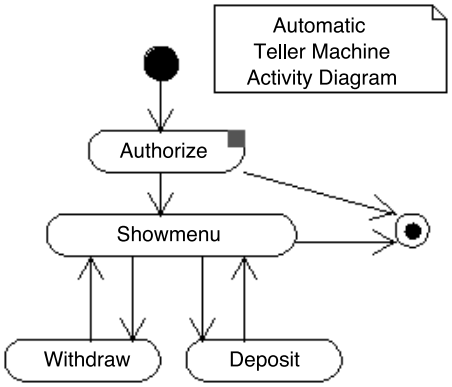

図 3 コレスポンデンス：アクティビティ図の起源要素
Fig. 3 Correspondence: origin in activity diagram.

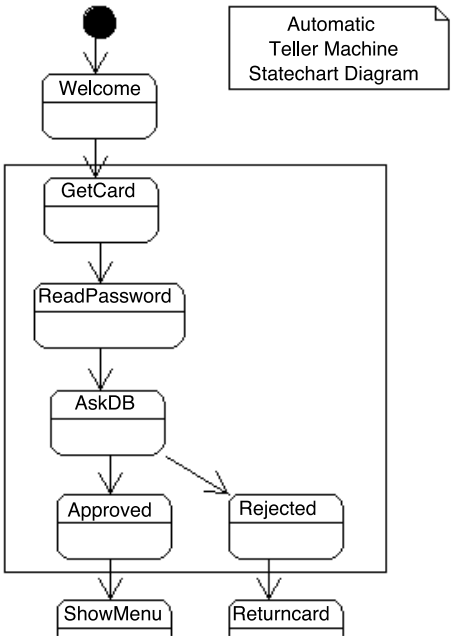

図 4 コレスポンデンス：ステートチャート図の展開要素
Fig. 4 Correspondence: expansion in statechart diagram.

表 1 コレスポンデンスの利用可能性
Table 1 Usabilities of correspondence.

from \to	Activity	Statechart	Sequence	Collaboration
Activity	○	○	○	○
Statechart	×	○	×	×
Sequence	×	×	○	○
Collaboration	×	×	○	○

げる．

図中の○印は実用的であり，×印は実用的でないことを表す．具体的には，以下のようになる．

i) 利用できる．ただし，ステートチャート図からアクティビティ図への方向の展開は，一般にステートチャート図の方が詳細であるため，実際に使われることは少ないと思われる．

ii) あるクラスの状態を記述するステートチャート図と他のダイヤグラムが連携することはない．アクティビティ図からシーケンス図やコラボレーション図への指定は，利用できる．

iii) シーケンス図，コラボレーション図からアクティビティ図，ステートチャート図へは，詳細度のレベルの違いから利用されることはないと考えられる．

iv) 展開（decomposition）的な使い方として利用できる．

IIOSS における実装においては，すべての組合せが定義はできる．しかし，実際の応用上の局面によっては，以上を制限として課すことも考えられる．

3.3 メッセージの送受信

2 つのダイヤグラムにまたがったモデル要素間のメッセージ送受信機能を導入する．これは，モデル要素の属性で，通常，遷移が起こったときのアクションを指定する effect と呼ばれる関連（association）を用いる．ユーザは，振舞い図の遷移オブジェクトに付帯する effect 関連にメッセージを指定できる．実際には，アクティビティ図やステートチャート図の遷移型の要素やコラボレーション図やシーケンス図のメッセージに，effect 関連を使ってメッセージを指定する．

シミュレータは，モデル要素の属性として，文字“^”で始まる文字列をメッセージ送信として解釈する．その形式は以下のとおりである．

^Class 名．アクション名（引数）

ここで，Class 名は，クラス名，あるいは，ダイヤグラム（アクティビティ図）名であり，アクション名は，メッセージの送信先のダイヤグラム名である．例としては，5 章の図 7 にあげる

^Stock．starting

がある．

コレスポンデンスは，あるオブジェクトの展開や分解であるが，メッセージ送受信は 2 つのモデル要素を関係付けるだけであり，より汎用な使い方ができる．典型的な使い方としては，ある振舞いを，2 つのダイヤグラムに記述し，同時に実行したい場合がある．

3.4 その他の機能

(1) 実プログラムの利用

Java プログラムを仮想モデル要素として扱うことができる．これは，Java プログラム（図 2 における既製 Java オブジェクト）をモデル要素（仮想 Java オブジェクト）として登録する，あるいは，モデル要素（UML モデルオブジェクト）から，スケルトンのソースプログラム（スケルトンオブジェクト）を生成することで利用可能となる．スケルトンソースプログラムは編集してもかまわない．シミュレータが Java プログラムにリンクした仮想モデル要素に到達すると，実際にその Java プログラムを起動する．その際に，メッセージを引数として渡す．Java プログラムの実行が終了すると，シミュレータは Java プログラムから戻り値をメッセージとして受け取り，その値を使ってシミュレーションを続ける．

(2) ユーザによる解決

一般に，モデルはプログラムよりも曖昧に記述されている．またユーザは，モデルを十分に詳細まで書き込まない段階でもシミュレーションを行いたいことが考えられる．シミュレータは複数の外向きの遷移を持つモデル要素に到達し，どの遷移に行くか判断できない場合に，自動的にユーザに問い合わせる．ユーザには，判断材料として，可能な遷移の一覧を提供する．このとき，スタブなどを定義する必要はない．

3.5 UML モデルのシミュレーション

UML モデルのシミュレーションは以下のように行う．

Step 1. ユーザは，シミュレーションを始めるダイヤグラムを選択する．メニューから，“model debug”を選びシミュレーションを開始する．新規に 2 つのウィンドウが表示される．1 つは，コマンドウィンドウで，ここからシミュレーションを操作する．もう 1 つは，履歴（ログ）ウィンドウで，シミュレーション中の各種メッセージが表示される．ユーザは，クラス図を選択することで，そのクラスに関連するステートチャート図のシミュレーションを行うこともできる．ユーザは，モデル要素にブレークポイントを設定することができる．シミュレータは，ブレークポイントが設定された要素に到達すると，シミュレーションを停止する．

Step 2. ユーザからのシミュレーション開始要求に

より，シミュレーションを始める．モデル要素に定義された変数の値が設定，もしくは参照された場合，変数一覧ウィンドウが表示される．現在，実行中のモデル要素は反転表示される．

Step 3. シミュレーションは次のように続行する．

3-(i) 1 つのダイヤグラム内の遷移

あるモデル要素から次のモデル要素への遷移は，ただ 1 つの遷移が存在する場合に行う．たとえばステートチャートにおいて，現在のモデル要素が state（状態）であれば，外向きの transition（遷移）が 1 つだけの場合にその transition に移動する．現在のモデル要素が transition であれば，矢印の先の state に移動する．モデル要素にブレークポイントが設定されていれば，シミュレータはいったん停止し，ユーザに処理を委ねる．ブレークポイントは設定されていないが，ガード条件が設定されている場合，シミュレータは条件を評価し，遷移を行う．ガード条件は，モデル要素の属性として定義できる条件式である．そのモデル要素が実行される際に評価される．結果が真であれば，遷移が起こる．以上のすべての評価が行われた後に，なお，シミュレータが遷移先を 1 つに特定できない場合は，可能な遷移先の一覧を表示し，ユーザに判断を仰ぐ．ユーザが一覧から遷移先を選択することで，シミュレーションを継続する．

3-(ii) コレスポンデンスによる他種ダイヤグラムへの移動

モデル要素がコレスポンデンス属性を持つ場合，その指定に従った遷移が行われる．実行中のダイヤグラムも指定されたダイヤグラムに移動する．コレスポンデンス終了に着くと，元のダイヤグラムの要素に戻る．

3-(iii) メッセージの送受信による他のダイヤグラムへの遷移

モデル要素が送信メッセージを持っている場合，シミュレータはそのメッセージに従った遷移を行う．実行モデル要素は，メッセージの送信先に移る．このとき，移動先のモデル要素は，送信元のモデル要素があるダイヤグラムと同じである必要はない．別のダイヤグラムの場合は，実行中のダイヤグラムも新しいダイヤグラムに移動する．

3-(iv) Java プログラム実行

モデル要素が Java プログラムにリンクした仮想モデル要素の場合，現実の Java プログラムを実行する．プログラムが実行を終えると，処理は実行したモデル要素に戻る．シミュレータは Java プログラムからの戻り値を利用できる．

Step 4. シミュレータが終了に到達すると処理を停止する．終了でなければ，Step 3 に戻りシミュレーションを継続する．

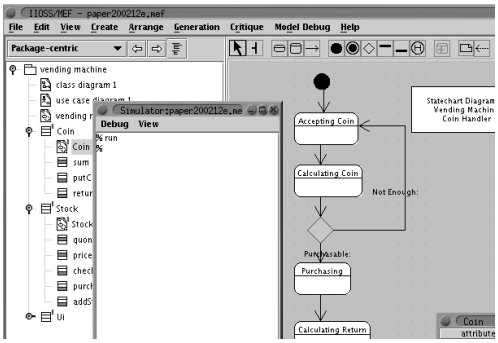

図 5 IIOSS MDF
Fig. 5 IIOSS MDF.

図 5 は，IIOSS MDF におけるモデルシミュレーション中の画面である．

4. シミュレータの実現について

本章では，シミュレータ実現における要点について述べる．

4.1 シミュレーションのリソースとエンジン

シミュレータは，UML エディタである MEF の上に，仮想的に定義したシミュレーションレイヤを利用して動作する．シミュレーションレイヤはシミュレータが利用するすべてのインスタンスが展開されている空間である．シミュレータは，モデル要素情報などの MEF が持つ情報は，MEF のものを共有する．このためユーザは，モデル編集の途中でも，容易にシミュレーションを行うことができる．シミュレータが起動すると，パッケージ org.iioss.mdf が作動する．このパッケージは UML エディタである uci.uml（初期版はカリフォルニア大学アーバイン校で開発された）パッケージから，ダイヤグラムの要素や属性情報を得る．それらを使って，シミュレータとして利用する情報をシミュレーションレイヤ上に展開する（アニメーションの反転を含むモデル要素の位置や表示に関する情報は，uci.uml 側のものをそのまま利用する）．また，ブレークポイントや，変数と変数値などの，シミュレータだけが利用する情報も展開する．シミュレータはまた，仮想モデル要素にリンクした Java プログラムの情報，シミュレータのコマンドの解釈，ユーザからの入力，履歴やスナップショット値の出力と，可能な型変換を行う．モデル要素間の移動の際には，モデル要素の関係をチェックする．外向きの遷移のガード条件などの制約条件を確認し，遷移が可能であれば

．現在のモデル要素の反転表示を元に戻す．

. 現在のモデル要素を次のモデル要素に移動し，新しい現在のモデル要素とする．

. 新しい現在のモデル要素の表示を反転する．

. 2 秒間待つ．

2 カ所以上の外向きの遷移が見つかり，シミュレータが遷移先を判断できない場合，ユーザに対してどちらに遷移するかを問い合わせる．終わりのモデル要素に到達した場合は，シミュレーションを停止する．

本システムは，実際には 2 種類のシミュレータを持っている．1 つは，ステートチャート図とアクティビティ図用で，もう 1 つは，シーケンス図とコラボレーション図用である．これら 4 種類の振舞い図は，実際には 2 種類の違った性質のグループに分けられるからである．

4.2 モデル要素間のメッセージ送受信

メッセージ送受信は，パッケージ org.iioss.mdf で行っている．シミュレータは，実行中の要素の属性にメッセージ送信を見つけると，メッセージ，変数，値，送信先のダイヤグラム名とモデル要素名を，メッセージ送受信エミュレータに送る．メッセージ送受信エミュレータは，受け取り先のモデル要素を見つけ，変数と値をそのモデル要素に渡す．エミュレータにエラーがなければ，シミュレータは処理を渡されたダイヤグラムのモデル要素に移す．

4.3 いくつかの制限事項

本システムでは，実行可能な Java プログラムだけを扱うことができる．また，引数は以下の型に限られる．

void，boolean，char，byte，string，short，int，long，float，double

これは，本システムが参照型を扱えないことによる制限である．また，プログラムからモデルを実行することはできない．ただし，実際に，変数の型が重要となるのはソフトウェア開発の下流工程であるプログラミングにおいてである．本システムのモデルシミュレーション機能が利用されるソフトウェア開発の上流工程（設計フェーズ）において，ユーザが，変数の型に制限があることで不自由に感じることは少ないと思われる．

シミュレーション中にモデル要素のアニメーションを行っているが，このとき，モデル要素間の遷移の際に，2 秒間の待ちが入っている．このため，実行形式ファイル型のシミュレータのようにタイミングを扱うことができず，リアルタイム系のソフトウェア開発向きではない．

5. 大域的実行の動作例

本システムの動作例として，1 つのアクティビティ図と，2 つのステートチャート図からなるモデルをあげる．このモデルは簡単な自動販売機を記述したものである．

5.1 ダイヤグラム例

3 つのダイヤグラムを提示する．1 つは自動販売機（Simple Vending Machine，SVM と略記する）のアクティビティ図 [A] である（**図 6**）．他の 2 つは，コイン取扱い（Coin Handler of SVM）のステートチャート [B]（**図 7**），と在庫管理（Stock Handler of SVM）[C]（**図 8**）である．

アクティビティ図 [A] は，自動販売機から商品を購入するときの人間の一般的な動作を記述している．ステートチャート図 [B] は，自動販売機に投入された貨幣と釣銭の扱いを，ステートチャート図 [C] は，在庫の管理を表している．この例では，各ダイヤグラム間の精細度は互いに異なっている．

5.2 SVM のシミュレーション

まず，アクティビティ図 [A] をシミュレーションを開始するダイヤグラムとする．“Model Debug” メニュー

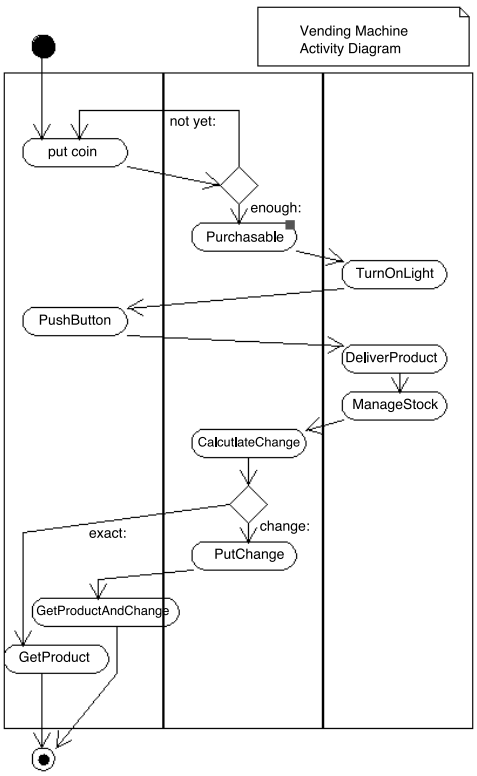

図 6 自動販売機 [A]
Fig. 6 Simple Vending Machine (SVM) [A].

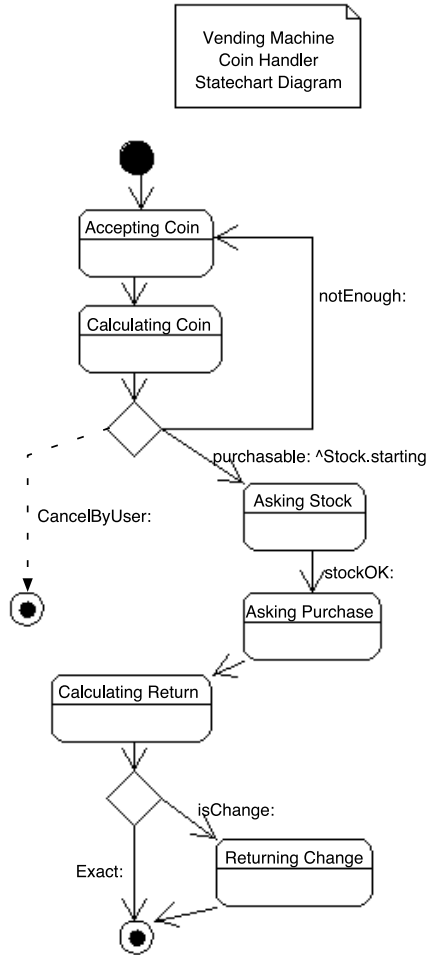

図 7 コイン取扱 [B]
Fig. 7 Coin handler of SVM [B].

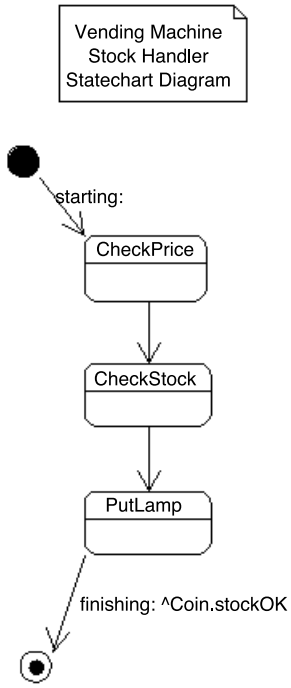

図 8 在庫管理 [C]
Fig. 8 Stock handler of SVM [C].

から “Start” を選択し，シミュレータを起動する．すると 2 つのウィンドウが現れる．1 つは，シミュレータウィンドウで，シミュレーションの操作を行う．もう 1 つは，Trace Log ウィンドウで，ログメッセージが表示される．ユーザが，シミュレータウィンドウのメニューから “Run” を選ぶか，“Run” に続けてリターンキーをタイプすることで，シミュレーションを開始する．

シミュレータは，アクティビティ図 [A] から begin 状態（大きな黒丸）を見つけ，最初の現在のモデル要素（以後，現在位置と呼ぶ）とする．この後，シミュレーションが停止するまで，現在位置はその要素をハイライト（反転表示）することで表される．次に，現在位置は，“begin” から外向きの遷移（矢印付きの線分）に移動し，それから矢印の先にあるアクション状態に移動する．この結果，現在位置は “put coin” となる．シミュレータがダイヤグラムの実行を始めると，そのダイヤグラム中で使われている変数とその値の一覧が，別のウィンドウに表示される．

次に，シミュレータは分岐（branch）状態（ダイヤモンド型のアイコン）に移動する．この分岐は，2 方向の外向きの遷移を持っている．“enough” と “not yet” である．ダイヤグラム中にはどちらの遷移に進むべきか記述がない．ここで記述とは，たとえば，ユーザの動作（いくら投入されたか，価格はいくらか）や，シミュレータが参照するガード条件，メッセージ送信といった属性である．シミュレータはここで動作を止め，ユーザに判断を仰ぐ．この例では，“enough” と “not yet” がポップアップメニューに表示される．ユーザが “enough” を選択すると，本システムはシミュレーションを再開し，“enough” 側の次のオブジェクトに現在位置を移す．

“Purchasable” には，コレスポンデンスを意味する赤い四角がアイコンの右上に示されている．シミュレータが “Purchasable” まで達すると，“Start Corresponding” で指定したモデル要素にジャンプし，シミュレーションを継続する．“End Corresponding” が定義されたモデル要素に着くと，処理が “Purchasable” に戻る．この例では，現在位置は，ダイヤグラム [B] 中で，“Start Corresponding” で定義された “begin” 状態に移る．ユーザは，ダイヤグラムの左下にあるダ

イヤグラムタブをダブルクリックすることで，ダイヤグラムをウィンドウとして取り出して表示し，複数のダイヤグラムを同時に見ることができる．ダイヤグラム [B] の “Purchasable” の属性を見ると，effect 関連が記述されている．シミュレータがダイヤグラム [B] の “Purchasable” に着くと，effect 関連内の記述を評価し，そこで指定された動作を行う．いまの場合はダイヤグラム [C] へのジャンプである．シミュレータはダイヤグラム [B] の代わりにダイヤグラム [C] を表示する．[C] の “end” 状態に着くと，現在位置はダイヤグラム [B] の “Purchasable” に戻り，シミュレーションを継続する．そして最後に，ダイヤグラム [A] の “end” に着き，シミュレーションを正常終了する．

この 3 つのダイヤグラムにまたがった記述において，ダイヤグラム [B] 中に点線で示した遷移は，購入のキャンセルにかかわる動作を表す．この変更自体は，ダイヤグラム [B] の中で閉じている．しかし，この変更を加えた後にシミュレーションを行うと，ダイヤグラム [A] において矛盾があることが判明する．ダイヤグラム [A] には，このキャンセル処理にかかわる記述がない．しかしシミュレーション矛盾により，ユーザはダイヤグラム [B] の変更にともなった修正を，ダイヤグラム [A] に施すことができる．これが，2 種類以上のダイヤグラムにまたがったシミュレーションを行う利点である．このようなエラーを見つけることは通常の UML 図によるモデルでは困難である．

別の例として，StockHandler の Java プログラムがあり，モデルのクラス図にスケルトンオブジェクトが定義されている場合を考えると，StockHandler クラスを選び，“Object config” メニューから “Type” を選択することで，モデル要素がスケルトンオブジェクトを持っているかどうかが分かる．この場合，“StockHandler” クラスは Java プログラムをスケルトンオブジェクトとして持っている．ステートチャート中のこのクラスに移動した場合，本システムは Java プログラムを起動する．

6. IIOSS 上での評価

本論文で述べてきた UML ダイヤグラムの大域的なシミュレーションの諸機能は，IIOSS 上で MDF（Model Debugging Facility）として実現されている．ソフトウェア開発において，UML ダイヤグラム記述およびモデルシミュレーションという IIOSS の機能を用いる場合と，IIOSS（すなわち UML）を用いず，従来の方式で行った場合の大規模（100 人日以上）な比較実験を行い文献 13) において報告した．実用に供する「ビジネスコンポーネント・クラスライブラリ」の設計開発に IIOSS を用いた結果，IIOSS の習得に時間がかかるものの，内部設計・製作・試験の見積り（人日）を実績は 14%下回った．一方，同時期に行われた，IIOSS を用いない従来型の類似開発では，見積りに対して実績の人日は 10%増であった．特に，テスト工程において実績の増が大きい（21%）．これはテスト時に内部設計時の誤りを検出して，いわゆる手戻りが発生しているためであった．IIOSS を利用した開発では，内部設計の段階において MDF を利用することで，手戻りを減らし，テスト工数を抑えることができた．

また，上記の 2 つのソフトウェア成果物の品質の比較を第三者評価（社内）によって行った．それによると，IIOSS を用いた場合，機能性（要求機能を満たす），使用性，開発の効率性，移植性の評価が高く，そして IIOSS を用いなかった成果物に優った．一方，信頼性と保守性は差がなかった．特に開発の効率の向上の点で IIOSS の使用は優位であった．この理由としては，モデル化によってメンバ間の正しい情報共有が進んだことと，MDF の利用によりテスト段階における手戻りが減少したことが考えられる．

コレスポンデンス機能はこの評価時には未完成で使用していないが，他の IIOSS の機能は上記のソフトウェア開発の内部設計においてすべて用いている．このような開発効率に対して，IIOSS における MDF の各機能の貢献の割合を定量的に分析することはむずかしい．今回の事例では，MDF を用いて内部設計時におけるチェックを十分に行うことにより，テスト工程における手戻りを削減できることが観察できた．

7. 結　　論

ここで紹介した動的なシミュレーションに関する新しい 5 つの機能は，ソフトウェア設計者にモデルに内在する意味的な不都合を早期に見つける手段を提供する．ソフトウェア設計者が，プログラミングを始める前に，意味的な間違いを早期に発見できるため，ここにあげた方法はソフトウェア開発に有効性があると考えられる．状態遷移図を用いた形式仕様に基づいた自動モデルチェック機能と比べても，我々が提案した 5 つの機能を中心とした意味的チェックの方法は，ユーザにとって使いやすいと思われる．理由は，要求分析や設計といった初期段階のモデル記述では，内容は完全ではないが，モデルのチェックは可能だからである．

ここで紹介した他種のダイヤグラムに遷移する方法は，実用上簡単で使いやすい．しかし，複雑なシミュ

レーションをコラボレーション図やシーケンス図に適用した場合には，整合をとるのが難しい．今後，展開方法や制限について適用例を増やすことで，より使いやすく，実用的なシステムにしていきたい．なお，これらの考え方を Action Semantics に適用することも考えている．

参 考 文 献

1) Alur, R., et al.: jMocha: A model-checking tool that exploits design structure, *Proc. 23rd Annual IEEE/ACM Int. Conf. Software Engineering*, pp.835–836, IEEE Computer Society Press (2001).
2) Douglass, B.: *Real-Time UML*, 2nd ed., Addison-Wesley (2000).
3) Eshuis, R. and Wieringa, R.: A real-time execution semantics for UML activity diagrams, *Fundamental Approaches to Software Engineering* (*FASE 2001*), LNCS 2029, pp.76–90, Springer (2001).
4) Fowler, M. and Scott, K.: *UML Distilled*, 3rd ed., Addison Wesley Longman (2003).
5) 池田健次郎，岸　知二：初心者のためのモデリング支援環境の構築，オブジェクト指向最前線 2002，pp.19–26，近代科学社 (2002).
6) Jacobson, I., Booch, G. and Rumbaugh, J.: *The Unified Software Development Process*, Addison Wesley Longman (1999).
7) Mellor, S.: *Executable and Translatable UML*, Project Technology (2002).
8) Mellor, S. and Balcer, M.J.: *Executable UML*, Addison Wesley (2002).
9) Mellor, S., Tockey, S., Arthaud, R. and Leblanc, P.: Software-platform-independent, precise action specifications for UML, UML98, pp.281–286, Mulhouse, France (1998).
10) Object Management Group: Action Semantics for the UML, OMG ad/01-03-01 (2001).
11) Object Management Group: OMG Unified Modeling Language Specification Version 1.5, OMG formal/03-09-09 et. al. (2003).
12) オープンテクノロジーズ：オブジェクト指向設計およびプロトタイピング統合開発環境の開発，情報処理振興事業協会 (1999).
13) オープンテクノロジーズ：オブジェクト指向設計およびプロトタイピング統合開発環境の開発実施検証報告書，情報処理振興事業協会 (2000).
14) Sano, M. and Hikita, T.: Dynamic Semantic checking for the UML models in the IIOSS system, *Int. Symp. Future Software Technology*, Xian, pp.220–225 (2004).
15) 佐野元之，山田正樹，疋田輝雄：IIOSS における UML モデルの動的検証，オブジェクト指向最前線 2002，pp.109–112，近代科学社 (2002).
16) Shlaer, S. and Mellor, S.: *Object-oriented Systems Analysis — Modeling the World in Data*, Yourdon Press (1988).
17) Starr, L.: *Executable UML*, Prentice Hall PTR (2002).
18) Starr, L.: *Executable UML — A Case Study*, Model Integration LLC (2001). ISBN0-9708044-0-7
19) Stevens, P. and Pooley, R.: *Using UML — Software Engineering with Objects and Components*, Pearson Education (1999, 2000).
20) Suny, G., et al.: Using UML Action Semantics for executable modeling and beyond, *CAiSE 2001*, LNCS 2068, pp.433–447, Springer (2001).
21) 鈴木重徳，倉骨　彰，佐野元之，垣花一成：IIOSS，アスキー (2001).
22) 渡辺博之，渡辺政彦，堀松和人，渡守武和記：組み込み UML，翔泳社 (2002).
23) Wilkie, I., et al.: *UML ASL Reference Guide — Manual Revision D*, Kennedy Carter(2003).

(平成 17 年 9 月 22 日受付)
(平成 17 年 12 月 26 日採録)

佐野　元之（正会員）

1959 年生．1978 年国際基督教大学より B. Liberal Arts．同年株式会社ソフトウェアリサーチアソシエイツ（現，株式会社 SRA）入社．1987 年より 1990 年にかけて米国 University of Hawaii at Manoa 客員研究員．1994 年より株式会社オープンテクノロジーズ．現在に至る．2001 年より明治大学博士後期課程在学．ソフトウェア工学に関する研究に従事．著書に『IIOSS』（共著），翻訳書に『JXTA』（共訳）．ACM，IEEE-CS，IEEE-SA，電子情報通信学会，ソフトウェア技術者協会各会員．

疋田　輝雄（正会員）

1947 年生．1978 年理学博士（東京大学）．1989 年より明治大学理工学部情報科学科教授．計算理論，ネットワークコンピューティング等に興味を持つ．著書に『コンパイラの理論と実現』（共著，共立出版）ほか．