効果的なソフトウェア開発のためのシステムシミュレータ

4J－4

石井 基樹† 山田 将弘† 金指 文明† 富樫 敦‡

† 静岡大学大学院理工学研究科 ‡ 静岡大学情報学部情報科学科

1 はじめに

近年のソフトウェアの需要の増大，人間の介入によるバグ混入への対処のため，手順の多くを自動化したソフトウェア開発支援環境を構築する試みが盛んに行なわれている．その中で我々は，命題論理を元にしたユーザ要求を受け取り，一連の自動化された手順のうちにもユーザ自身が参画できるようなシステムを通じ，最終的に実行可能コードを生成するという，ユーザ側の負担を極力減らしたソフトウェア開発環境を提案している[1][2]．このシステムの流れとしては，まず，ユーザが設計したいソフトウェアを要求仕様として入力し，これを合成システムによって形式仕様に変換，その後ジェネレータを通してリアクティブシステムとして動作する実行可能コードを生成する．

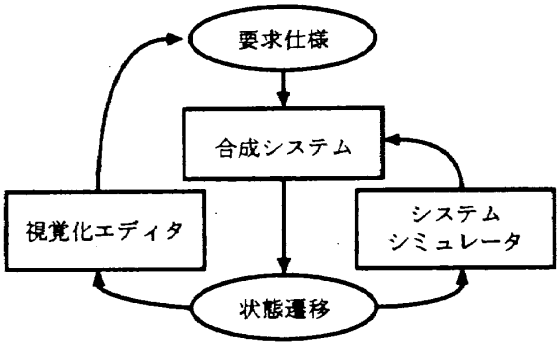

図 1: ソフトウェア開発環境

ここで，ユーザが最初に与えた要求仕様は必ずしもユーザ自身の思い描いているソフトウェアを完全に記述していないことに着目した．このため，要求仕様を形式仕様に変換し，得られた状態遷移を視覚化してユーザ自身に確認させるシステムシミュレータ，そこで発見された不具合を状態遷移システム上で修正する視覚化エディタ，の二つのツールを用いたフィードバック機構を用意した．この事は初期の要求修正だけでなく，後に要求そのものが変更になった場合もその修正作業を容易にする役割を持つ．

本論文では，このうちシステムシミュレータの開発を行った．

2 シミュレータの動作

要求仕様から合成システムによって生成される形式仕様は，状態遷移システムである．これがユーザが本来望むものと合致するかどうかを，状態遷移システムを視覚的に表現しその動作をシミュレートすることで確認するために用意されたツールが，状態遷移システムシミュレータである．

TV をモデルとした簡易な設計を考える．ここで，on とは電源が入っている事を示すリテラル，$mute$ とは音声が消えている事を示すリテラルとする．

ユーザ要求は，例えば次のようになる．

$$on, mute \xrightarrow{mute} \neg mute$$

$$on, \neg mute \xrightarrow{mute} mute$$

$$on \xrightarrow{power} \neg on$$

$$\neg on \xrightarrow{power} on$$

これを合成システムに通す事で，形式仕様が得られる．シミュレータはこの形式仕様を入力として受け，図 2 のように状態遷移システムを視覚的に表現する．ユーザは，シミュレータ上に展開された図形に対する操作によって，形式仕様の検証・確認を行う．

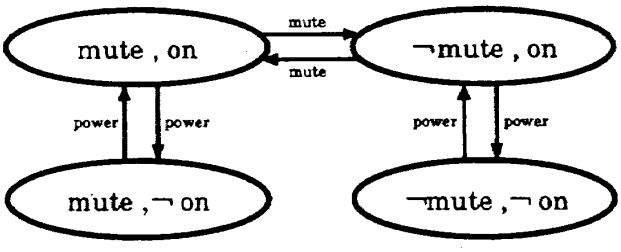

図 2: 視覚的表現

A System Simulator for effective software development

Motoki Ishii, Masahiro Yamada, Fumiaki Kanezashi, Atsushi Togashi

†Graduate school of Science and Engineering, Shizuoka University

‡Department of Computer Science, Shizuoka University

3 リテラルへの階層的意味付け

複雑なシステムを設計するうえで，状態数が莫大なものになることが予測される．これらを全て一度に扱うとなると，表示の面でも煩雑になり，シミュレートの面でも把握が困難となる．また，形式仕様の規則により，一つの状態には全てのリテラルが含まれる事になるが，その中にはある組合せにおいて意味を為さないリテラルも存在し，冗長な情報となる．システムシミュレータはこの事に対処するため，リテラルへの階層的意味付けを行い，これを元に扱うリテラル・状態を制限している．具体的な例として，先の TV のモデルを考える．

$mute$ は電源が入っている時以外は意味を持たないことに着目する．$mute$ は on という素命題と共になった状態にある場合のみ有効であるため，$\neg on$ という素命題と共にある場合においては $mute$ というリテラルは不要なものとなり，$\{mute, \neg on\}$ $\{\neg mute, \neg on\}$ という状態は一つの状態 $\{\neg on\}$ と見なすことができる．

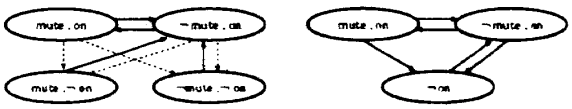

図 3: 階層的意味付け

このように，リテラルに対してそれぞれの依存関係を階層的に指定することを階層的意味付けと呼んでおり，これにより冗長なリテラルの削除，状態数への制限を行うことができる．

4 共通グラフィックエンジン

システムシミュレータと視覚化エディタは個別のものとして開発されている．しかし，シミュレート中に不都合が発見された場合，その場で修正することを可能にするために，これらは統合化されていた方が都合が良い．

ただ，システムシミュレータと視覚化エディタは，状態の扱いが異なるという問題があり，統合は非常に困難である（前者は状態に含まれるリテラルが変化するのに対し，後者は全てのリテラルを含む形式仕様そのものが状態として固定される）．

そこで，個別に開発されたとしても使用する上で違いを感じないよう，状態表示・操作部を共通化する事にした．具体的には，共通グラフィックエンジンを開発し，両者がこれを状態表示部に組み込む．このエンジンは，状態を表す図形，遷移を表す有向曲線，ポインタが図形上で静止した際に指定した文字列を表示する等の表示全般を司り，また図形上でクリックされた際にプログラムにメッセージを送るというようなユーザの操作の検出も行う．

ここで，エンジンが扱うのは図形として，曲線として，といった抽象化概念のみであり，それぞれの図形がどの状態に対応するかといった内容は，呼び出し元のシステムシミュレータ，視覚化エディタの扱う範囲である．

このように，状態の意味合いの違いをそれぞれのプログラムが隠蔽しながら，共通のエンジンを用いることで操作感の統一を計っている．

5 おわりに

システムシミュレータと視覚化エディタを統合することでユーザの生産効率が向上することは明らかである．そのため，統合化への道を模索していきたい．

システムシミュレータは全体的に，ユーザ主導型ツールであり，主要な状態遷移の確認を行う箇所等の発見もユーザまかせである．このような確認の要所を発見する方法論もシミュレータに実装し，ユーザの負担を軽減することが望ましいが，そのためには入力として形式仕様のみならず，付随する意味情報も与えられるべきである．実用的な使用を考える上では，ユーザの要求を受ける段階でその意味付けを行うアプローチが必要と思われる．

謝辞

本論文は，一部文部省科学研究費 (基盤研究 (C) 08680343，重点領域研究 09245214)，電気通信普及財団，東海産業技術新興財団，栢森情報科学新興財団の援助による．

参考文献

[1] Togashi, A., Kanezashi, F. and Lu, X.: A Methodology for the Description of System Requirements and the Derivation of Formal Specification, In FORTE/PSTV'97, pp.383-398(1997).

[2] 金指文明, 陸暁松, 富樫敦: システム要求と形式仕様の導出,1997(投稿中)

[3] 山田将弘, 石井基樹, 金指文明, 富樫敦: システム要求仕様化のための状態遷移システム表示インターフェースの開発,1998