3．制御用プログラムの試験

Testing Control Programs: A Survey by Naoshi UCHIHIRA and Hideji KAWATA（Research & Development Center, Toshiba Corp.）.

内平　直志[1]　川田　秀司[1]

1（株）東芝研究開発センターS&S研究所

1．はじめに

制御用プログラムは制御対象と組み合わせて完結した試験が可能となる．しかし，制御対象と組み合わせた総合試験は開発の最終段階まで実施できないことが多く，試験に要するコストも高い．そこで，総合試験より前のプログラム開発段階でできるだけバグを除去しておく必要がある．プログラム開発環境における試験では，何らかの手段で制御対象の挙動を模擬する必要がある．設計者が制御対象の挙動を模擬して手動で信号を入力し，それに対する出力を確認する原始的な方法もあるが，複雑な試験に対しては限界がある．そこで，制御対象の挙動を自動的に生成するシミュレータが有用である．さらに，制御対象の可能な挙動の組合せが膨大な数になる場合でも，制御対象のシミュレータを用いることで，それらの挙動を網羅的に生成し試験することが可能となる．本稿では，制御用プログラムの試験方法を概説し，とくに（1）制御対象シミュレータの作成支援技術，（2）制御対象シミュレータを用いた自動試験技術を説明する．具体的な事例としてシーケンス制御システムと組込みシステムのプログラム試験環境を紹介する．

2．制御用プログラム試験の特徴

制御用プログラムの試験の一般的な特徴を示す．

- 制御用プログラムは，制御対象からのセンサ情報を入力して制御対象へ制御命令を出力する**反応型システム**（reactive system）であり，制御対象と組み合わせてはじめて全体として試験できる．
- 通常，制御用プログラムはセンサ情報に基づいて起動される複数の並行処理単位（タスク）から構成される．同じ試験データに対してもタスク起動のタイミングによって挙動や結果が異なる．すべてのタイミングを漏れなく試験するのは難しい．また，試験の再現も難しい．
- 正常処理のプログラムに対して異常処理のプログラムの比率が大きい．たとえば，制御機器に起動命令を送った後，通常は正常に作動しているか否かをセンサで確認し，もし正常でない場合は異常処理を行う．このような異常処理の複雑な組合せを漏れなく試験するのは難しい．
- プログラム開発環境と実行環境が異なる場合が多い．開発環境をホスト環境，実行環境をターゲット環境と呼ぶこともある．また，開発環境と実行環境が異なる場合をクロス環境と呼ぶ．
- ソフトウェアとハードウェアを並行して開発することが多く，総合試験ではソフトウェアとハードウェアを同時に試験／デバッグする．
- 制御システムの多くはリアルタイムシステムであり，実時間制約を満たしているか否かの試験が必要である．

制御システムの具体例には，プラント，航空機，自動車，ロボット，医療機器，計算機周辺機器，電話機，家電製品などがある．以下では，プラントのプロセス制御システムと家電製品などの量産品のマイコン組込みシステムのプログラム試験について具体的に説明する．

2.1　プロセス制御用プログラムの試験

電力，鉄鋼，化学プラントなどのプラントにおいて個々の制御機器（モータ，バルブなど）を統括してプラント全体を制御することをプロセス制御と呼ぶ．プロセス制御は，プロセス制御用コンピュータ（プロコン）やプログラマブルロジックコントローラ（PLC）によって行われる．通常，制御対象であるプラントは客先に存在し，ソフトウェアを開発するメーカ側にはない．このようなプロセス制御用ソフトウェアの試験は，一般に次のステップで行われる．

1．プログラム単体試験

プログラム開発担当者がモジュールやタスクの単位で試験を行う．具体的には，机上デバッグおよびスタ

ブを使った試験実行を行う．

2．プログラム結合試験

ソフトウェア設計部門で開発したプログラム全体を結合して試験を行う．プログラム開発環境で行う場合もあれば，ターゲット環境（プロコン，PLC）で行う場合もある．

3．工場出荷前試験

メーカ側の工場出荷前に品質保証部門がソフトウェアとハードウェアを総合的に試験する．ここで，ハードウェアとはプロコン，PLC，操作パネル，一部の計測／制御機器などのメーカ側で開発した関連機器である．

4．現地調整

客先のプラントにソフトウェアおよびハードウェアを設置し，実際の運用の前にプラント全体としての試験と調整を行う．

プロセス制御用ソフトウェアでは，総合試験が工場出荷前試験と現地調整の2段階から構成される．一般に，工場出荷前試験や現地調整などの後工程になればなるほど，試験／デバッグにはより大きな労力を要する．「変数を初期化する前にあるタスクがその変数をアクセスした」などのプログラムの論理的な誤りは，総合試験以前に取り除くべきであり，その種の誤りを総合試験で取り除くのは効率が悪い．総合試験では，制御対象との微妙なタイミングや実時間制約のチェックに集中すべきである．

2.2 組込み制御用プログラムの試験

量産品のマイコン組込みシステム開発固有の特徴としては，(1) 開発期間が短い，(2) ターゲット環境は計算資源の制約が厳しくクロス環境が不可避，などがある．クロス環境特有のソフトウェア試験／デバッグツールには以下のものがある．詳しい解説は文献1)，2) にある．

- ターゲットモニタ：

ターゲット環境においてアプリケーションと同時にモニタ用プログラムを動かし，ホスト環境からシリアルポートやネットワーク経由でターゲットの実行の監視や制御を行う．

- インサーキットエミュレータ：

インサーキットエミュレータ（ICE: In-Circuit Emulator）は，ターゲット環境においてプロセッサの位置に端子（プローブ）を差し込み，プロセッサをハードウェア的にエミュレートする装置である．ICEはホスト環境から監視および制御でき，実行状況をソースコードと対応させて追跡できる．プロセッサの代わりにROMにプローブを差し込むROMエミュレータでも同様の試験が可能である．

- コードレベルシミュレータ：

シミュレータはプログラム開発環境で作動する試験／デバッグツールである．すなわち，ターゲット環境がない状況でもプログラムの試験が可能である．シミュレータには，プログラムのシミュレータと制御対象のシミュレータ（周辺回路シミュレータ，機械制御系シミュレータ，外部環境シミュレータ）がある．プログラムシミュレータには，ターゲット用にコンパイルされたオブジェクトコードを実行するタイプ（すなわち，プロセッサのシミュレータ）とソースコードをプロセッサに依存しない形式で実行するタイプがある[3]．制御対象のシミュレータに関しては後で詳しく述べる．

- 設計仕様レベルシミュレータ：

本ツールは，設計仕様エディタ，仕様試験部，コード生成部から構成される．設計仕様は主に状態遷移表現（Statechartなどの状態遷移図や状態遷移表）で記述される．仕様試験部では，設計仕様の（網羅的な）シミュレーションを行う．また，設計仕様レベルで実行時間の見積りや解析を行うツール☆もある．仕様試験部で正しさの確認ができた設計仕様からターゲット環境で実行可能なプログラムのコードを生成する．

これらのツールを使った試験手順の一例を示す．

1．設計仕様試験

システムの設計仕様を状態遷移表現で記述し，仕様レベルのシミュレーションを行う．正しく動くことが確認できたらターゲットのソースコードの骨格を生成する．

2．プログラム単体試験

生成されたソースコードを基にドライバや制御アルゴリズムなどの詳細な部分のプログラミングを行った後，モジュールやタスク単位で試験を行う．

3．プログラム結合試験

プログラム開発環境のコードレベルシミュレータを用いてプログラムの結合試験を行う．

4．ターゲット（実機）試験

ICEなどを用いてターゲット環境上の総合的な試験を行う．

現実には，設計仕様レベルやコードレベルのシミュレータが利用できない場合が多く，ICEなどを用いたターゲット試験が主体となる．

3．制御対象シミュレータ作成支援

制御用プログラムの試験には制御対象が不可欠であるが，以下のケースでは制御対象が存在しない状況で

☆ CARDtools systems社のCARDtoolsなど．

試験をしなければならない．

• 制御対象がソフトウェア開発現場にない．

大規模プラントなどは，制御対象は客先に存在しソフトウェア開発現場には存在しない．また，制御対象と結合した試験（現地調整）には大きな準備とコストが必要である．

• 制御対象が未完成，あるいは数が不足．

組込みシステム開発では，ソフトウェアとハードウェアを並行に開発することが多い．制御対象のハードウェアの完成を待ってソフトウェアの試験を行うのでは，効率が悪く期限に間に合わない．また，ソフトウェア開発用の試作機があったとしても数が足りない．

このような場合，制御対象のシミュレータが試験の効率化に有効であるが，シミュレータの作成コスト，期間，品質が問題となる．シミュレータの作成期間が制御用ソフトウェア自体の開発期間より長くては意味がない．また，シミュレータの品質に問題があると，シミュレータのバグと試験対象のプログラムのバグが区別できずに混乱する．開発の規模によっては力ずくでシミュレータを作るのは効率的でない．そこで，（半）自動生成を含むシミュレータの作成支援技術が重要である．

3.1 プロセス制御用制御対象シミュレータ

大規模かつ信頼性が要求されるプロセス制御システムでは，比較的開発期間が長く開発予算も大きいことから，専用の制御対象シミュレータによる試験を行うことが多い．制御対象シミュレータは制御対象から入力されるデジタル値（例：バルブの開閉情報）および圧力・流量などのアナログ値を模擬する．さらに，制御対象への制御命令（入力）とセンサ情報（出力）との因果関係を模擬する（例：バルブ閉命令→流量をゼロとするセンサ情報）．また，異常や事故が発生した状況の模擬も重要な役割である．

電力などの分野では，ソフトウェアで制御対象の挙動を模擬するソフトウェアシミュレータだけでなく，制御対象のミニチュアをハードウェアで作るハードウェアシミュレータもある．ハードウェアシミュレータは微妙なタイミングの試験に使われる．

これらのソフトウェア試験用の制御対象シミュレータの多くは，メーカごとに内作されている．これは，制御対象の特殊性に依存する部分が多く，シミュレータの汎用化は難しいからである．しかし，化学プラントなど対象分野を限定すれば部品化／再利用によるシミュレータ自動生成が可能である[4), 5)]．

プロセス制御用プログラムは異常処理の比重が高い．異常処理を試験するには，あらかじめ制御対象シミュレータの中に異常時の挙動をモデル化しておく必要がある．しかし，異常時の挙動は正常時の挙動に比べて洗い出しが難しい．制御対象の異常かつ安全性にとって致命的な挙動の可能性の抽出とモデル化には，ハザード解析[6)]などの手法が有効である．

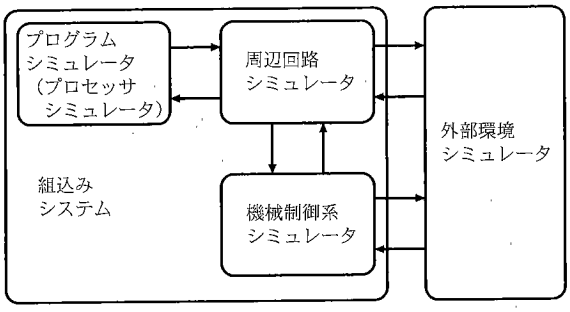


図-1 コードレベル協調シミュレーション

3.2 組込みシステム用制御対象シミュレータ

マイコン組込みシステムは，大規模プラント制御システムと異なり，専用のシミュレータは作成コストが効果に見合わないことが多い．ここでは，コードレベルおよび設計仕様レベルの制御対象シミュレータ作成支援について説明する．

(1) コードレベルの制御対象シミュレータ

制御用プログラムと制御対象を全体としてシミュレーションすることを**協調シミュレーション**（co-simulation）と呼ぶ．ここで，制御対象とは制御用プログラムおよびそれが実行されるプロセッサ以外の部分を指し，具体的には周辺回路，機械制御系，人間系を含む外部環境などである（**図-1**）．

プロセッサのシミュレータと周辺回路シミュレータによる協調シミュレーションは，近年いくつかのツールが商品化されつつある[7)]．周辺回路シミュレータには，(a) VHDLやVerilogHDLなどのハードウェア記述言語で記述された周辺回路モデルを直接シミュレーションするタイプと (b) Cなどのプログラミング言語で抽象度の高い挙動モデルを記述し，それらを結合してシミュレータを構成するタイプがある．前者はシミュレーションに時間がかかる点，後者は部品ごとにC言語モデルを記述する手間が課題である．後者に関しては，より抽象度の高い制御／データフローモデルからC言語モデルを自動生成するツール[☆2]もある．

機械制御系シミュレータに関しては，制御システムの開発支援／シミュレーションツールMatlab/Simlink（MathWorks社）で機械制御系を記述し，制御用プログラムシミュレータと連動させることで協調シミュレータを実現している例もある[☆3]．

☆2 C.A.E. Plus社のArchGenなど．

☆3 ガイオテクノロジー社のMsim-Gなど．

外部環境のシミュレータに関しては，汎用性が少ないこともあり，通常はプログラムシミュレータからのインタフェースが提供されているのみで，設計者がアプリケーションごとにCなどで記述する必要がある[8]．アプリケーション（たとえば，携帯電話の端末）を特定すれば外部環境シミュレータにおける部品化再利用も可能である．

(2) 設計仕様レベルの制御対象シミュレータ

制御プログラムの設計仕様を状態遷移表現で記述する場合，制御対象も同じ状態遷移表現で記述することにより，システム全体の協調シミュレーションが可能となる．また，外部環境シミュレータの一部としてのパネルの作成支援も有効である．たとえば，DVDプレイヤなどのAV機器のマイコン制御システムの試験では，疑似的なユーザの操作パネル（プレイ／ストップボタン，音量調整器）が必要である．仕様レベルのシミュレーション環境では，簡単にパネルを作成できるツールが提供されていることが多い．

(3) 設計仕様レベルとコードレベルの融合

最近，状態遷移仕様レベルとコードレベルのシミュレーションを融合した組込みシステムの開発環境が登場している．すなわち，仕様レベルのシミュレーションで作成した状態遷移表現やパネルなど試験情報をコードレベルのシミュレーションで再利用し，試験の効率化を図るアプローチである．具体例を下記に示す．

- 制御対象シミュレータの自動生成

状態遷移表現で記述した制御対象のモデルからコードレベルの制御対象シミュレータを自動生成する．

- 状態遷移仕様からの試験シナリオの自動生成

状態遷移表現で記述したシステム全体の仕様からコードを試験するための試験シナリオや試験仕様書を自動生成する☆4．

- コードレベルシミュレーションでの状態遷移表現の利用

コードレベルのシミュレーションの実行状況を，仕様レベルで用いた状態遷移表現に対応させて表示する．設計者は，仕様レベルと同じ環境でコードレベルの試験ができる．

- パネルの利用

仕様レベルシミュレーションで作成したパネルをコードレベルシミュレーションにおいても外部環境シミュレータの一部として再利用する．

4. 試験の自動化と網羅的試験

プログラムの試験の自動化に関してはさまざまな研究がある．とくに，GUIの試験の自動化に関しては，試験シナリオの登録とその自動実行を主な機能とする市販ツールが利用されている．制御用プログラムの試験の自動化では制御対象のシミュレータを用いる．すなわち，制御対象シミュレータに可能な挙動を自動的に生成させ，それらの挙動に対してプログラムが正しく対応できるかを確認する．制御対象シミュレータを用いることで，試験シナリオの登録と自動実行だけでなく，試験シナリオの網羅的な生成が可能となる．

4.1 非決定的な挙動のモデル化

制御用プログラムと制御対象のシミュレータを組み合わせることで閉システムを構成できる．このとき，閉システムの挙動のバリエーションは「制御対象の非決定性」としてモデル化される．この非決定性は以下の2点に分類できる．ここでは，制御対象がペトリネットでモデル化された場合の例で説明する．

- タイミングによる挙動の非決定性

ある時点で遷移可能なトランジションが複数ありトランジション間に競合が存在するとき，トランジションの時間的な実行順序により挙動が異なる（図-2 (a)）．

- 物理的要因による挙動の非決定性

外部環境の物理的要因で挙動が非決定的になる場合は，制御対象シミュレータの中に明示的に非決定性をモデル化する必要がある．たとえば，ハードウェアの故障に関しては，正常時のトランジションと故障時のトランジションの両方を用意し，故障するか否かは発火の非決定性としてモデル化する（図-2 (b)）．

4.2 網羅的シミュレーション

制御対象シミュレータにおける非決定的な挙動の選択肢を，別々のシナリオとしてすべて実行することにより，システムの挙動を網羅的にシミュレーションすることができる．これを，**網羅的シミュレーション**と

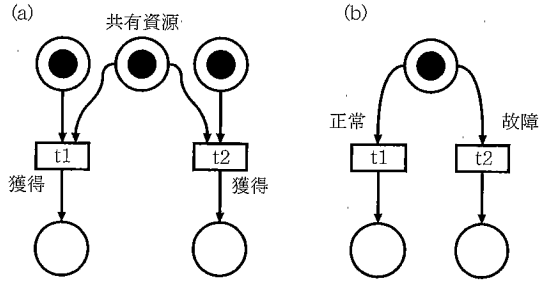


図-2 ペトリネットによる非決定性のモデル化

☆4 Teradyne社のTestMasterなど．

呼ぶ[9]．一般には，可能な挙動は無限に存在し，それらを網羅することは原理的に不可能であるが，制御システムは有限な状態および挙動をもつ離散事象システムとして定式化できる場合が多い．とくに，仕様が状態遷移表現で記述されている場合の仕様レベルのシミュレーションでは，網羅的シミュレーションによる試験機能を提供しているツールが多い．

4.3 シミュレーション結果の確認

通常のシミュレーションでは，プログラムの実行過程をビジュアルに表示し，ユーザが目視でチェックを行う．しかし，網羅的シミュレーションではユーザがすべてをチェックするのは難しいため，機械的にチェックする方法が必要である．

- シミュレーション実行時の自動チェック

デッドロックやインターロックの検出は，網羅的シミュレーション実行時に状態を生成する過程で検出できる．すなわち，あらかじめユーザが表明文Aや不変条件Iを検査項目として設定し，実行時に生成された各状態sがAやIを満たすか否かを随時機械的にチェックする．さらに，検査対象と並行して動く試験プロセスを導入することによって，時系列的な検査も可能である（**on-the-fly検証**）．

- シミュレーション終了後の一括自動チェック

網羅的シミュレーションの結果生成されたすべての実行結果をグローバル状態遷移グラフ（以下，**状態空間**と呼ぶ）で表現し，検査項目が満たされているかをチェックする．その代表例が**モデル検査法**（model checking）[10]である．モデル検査法における検査項目は時相論理やプロセス論理で記述される．状態空間における実時間制約の検査技術の研究も進んでいる[11]．

4.4 網羅的シミュレーションの効率化手法

網羅的シミュレーションでは起りうる挙動が組み合わせ的に多くなるので，能力の高い計算機および効率化手法が不可欠である．効率化手法には以下のアプローチがある．

- 状態および遷移の情報をなるべくコンパクトに記憶する．→BDD（Binary Decision Diagram）による状態空間の効率的記憶法[12]
- 状態を間引いて記憶する．間引いた状態が必要な場合は再生成する．→状態空間キャッシュ法（state-space caching）[13]
- 対象システムの階層的構造を利用して，検査項目に関係のない部分を縮約しながら状態空間を組み上げる．→積重ね法（compositional method）[14]
- 知的な枝刈りにより冗長なシナリオのシミュレーションを行わない．→半順序法（partial-order method）[15], [16]
- 与えられた計算機資源の範囲で網羅率を最大化する．→スーパートレースアルゴリズム（supertrace algorithm）[17]

上記の効率化手法を採り入れた網羅的シミュレーションツールは数多く提案されているが，ここではSPIN[17]とVeriSoft[18]を紹介する．SPINはプロトコル検査用に開発されたツールであるが，インターネットで入手可能であり幅広い分野で使われている．SPINでは，半順序法およびスーパートレースアルゴリズムが実装されている．SPINは制御プログラムおよび制御対象をPROMELAという専用の言語でモデル化する．これに対して，VeriSoftではCまたはC++で記述されたプログラムに対する網羅的シミュレーションが可能である．C/C++のプログラムの実行状態は大きいので，VeriSoftでは状態空間キャッシュ法を採用している．

5. 事例紹介

5.1 シーケンス制御システムの検査への適用事例

化学プラントのシーケンス制御システムにおけるプラントシミュレータ自動生成と網羅的シミュレーションの事例を紹介する[5], [19]．対象とするシーケンス制御用プログラミング言語はSFC（Sequential Function Chart）である．SFCで記述されたプログラムを試験するには制御対象であるプラントのシミュレータが必要である．SFCの機能の論理的な試験には，プラントの厳密なシミュレータは必ずしも必要でなく，定性的／離散的なシミュレータで十分である．

5.1.1 離散的プラントシミュレータ自動生成

化学プラントは，パイプ，バルブ，モータ，流量計，レベルセンサ，反応缶など100種類程度の比較的定型的な機器から構成される．そこで，あらかじめ機器単体の定性的／離散的挙動を記述し，シミュレーション部品として登録しておけば，機器間の接続情報からプラント全体の定性的／離散的な挙動を模擬するシミュ

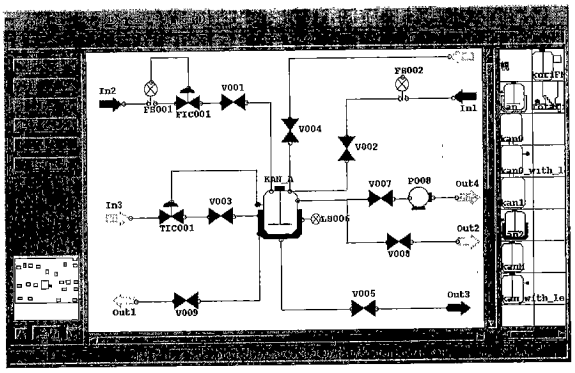


図-3 プロセスフロー図エディタ

レータを自動生成することが可能である．具体的には，ユーザは専用図形エディタを用いてあらかじめ部品ライブラリに登録された機器をメニューから選択し，キャンバス上に配置し，パイプで機器を接続しプロセスフロー図を作成する（**図-3**）．作成されたプロセスフロー図から機器の接続情報が抽出できるので，部品データベースから検索された各機器の挙動モデルを結合して，プラント全体の挙動を模擬するプラントシミュレータを自動生成する．ここで，機器単体の挙動モデルは，オブジェクト指向ルール型言語で記述される．通常，反応缶以外の機器は部品ライブラリから直接利用できるが，反応缶はプラントごとに固有の形状をしている場合が多い．しかし，この場合でもオブジェクト指向の継承機構を使えば，基本部品の組合せでプラント固有の反応缶の挙動を容易に記述することができる．

5.1.2 網羅的シミュレーション

SFCプログラムのシミュレータとプラントシミュレータを連動させて網羅的シミュレーションを行う．実行過程はSFC表示部およびプラントアニメーション表示部で目視チェックできる．また，状態数が爆発的に増加する場合の効率化技術として，半順序法と一部積重ね法を採用している．実際の化学プラント（バルブ21個，流量積算計5個，反応缶5個，モータ5個，制御用SFCプログラム8タスク/105ページ）に適用した事例では，状態空間の状態数は約3万であった．網羅的シミュレーションで生成された状態空間に対して，検査項目が満たされているか否かをモデル検査法によりチェックする．満たされない項目に関してはデバッグを行う．本事例では，状態空間の情報を基に，どのような経緯で検査項目が満たされなかったのかをシミュレーションで再現し，デバッグを支援する．

5.2 組込みシステムの試験環境

Harelらによって提案された組込みシステムの開発環境であるi-Logix社のStatemate MAGNUM[20]の試験環境を紹介する．ここでは，設計仕様レベルの試験で用いた制御対象の情報がコードレベルの試験でも利用できる点を中心に説明する（**図-4**）．Statemateによる制御用プログラム開発手順を以下に示す．

1．設計仕様記述

Statemateは，対象システムを3つの視点（構造的視点，機能的視点，動的視点）からとらえ，図形エディタで3つのチャート（Module-Chart, Activity-Chart, Statechart）を記述する．とくに，システムの挙動は階層的状態遷移機械であるStatechartで記述され，仕様レベルシミュレーションで中心的な役割を果たす．制御用ソフトウェアの設計仕様だけでなく，制御対象の挙動も同じStatechart（テストベンチと呼ぶ）で記述される．

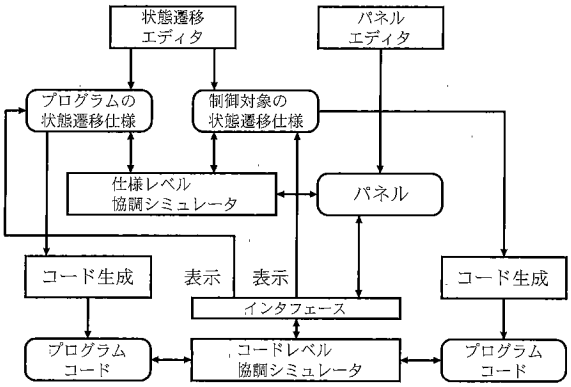


図-4 組込みシステムの試験環境

2．仮想パネル作成

パネル作成支援ツール（PGE: Panel Graphics Editor）により，組込み制御システムと人間系および外界とのインタフェースである操作パネルを作成する．

3．設計仕様試験

設計仕様レベルでの協調シミュレーションがインタラクティブおよびバッチモードで実行できる．シミュレーションの過程は，Statechartの状態遷移アニメーションで表示される．また，パネルによりユーザや外界とのやりとりをビジュアルに確認できる．さらに，網羅的シミュレーションによる検査項目のチェックやデッドロックの検出が可能である．

4．コード生成

3つのチャートから，C/Adaプログラムや VHDL/VerilogHDL記述を生成する．生成されたコードに対して，設計者はターゲット固有のコードを追加することができる．Statechartの状態にタスクを割り当てることにより，リアルタイムOS用のコード生成も可能である．ここで，制御対象も同じチャートで記述されているので，その挙動を模擬するプログラム（制御対象シミュレータ）も同様に生成できる．

5．コード試験

生成された制御プログラムおよび制御対象シミュレータと設計仕様試験で作成した操作パネルを結合したコードレベルシミュレーションが可能である．また，GBA（Graphic Back Animation）と呼ばれる機能により，設計仕様試験で行ったものと同様のStatechartの状態遷移アニメーションが，コードレベルシミュレーションと連動して表示できる．設計者にとっては，設計仕様試験とコード試験を同じ操作環境でシームレスに行うことができる．

近年，Statemate以外にも同様の機能をもつ商用ツール（テスコ社のZIPCやEmultek社のRapidなど）が販売されている．ZIPCは，状態遷移表で設計仕様を記述するツールであり，設計仕様試験とICEを用いたコード試験を同じ操作環境で行うことができる[21]．Rapidは対象製品の精密な仮想操作パネルを比較的容易に作成できる点に特徴がある．

6. おわりに

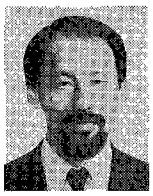
制御システムの複雑化および短納期化というニーズと開発環境の計算機の性能向上というシーズにより，プログラム開発環境上での制御用プログラムと制御対象の仕様レベルおよびコードレベルの（網羅的）協調シミュレーションは着実に浸透していくと思われる．また，本稿で紹介した仕様レベルとコードレベルの協調シミュレーションの融合もますます進展するであろう．今後は，制御対象の各構成要素（制御機器，論理回路）のシミュレーション用の挙動モデルの表現形式の標準化と，構成要素の販売に付随したシミュレーション用モデルの流通が重要な課題である．

謝辞　情報を提供していただいた伊藤忠テクノサイエンスの米澤紳一氏と東芝の五十嵐真悟氏に感謝します．

参考文献

1）河井，西山: 組み込みシステム用ソフトウェア開発環境，情報処理，Vol.37, No.9, pp.872-879（Sep. 1996）．
2）宿口雅弘: 組込みシステムのデバッグ手法，情報処理，Vol.38, No.10, pp.886-891（July 1997）．
3）栗山，大野: シミュレーションによるリアルタイムシステム開発環境，計測と制御，Vol.31, No.7, pp.811-815（1992）．
4）小林久浩他: 実行可能な仕様記述におけるプラント制御システムの環境のモデル化，情報処理学会論文誌，Vol.35, No.7, pp.1402-1409（July 1994）．
5）川田，内平: シーケンス制御ソフトウェア検証のためのプラントシミュレータの自動生成システム，計測自動制御学会システム情報関連合同シンポジウム，富山，pp.277-284（1995）．
6）Leveson, N. G.: Safeware: System Safety and Computers, Addison-Wesley（1995）．
7）ハードとソフトは並行設計が常識に，日経エレクトロニクス，1997.9.1, pp.67-85（1997）．
8）岡島，五十嵐，小山，安田: 組込みソフト開発支援のためのシステムシミュレーション環境，第51回情報処理学会全国大会論文集，Vol.5, pp.259-260（1995）．
9）内平，川田: 離散事象システムの網羅的シミュレーション，計測と制御，Vol.35, No.10, pp.763-769（1996）．
10）Clarke, E. M. et al.: Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications, ACM Trans. on Prog. Lang. Syst., Vol.8, No.2, pp.244-263（1986）．
11）米田友洋: ハードリアルタイムシステムの検証，情報処理，Vol.35, No.1, pp.41-47（Jan. 1994）．
12）Burch, J. R. et al.: Symbolic Model Checking: 10^{20} States and Beyond, Proc. 5th IEEE Symp. on Logic in Computer Science, pp.428-439（1990）．
13）Godefroid, P., Holzmann, G. J. and Pirottin, D.: State-Space Caching Revisited, Formal Methods in Systems Design,Vol.7, Kluwer Academic Publishers, pp. 227-241（1995）．
14）内平直志: 様相論理による並行プログラムの積重ね式検証法，電子情報通信学会論文誌，Vol.J75-DI, No.2（1992）．
15）Valmari, A.: Stubborn Sets for Reduced State Space Generation, Proc. 11th Internat. Conf. on Application and Theory of Petri Nets, Lecture Notes in Computer Science, Vol.483, Springer-Verlag, pp.491-515（1990）．
16）Godefroid, P., Peled, D. and Staskauskas, M. : Using Partial-Order Methods in the Formal Validation of Industrial Concurrent Programs, IEEE Trans. on Software Engineering, Vol.22, No.7, pp.496-507（1996）．
17）Holzmann, G. J.: Design and Validation of Computer Protocols, Prentice Hall（1991）．
水野他訳，コンピュータプロトコルの設計法，カットシステム（1994）．
18）Godefroid, P.: Model Checking for Programming Languages Using VeriSoft, Proc. 24th ACM Symposium on Principles of Programming Languages, pp.174-186（1997）．
19）Kawata, H. and Uchihira, N.: Practical Program Validation for Plant Control Systems Using SFC and Temporal Logic, 1996 IEEE International Conference on Systems, Man, and Cybernetics（SMC '96）, pp.3186-3191（1996）．
20）Harel, D. et al.: STATEMATE: A Working Environment for the Development of Complex Reactive Systems, IEEE Trans. on Software Engineering, Vol.14, No.4, pp.403-414（1990）．
21）沼田，奥村: マイコンソフトウェア用統合CASE環境，NEC技報，Vol.50, No.3, pp.245-248（1997）．

（平成9年10月29日受付）



内平　直志（正会員）
1959年生．1982年東京工業大学理学部情報科学科卒業．同年東京芝浦電気（株）（現（株）東芝）入社．現在，同社研究開発センターS&S研究所主任研究員．並行プログラムの高信頼化技術の研究開発に従事．現在，超逐次プログラミング技術による組込みシステム用Javaデバッガを開発中．1986年度情報処理学会論文賞受賞．ACM会員．
e-mail:uchi@ssel.toshiba.co.jp

川田　秀司（正会員）
1961年生．1986年学習院大学理学部数学科卒業．同年（株）東芝入社．1988年より1992年までICOTに出向．現在，（株）東芝S&S研究所主務．プログラムのテスト／検証に関する研究に従事．ソフトウエア科学会会員．
e-mail:kawata@ssel.toshiba.co.jp