IIOSSにおけるUMLモデルのインタラクティブな動的検証

佐野元之　疋田輝雄
明治大学理工学部 / 株式会社オープンテクノロジーズ　明治大学理工学部

既存のUMLツールは、文法上の間違いをチェックできるが、意味上の間違いは見つけることができない。本稿では、これを解決する方法として、モデル要素間のメッセージ通信機能を使った、UMLモデルのシミュレーションによる動的な検証を提案する。ユーザによる会話的な指示も用いる。また、実装オブジェクトとのメッセージ通信による、モデルと実装済みのプログラムが混在した環境でのシミュレーションについても提案する。

Interactive Validation for Dynamic UML Models in the IIOSS System

Motoyuki Sano　Teruo Hikita
Meiji University/Open Technologies　Meiji University

Abstract: Most of UML editors have a capability to check the syntax of the models described, but not its semantic behaviors. 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. We also offer a unique capability to send and receive messages between model objects, and even between model objects and real programs to simulate models in the more flexible and usable manner.

1. はじめに

ソフトウェア開発の上流工程において、ユーザのニーズを洗い出してまとめ、ソフトウェアの設計を行うために、最近、オブジェクト指向の統一モデリング言語であるUML（Unified Modeling Language）を使用することが有力となってきた[2],[3],[6]。UML用のグラフィカルなエディタで作成した各種のUML図式（ダイアグラム）は、クラスやオブジェクトの配置や関係、動作等の記述が直感的にわかりやすく、修正も簡単であり、さらに、再利用が可能である。またUMLのルールに従った図しか書けないということや、作成途中でも図の文法的なチェックができるなどの利点がある。

しかし、既存（市販）のUMLエディタは、図式の静的な文法チェック以上の、例えば、図式の意味に関わる部分の正しさはチェックできない。プログラミング言語では、文法チェックはコンパイラでできても、意味のチェックは、実際に動かしてみなければわからないことが多い。これはUMLモデルの図式についても同様である。ソフトウェア開発の上流工程での意味のチェックもれが、ソフトウェア開発の段階ではわからず、設計どおりにプログラムを作ったのに、そのプログラムは考えていたとおりに動かないという事態に陥ることがある。

モデリングの段階でモデルのシミュレーションを行うことで、プログラム・レベルに落とす前に、そのシステムの振る舞いをチェックすることができる。その結果、設計時のモデルの持つ問題点や誤りを見つけ出すことができ、下流工程以降でのトラブルを減らすことができる。そのためにUMLモデルのシミュレーションを行い、モデルの動的なチェック機能を導入する。モデルのシミュレーションは、UML図式の中で、振る舞い図と呼ばれる、動作を記述する４種の図式を対象として行う。

モデルはプログラムよりも抽象度が高いため、モデルの持っている情報だけではシミュレーションできないことがある。この場合における、ここで提案する方法は次のようなものである。

(i) シミュレーションにおける動作の分岐点において、分岐の方向が定まらないときは、ユーザに会話的に動作をたずねる。

(ii) モデル中のある部分が、すでにプログラム化され、実行可能な場合には、シミュレーション途中で必要に応じてそのプログラムを実行し、返り値を得てシミュレーションを継続する。

ソフトウェアの再利用や、ソフトウェアのコンポネント化によるソフトウェア設計および開発の効率化が言われて久しいが、モデルから実装オブジェクトを利用することで、既存のソフトウェア・リポジトリを有効に利用できる。

この方法は、統合オブジェクトモデリング／シミュレーションシステム IIOSS 中の４種の UML ダイヤグラムのシミュレーションにおいて実現されている。IIOSS は、Integrated Inter-exchangeable Object-modeling and Simulation System for Open Source Software Environment の略で、第一著者他によって開発されたオブジェクト指向設計／開発ツールである。本稿で述べる UML モデルのシミュレーションによる動的な検証方法は、この IIOSS システム中で実装されたものである。

2. 準備：UMLのダイアグラム

2.1 UML

UML（Unified Modeling Language）は、ソフトウェア・システムを分析、設計する際に、その成果物を明確化し、視覚化するための言語である [2]。1997 年に業界の標準化団体である OMG（Object Management Group）に対して、それまでのいくつかの代表的なオブジェクト指向設計手法を統合したものとして、G. Booch、I. Jacobson、J. Rumbaugh によって提案された [3]。

1997 年に OMG で現在の UML の基となる、UML1.0 が承認された。その後 UML の規格は拡張され、現在では UML1.4 となっている [6]。また OMG では現在、UML2.0 の策定が行われている。

2.2 各種のダイアグラム

UML には以下の９種類のダイアグラム（図式）が存在し、それぞれ用途によって使い分ける。

クラス図
オブジェクト図
ユースケース図
ステートチャート図（状態図）
シーケンス図
アクティビティ図
コラボレーション図
コンポーネント図
配置図

あるシステムを記述するために、必ずしも９種類すべての図を記述しなくてもよい。

動的なモデル・シミュレーションは、一般に振る舞い図と呼ばれ、システムやオブジェクトの状態や振る舞いを表わしている以下の４種類のダイヤグラムを対象とする。

a) ステートチャート図（状態図）

一般的な状態図は UML モデルに特有の図ではなく、UML 以前のさまざまな方法論でも使用されてきた。UML のステートチャート図は状態図の一種で、あるオブジェクト内部での状態遷移を記述したものであり、一つのクラスのオブジェクトのアルゴリズムを記述する。もし、ステートチャート図が十分に詳細に書かれていれば、機械的に実行可能である。もちろん、内容によって、あるいは、実行環境によって制約される場合はあるが、本質的には実行コードにコンパイルすることができる。市販の上流 CASE ツールの中で、モデルのシミュレーション機能を持っているものの多くは、状態図を使ったシミュレーションを行っている。

b) アクティビティ図

アクティビティ図は複数のアクティビティ（アクション状態）の時間的な遷移をあらわす。これは従来のデータフロー図、あるいはフローチャートなどに相当するものであると考えられる。記述内容は、ステートチャート図が外部要因を記述するのに対して、アクティビティ図は主に内部処理の振る舞いを記述する。

c) 相互作用図（シーケンス図、コラボレーション図）

シーケンス図および、コラボレーション図は、どちも、複数のオブジェクト間の関係をメッセージのやりとりとして記述する。これらは論理的に同じものに対して別の見方をしている。この２つの図は用途も近いが、シーケンス図にはコラボレーション図よりも詳細な時間関係を記述する場合が多い。相互作用図は、ステートチャート図に比べると、一般にシミュレーションに必要なレベルの情報まで記述しないため、シミュレーションをしにくい。このため、既存のツール等で実際に行っている例はないようである。

3. UMLモデルのシミュレーションによる検証の基本的な方法

3.1 モデルの内部表現

UML モデルの内部表現形式としては、XMI と PGML を使用する。XML Metadata Interchange (XMI)フォーマットは、UML モデルや MOF (Meta Objects Facility) などを交換するために IBM 他によって提案され、OMG で標準化された形式である [7], [8]。XMI フォーマットで、UML ダイヤグラム、ダイヤグラム内のオブジェ

クトやアトリビュート情報を持つ。

Precision Graphics Markup Language (PGML)は、W3 コンソーシアムにおいて標準化された言語で、2次元の絵を表すことができる［14］。PostScript や Portable Document Format (PDF)をベースとして、Web での使用を考慮した変更がなされている。PGML フォーマットで、ダイヤグラムのグラフィカル表現情報(位置情報)を持つ。

3.2 シミュレーション・レイヤ

UML ダイヤグラムの編集機能とは別に、モデルのシミュレーションを行うために、シミュレーション・レイヤを導入する。このレイヤでは、UML ダイヤグラムとして、UML エディタで扱うオブジェクトに関する情報を継承すると共に、シミュレーション実行中に必要となる、オブジェクト間の遷移にかかわる関係の情報、および、モデル・オブジェクトが実装オブジェクトにリンクしている場合の連係に必要な情報を持つ。この情報には、実装オブジェクト(プログラム)の、名前、引数、型、実プログラムの所在といった情報(メタ情報)を抜きだしたものが含まれる。シミュレーション・レイヤはまた、モデル・オブジェクトおよび実装オブジェクトのメッセージの送受信のシミュレーションも行う。シミュレーションはすべてこの空間において実行する。

なお、実装オブジェクトをシミュレーション中に使用する場合は、ユーザは事前にモデル・オブジェクトに実装オブジェクトを関連付けておかなければならない。この対応関係は、シミュレーション・レイヤで管理、使用する。

3.3 シミュレーション・レイヤの機能

シミュレーション・レイヤは以下の機能を持っている。

a) シミュレーション起動時、あるいは、該当モデル・オブジェクトが Activate された時に、シミュレーションに必要な情報をレイヤ中に取り込み、展開する。

b) 時間の流れやガード条件など、ダイヤグラム特有のルールに従って、シミュレーションを行う。

c) 遷移が一意に判定できない場合にはユーザに問い合わせる。

d) モデル・オブジェクトのメッセージ送信、および、受信をシミュレートする。

e) シミュレーション中に、関連付けられた実装オブジェクトとの間のメッセージ通信をシミュレートする。

f) 実装オブジェクトとのメッセージ通信を行うために、実装オブジェクト(プログラム)中の、名前、引数、型、実プログラムのファイルシステム上での所在、といったメタ情報を抜きだし、モデル・オブジェクトに関連付けておく。

3.4 シミュレーション実行の基本アルゴリズム

UML の振る舞い図のシミュレーションの基本アルゴリズムは、以下のような形である。

複数の、同種および異種のダイアグラムの XMI による表現をシミュレーションの対象データとする。ただし、シミュレーションの表示にかかわるオブジェクトの位置情報は、PGML による表現を用いる。以下のアルゴリズムの記述は、各種のダイアグラムに共通の、概念的な実行を表現している。各ダイアグラムに対するより詳しいアルゴリズムは4節で述べる。

シミュレーション・アルゴリズム

ステップ1

対象ダイアグラムのスタートオブジェクトを、シミュレーションの着目ノードとする。実行開始時に、ただ1つのオブジェクトが選択されている場合は、そのオブジェクトを着目ノードとする。

ステップ2

現在の着目ノードの内容を実行する。内容は実装プログラムの場合もある。終了オブジェクトの場合は実行を終了する。

ステップ3

着目ノードに付随する遷移情報が1つの場合は、その遷移情報にしたがって、着目ノードを次のオブジェクトへ遷移する。遷移情報が複数存在する場合は、ノードの実行結果、他のオブジェクトからのメッセージ、あるいは遷移先はユーザに問い合わせることで、次の着目ノードを決定し移動する。ステップ2へ戻る。

3.5 特徴

このシミュレーションアルゴリズムの特徴は次の3点である。

a) オブジェクト間でのメッセージ交換機能を実装し、複数のダイヤグラム間でのシミュレーションが可能である。

b) ステップ2において、モデルとプログラムの混在状態での動作が可能である。できあがった Java プログラムをモデル側の世界に登録することで、ダイヤグラムとのメッセージ交換機能が可能となる。この機能を使うことで、引数の受け渡し、戻り値の設定や参照を含めたモデルと Java プログラムとの相互呼び出しができる。UML モデリングの途中で、ある一部分はすでにプログラム化されている場合などに有効である。

c) ステップ3において、抽象的で、判断できない点はインタラクティブにユーザに問い合わせる。シミュレーションの途中で、状態遷移の分岐などが考えられる。こ

れをシステムが判断できない場合は、ユーザに可能な選択肢を提示し、問い合わせることでシミュレーションを続けることができる。

4. UMLモデルのシミュレーション

4.1 ステートチャート図のシミュレーション

4.1.1 ステートチャート図の特徴

ステートチャート図は、オブジェクトの状態を表す。オブジェクトはイベントを受取り、状態から状態へ遷移する。各状態は、名前とアクティビティのリストを持つ。ステートチャート図のシミュレーションは、イベントの受取りと、その時点での状態の表示である。

4.1.2 シミュレーションの開始および終了

ステートチャート図には、特殊な状態として、開始状態および終了状態がある。

1. 開始位置の判断と開始準備

1つのステートチャート図に複数の開始状態が存在することはない。このため、開始位置が指定されなければ、開始状態からシミュレーションを始める。この時、モデル・オブジェクトのシミュレーションに必要なインスタンスを生成する。開始位置となるオブジェクトが指定されている場合は、指定されたオブジェクトのインスタンスを生成し、シミュレーションを始める。

2. 終了位置の判断と終了準備

終了状態にたどり着くとシミュレーションを終了し、インスタンスが消滅する。

4.1.3 シミュレーションにおける遷移

遷移は、指定された名前とシグニチャを持つイベントが起こり、指定されたガイド条件が満たされた場合に引き起こされる。

1つの状態から複数の遷移が発生する場合には、指定されたガード条件を満たした遷移が選択される。ガード条件は、Guard アソシエーションに指定する。システムが遷移先を決定できない時はユーザに問い合わせ、シミュレーションを継続する。なお、遷移は複数のアクティビティを持つことができる。

分岐が発生しない遷移を単純遷移と呼び、イベントは起こった順に処理される。

ネストされた状態へ遷移が起こった場合、履歴インジケータ機能を持つことで、ネストされた状態(サブ状態)からネストしている状態(スーパー状態)に戻った時に、サブ状態中の状態を記憶しており、再びサブ状態になった時に、以前と同じ状態に戻ることができる。

4.1.4 その他の考慮点

他に考慮すべき点として、アクティビティとイベントがある。アクティビティとは、イベントを受信した時に状態遷移以外に行われるアクションやイベント送信のことである。アクティビティの動作は、1) アクティビティに対応するイベントを受信すると、2) ガード条件を判定を行い、満たせば、3)定義されたアクションを実行することである。

ここで、アクションとは、イベント送信、自然言語、疑似コード、実装コード、他のUMLダイヤグラムなどで記述することができる。なお、特殊なアクティビティとして、entry, exit, do がある。entry は、他の状態から遷移してきたときに暗黙に実行されるアクティビティ、exit は、他の状態に遷移したときに暗黙に実行されるアクティビティ、do はネストしたステートマシンを呼び出すものである。また、イベントは状態を遷移させるきっかけとなる外部からの、あるいは自律的な刺激のことであり、状態の変化、シグナル、メッセージ、時間など、いくつかの種類があるが、メッセージ送信として実現することとする。

4.2 アクティビティ図のシミュレーション

4.2.1 アクティビティ図の特徴

アクティビティ図は複数のアクション状態の時間的な遷移を表わすものであり、従来のデータフロー図、フローチャートなどに相当するものであると考えることができる。

アクティビティ図はステートチャート図の一種類と見ることもでき、内部構造上も似ているため、シミュレーションの実装にあたっては、ステートチャート図の機能を流用し特に区別は行わないものとする。

4.2.2 アクティビティ図とステートチャート図の違い

アクティビティ図とステートチャート図の主な違いは、以下のようになる。状態ではなくアクション状態を扱う。データ(オブジェクト)の流れを表わすこともできる。ステートチャート図が主にクラスや操作の実装を示すのに対し、アクティビティ図はそれ以外にも多くのアクティビティ(例えばワークフローやユースケース間の関係)を記述できる。ステートチャート図のように明確なイベントがなくても、アクションの完了によって次の状態への遷移を行う場合がある。

4.2.3 アクティビティ図のシミュレーションにおける制限

アクティビティ図中に、シグナルの送受信とそのバッファリングを記述した場合でも、特別な対応は行わない。

4.3 相互作用図(シーケンス図)のシミュレーション

4.3.1 相互作用図(シーケンス図)の特徴

UMLの４種類の振る舞い図のうち、シーケンス図とコラボレーション図は、相互作用図と呼ばれる。表現方法は違うが、いずれもオブジェクト(インスタンス)間の時間に沿った具体的な一連のメッセージ交換関係を記述するために用いられる。内部構造も同じであるため、ここでは代表としてシーケンス図のシミュレーションについて述べる。

シーケンス図では、ライフラインによって時間の流れが表されている。また、それぞれのオブジェクトは矢印付きの線分で表されるメッセージの交換を行う。シミュレーションでは、時間の流れの中でのメッセージ交換の様子を表す。アクション実行(制御の焦点)は、ライフライン上の空白の細長い長方形で表される。この長方形はそのオブジェクトが活性(アクティブ)であることを意味し、オブジェクトが自身のメソッドを実行しているか、他のオブジェクトに送ったメッセージの戻りを待っている(同期)を表す。

4.3.2 シミュレーションの開始および終了

ライフラインはオブジェクトの寿命を表し、上から下へ、相対的な時間の流れを示している。シミュレーションの開始と終了はライフラインの書き方で判断する。

1. 開始位置の判断と開始準備

ライフラインが図の最初から始まっている場合、シミュレーションはここから開始する。シミュレーションを開始する時点でモデル・オブジェクトのシミュレーションに必要なインスタンスを生成する。ライフラインが図の途中から始まっている場合には、対応するメッセージが送られた時点でインスタンスを生成する。

2. 終了位置の判断と終了処理

ライフラインが図の最後まで続いている場合は、実行が終了した時点でシミュレーションを終了し、それらのインスタンスが消滅する。ライフラインが図の途中で終っている場合には、そこにたどりついた時点でインスタンスが消滅する。

4.3.3 シミュレーションにおける遷移

時間は上から下に流れるため、線の上下が時間の前後関係を表している。基本的なシミュレーションの流れはこの時間関係に従う。オブジェクトはモデルに現れるクラスのインスタンスである。メッセージは、オブジェクト間の情報のやりとりで、ライフラインを方向付きの線で結ぶことでオブジェクト間のメッセージ交換を記述する。

既に定義されたクラスのインスタンスに対するメッセージの送信は、そのクラスに対応するメソッドのシミュレーションとなる。

4.4 メッセージ通信による異種のダイヤグラムが連動したシミュレーション

種類の異なるダイヤグラムにあるモデル・オブジェクトとの間のメッセージ通信機能は、イベントの一種として記述し、以下のように行う。ここでは、例としてステートチャート図を使用する。ステートチャート図では、遷移を表すTransitionオブジェクトを使い、そのアソシエーションであるにeffectアソシエーションを記述することで動作する。

指定方法は、

^${ダイヤグラム名}.${オブジェクト名}

である。ここでeffectアソシエーションは、そのオブジェクトが発火した時に行うことを記述するフィールドである。

1. メッセージの送信元のモデル・オブジェクトを選ぶ。ここでモデル・オブジェクトは、Transitionオブジェクトである。
2. そのモデル・オブジェクトのeffectアソシエーションに、メッセージ送信先のモデル・オブジェクト(ダイヤグラム名とモデル・オブジェクト名)を指定する。ここでオブジェクト名は、1. と同様に一般にはTransitionである。

この後、メッセージ送信元のダイヤグラムのシミュレーション中に、該当のモデル・オブジェクト(Transition)に到達すると、Effectが実行され、別のダイヤグラムのモデル・オブジェクト(Transition)にメッセージが送られる。

4.5 モデルとJavaプログラムとの間のメッセージ通信

4.5.1 実装プログラムとのメッセージ通信

シミュレーション中に、あるモデル・オブジェクトを実行した時に、そのモデル・オブジェクトが実装オブジェクトに関連付けされていた場合に実行される。

1. モデルの送信する抽象的なメッセージを実装オブジェクトのメッセージに変換する。
2. その実装オブジェクトをメッセージを付けて呼び出して実行する。
3. 実行終了を待つ。
4. 実装オブジェクトからの返り値を得て、モデルの抽象的なメッセージに変換する。
5. 送信元のモデル・オブジェクトに返答メッセージを渡す。

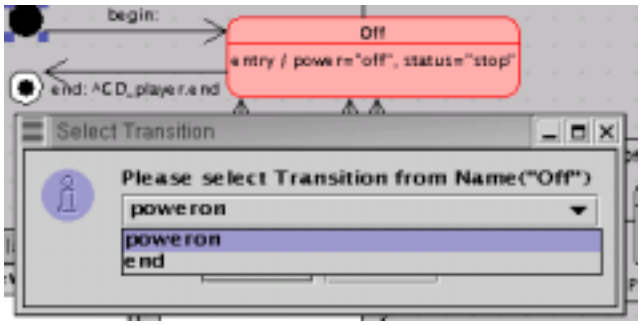
4.5.2 実装オブジェクトの取扱い上の注意点

1. 実装オブジェクト間のメッセージ交換については、実装オブジェクト間で直接行うため、考慮しない。
2. モデル・オブジェクトから実装オブジェクトへのメッセージの変数型の割り当ては、モデル・オブジェクトの内容から判断するため、記述が曖昧であるとうまく対応付けられない。

5. シミュレーション実行例

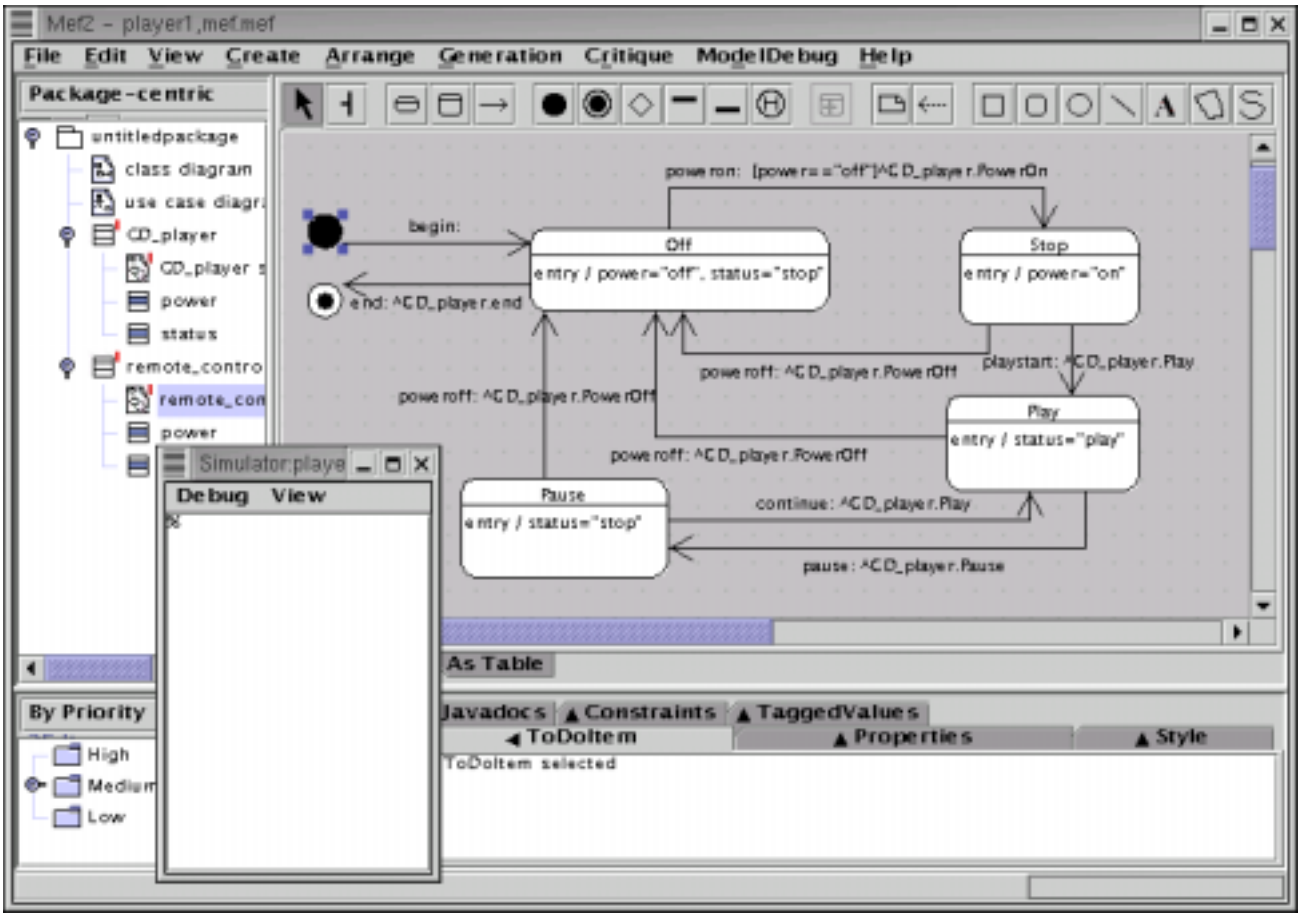
ここでは、CD プレイヤーのステートチャート図を使い、モデル間の通信をシミュレーションを行う。システムは、CD プレイヤー本体とリモートコントローラからなり、リモートコントローラで操作を行うとする。
リモートコントローラのステートチャート図(以下、リモートコントローラ図という)を選択、表示し、次にシミュレーションを開始すると、シミュレーションのコマンド用ウィンドウと、ログメッセージ・ウィンドウが表示される。

ステートチャート図は開始位置を示すオブジェクトは1つしか存在しないため、シミュレーションは開始状態から実行される。この時、図中で使用している各変数の値を表示するウィンドウが表示される。



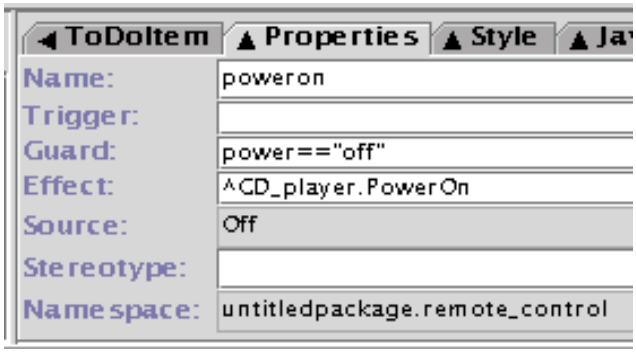
[図2] リモートコントローラ図：分岐状態に到達

シミュレーションの状態は、オブジェクトをハイライトすることで表わす。シミュレーションが進むと、クラス State の Off オブジェクトに到達する。このオブジェクトは遷移先として、poweron と end の2ヶ所あるが、システムは分岐のどちらに進むかを判断できないため、ユーザにポップアップメニューを表示して遷移先を問い合わせる。



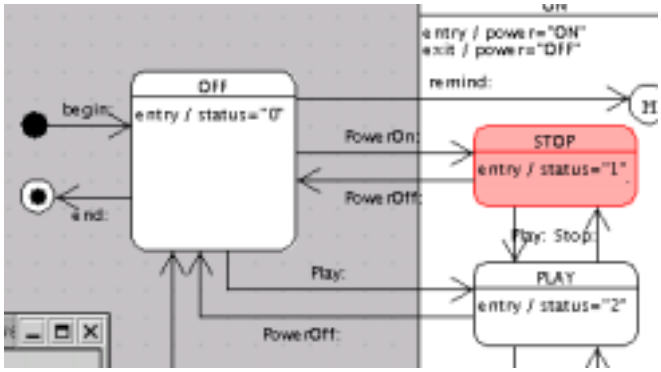
[図1] リモートコントローラ図：シミュレーション開始

ポップアップメニューから poweron を選択すると、停止した地点から実行を再開する。この時、Effect アソシエーションが指定されており、



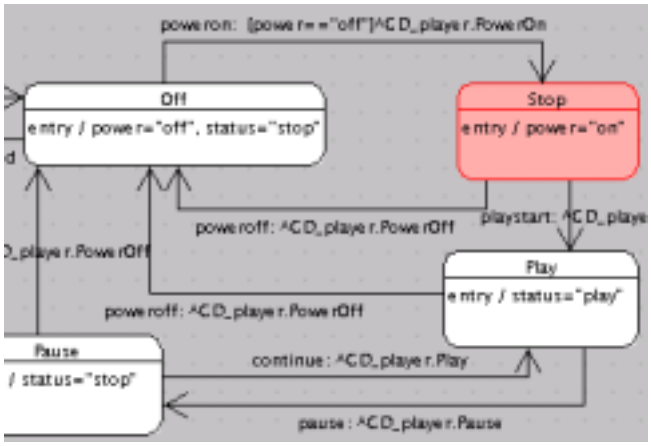
[図 3] Effect アソシエーション

シミュレーションが、リモートコントローラ図から CD プレイヤー本体のステートチャート図(以下、本体図と表す)に移動する。リモートコントローラからの poweron メッセージに従って、本体図のハイライトしているオブジェクトが OFF オブジェクトから STOP オブジェクトに遷移する。



[図 4] 本体図：遷移が発生

本体図の状態を変えた後、シミュレーションはリモートコントローラ図に戻り、次の State である Stop オブジェクトに遷移する。



[図 5] リモートコントローラ図：遷移

6 結論、考察

6.1 提案のモデル・シミュレーションの利点

a) モデリング初期の、あいまいなモデルでもシミュレーションを実行できる。
b) メッセージ通信機能により、大規模なモデルでもシミュレーションできる。
c) 実装オブジェクトとの連係機能により、既存のソフトウェア・ライブラリ(コンポネント)を利用でき、作業時間の短縮につながる。
d) デバッガに似たユーザ・インタフェースの採用で、使い始める際の教育に手間や時間がほとんどかからない。

6.2 他システムとの比較

モデルのシミュレーション（実行）自体は新しい考えではない。例えば、ステートチャート図（状態図）は、UML に限らず、古くからある記述形式である。その記述内容がプログラムに近いため、厳密に記述すればシミュレーションを行うことができることは知られていた。

UML 以前のモデル・シミュレーションの例として、UP（User Interface Prototyper, ICS Dept, Univ. of Hawaii at Manoa, 1987-1990）[13] がある。UP は、ステートチャート図と「概念的プロセス図」を対象とし、オブジェクト（UP システム中ではエンティティ）の属性として動作内容を記述することで、ユーザ・インタフェースのプロトタイピング・ツールとして使用したものである。当時の他のシステムも、モデルのシミュレーションという場合は、ステートチャート図のシミュレーションを行なうものが多数であった。

その後、特に組み込み系のソフトウェア開発に、モデルの実行が使われるようになった。組み込み系は、最終的に製品としての ROM チップを作成するが、いったん作成した ROM チップを後で交換するのは難しいため、システム完成前にプログラム（モデル）を動作させることは重要であった。最近でももっともユーザの多い分野である。これらのシステムでは、方法論の一つでありシミュレーション機能が備わった Shlaer-Mellor 法に基づいてモデルを記述しているものが多い [10]。実際に動くソフトウェア・ツールとしては、Project Technologies 社製の BridgePoint がある。

最近になって、UML モデルを実行しようという動きが出てきており、いくつかの提案がなされている [4] 。UML の標準化を行っている OMG（Object Management Group）でも討議が行われており、その中の有力な候補として Action Semantics がある。これは、UML の記述を拡張して、モデルを実行できるようにしようというものである [5]。

参考文献

1) T. DeMarco: Structured Analysis and System Specification, Prentice-Hall Inc. 1979. （トム・デマルコ 著 , 高梨智弘 , 黒田純一郎 監訳: 構造化分析とシステム仕様 , 日経 BP 出版センター , 1986.）

2) M. Fowler, K. Scott: UML Distilled, 2nd Edition, Addison Wesley Longman, Inc., 2000. （M. Fowler, K. Scott 著 , 羽生田栄一 監訳: UML モデリングのエッセンス 第 2 版 , 翔泳社 , 2000.）

3) I. Jacobson, G. Booch, J. Rumbaugh: The Unified Software Development Process, Addison Wesley Longman, Inc., 1999. （I. Jacobson, G. Booch, J. Rumbaugh 著 , 日本ラショナルソフトウェア 訳 , 藤井拓 監修: UML による統一ソフトウェア開発プロセス , 翔泳社 , 2000.）

4) S. J. Mellor, S. Tockey, R. Arthaud, P. Leblanc: Software-platform-independent, Precise Action Specifications for UML, UML99 at Colorado, USA, 1999.

5) Object Management Group: Action Semantics for the UML, OMG ad/2001-03-01, 2001, OMG TC March 2001.

6) Object Management Group: OMG Unified Modeling Language Specification Version 1.3, 1999, OMG00-03-01.

7) Object Management Group: XMI Specification, OMG ad/98-10-05, 1998.

8) Object Management Group: XMI Specification, Appendices, OMG ad/98-10-06, 1998.

9) （株）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　基本設計書 , 情報処理振興事業協会向けビジネスオブジェクト関連システム開発事業 , 1999.

10) L. Starr: How to build Shlaer-Mellor Object Models, Prentice Hall, Inc., 1996. （L. Starr 著 , Shlaer-Mellor 研究会 訳: オブジェクト・モデリング , プレンティスホール出版 , 1998.）

11) P. Stevens and R. Pooley: Using UML – Software Engineering with Objects and Components, Pearson Education Limited, 1999,2000. （P. Stevens, R. Pooley 著 , 児玉公信 監訳: オブジェクト指向とコンポーネントによるソフトウェア工学 , ピアソン・エデュケーション , 2000.

12) 鈴木 , 倉骨 , 佐野 , 垣花: IIOSS – UML に基づく設計／開発環境のすべて , アスキー , 2001.

13) Software Engineering Research Laboratory, ICS Department, University of Hawaii at Manoa: UP (User Interface Prototyper) http://www.ics.hawaii.edu/

14) World-Wide Web Consortium (W3C): NOTE-PGML-19980410 (Precision Graphics Markup Language), W3C Technical Reports, W3C, 1998.

付録：IIOSS の概要

IIOSS（Integrated Inter-exchangeable Object modeling and Simulation System for Open source software environment）は、オープンソースのライセンスで公開されている、オブジェクト指向設計支援ツール、プロトタイピング開発環境およびコンポネント構築支援ツールである [9] , [12]。

IIOSS は、ファシリティと呼ばれる 6 つの独立したツールからなるが、本論文では主に、UML エディタであるモデル編集ファシリティ（Model Editing Facility：MEF）と、UML モデルのシミュレーションを行い、モデルの検証やデバッグをインタラクティブに行う、モデル検証ファシリティ（Model Debugging Facility：MDF）について述べた。

他には次の 4 つの機能がある。

・インタフェース構築ファシリティ（Interface Building Facility：IBF）

・フォーマット変換ファシリティ（Format Conversion Facility：FCF）

・データベースファシリティ（DataBase Facility：DBF）

・総合開発環境（Integrated Development Environment：IDE）

これらのツールは、要求分析、設計、設計の検証、プロトタイピングといったソフトウェア開発の各段階を支援するものであり、互いに連携することによってソフトウェア開発のプロセス全体をサポートしている。詳しくは、http://www.iioss.org を参照されたい。なお、IIOSS プロジェクトは、情報処理振興事業協会(IPA)が推進する「ビジネスオブジェクト関連システム開発事業」の一環として行われた。