Vol.2013-SLDM-160 No.31
Vol.2013-EMB-28 No.31
2013/3/14

モデル変換と振る舞い検証を活用した組み込み制御ソフトウェア設計法

田村 雅成[1,a)]　兪 明連[1,b)]　横山 孝典[1,c)]

概要：組み込み制御ソフトウェア開発の効率向上のため，Simulink モデルから実装を考慮した UML モデルを効率的に設計する手法を提案する．一般に，組み込み制御ソフトウェア開発は制御ロジックを作成する制御設計と制御ロジックを元にソフトウェアモデルを作成するソフトウェア設計の 2 段階で行う．近年の制御設計では MATLAB/Simulink を用いて制御ロジックを Simulink モデルとして設計することが多い．しかし，MATLAB/Simulink はプリエンプティブなマルチタスク環境でのタスク間のプリエンプションなどについては考慮していない．そのため，制御ロジックをそのままソフトウェアとして実装する場合，プリエンプションによってデータの不整合が発生する恐れがあり，同期や通信の処理を追加する必要がある．我々はこれまでに，制御設計からソフトウェア設計への移行をスムーズに行うため，Simulink モデルをソフトウェア設計に適した形の UML モデルに変換するモデル変換ツールを開発してきた．本論文では，生成された UML モデルをプリエンプティブなマルチタスク環境で実行した場合に起こりうるデータの不整合を，モデル検査ツール SPIN を用いて発見する手法を提案する．そして，モデル変換ツールが生成した UML モデルから検証用の PROMELA コードを自動生成するツールを開発する．これにより，効率的なソフトウェア設計を可能とする．

1. はじめに

自動車や産業用機器など広い分野で組み込み制御ソフトウェアが用いられ，その開発量は増大している．一般に，組み込み制御ソフトウェアの開発は制御設計とソフトウェア設計の 2 段階で行う．制御設計では制御設計者が制御ロジックの設計を行う．ソフトウェア設計では制御ロジックに基づいて，ソフトウェア設計者がソフトウェアの構造や振る舞いを設計する．

近年，制御設計は MATLAB/Simulink[1] 等の制御系 CAE/CAD ツールを用いたモデルベース開発が主流になってきている．MATLAB/Simulink では，ブロック線図形式のモデル（以下 Simulink モデル）で制御ロジックを記述する．そして，作成した Simulink モデルを用いてシミュレーションを行うことで，制御ロジックの誤りを早期に発見できる．また，MATLAB/Simulink の関連ツールである Real-Time Workshop Embedded Coder などを用いることで，Simulink モデルからソースコードの自動生成が可能である．しかし，MATLAB/Simulink のような制御系 CAE/CAD ツールでソフトウェア設計を行う場合，構造と振る舞いとを分離して記述することができない，タスクやリソースなどのモデルが定義されていない等といった問題がある [2]．したがって，MATLAB/Simulink のような制御系 CAE/CAD ツールはソフトウェア設計には使用せず，制御設計のみに使用するべきである．

また，MATLAB/Simulink のシミュレーションは制御ロジックそのものの検証が目的のため処理時間をゼロとして行っており，タスク間のプリエンプションなどは考慮していない．実際の組み込み制御システムはプリエンプティブなマルチタスク環境で動作させるため，タスク間のプリエンプションによってデータの不整合が生じる恐れがある．この問題を解決するにはタスク間の同期や通信の機構を組み込む必要がある．しかし，Simulink モデルから自動生成されるソースコードは構造化されておらず，可読性も良くないため，機能の追加・修正は容易ではない．これらの機能を効率よく組み込むためには，ソースコードレベルではなくモデルレベルで設計できることが望ましい．したがって，制御設計で開発した Simulink モデルを UML モデルに変換し，その UML モデルを元にソフトウェア設計を行う．制御ロジックを UML モデルに変換することで，制御ロジック以外の UML モデルと組み合わせたり，タスク間の同期や通信の機構を組み込むことが容易となる．

[1] 東京都市大学
Tokyo City University, Setagaya, Tokyo 158-8557, Japan
[a)] g1181524@tcu.ac.jp
[b)] yoo@cs.tcu.ac.jp
[c)] yokoyama@cs.tcu.ac.jp

我々はこれまでに，Simulink モデルをソフトウェア設計に適した形の UML モデルに変換するモデル変換ツールを開発している [3]．この変換ツールは，時間駆動オブジェクト指向ソフトウェア開発法 [4] に基づいて，Simulink モデルを UML モデルに変換する．この手法では制御ロジック中のデータのうち，入力値，出力値，観測値，推定値，目標値，制御パラメータ等の制御上重要なデータ（物理量）をオブジェクトとする．この変換方法に基づき，データ構造を表す構造図としてクラス図を，データの算出ロジックを表す振る舞い図としてシーケンス図を生成する．また，ソフトウェア設計では，プリエンプティブなマルチタスク環境を想定した振る舞いの検証が必要となる．UML モデルをベースにタスク間の同期や通信の機構を含めた検証を行うことが望ましい．

マルチタスク環境のような並行動作する処理の検証に，SPIN (Simple Promela Interpreter)[5] のようなモデル検査ツールを活用できる．モデル検査では，与えられた動作モデルが取りうる状態を網羅的に検査し，不正な状態を検出できる．SPIN では検証対象となる動作モデルを PROMELA (Process Meta Language) と呼ばれる専用の言語で記述する．UML モデルが表す振る舞いを PROMELA コードに変換して SPIN により検証する手法がいくつか提案されているが [6][7][8]，それらはいずれもシステムの状態遷移に主眼を置いており，本研究が対象としている，Simulink モデルに基づいて設計されたデータを算出する制御ロジックの検証には適していない．

本論文では，制御機能のみを表現した Simulink モデルから，プリエンプティブなマルチタスク環境での実装を考慮した UML モデルを効率的に設計する手法を提案する．モデル変換ツールにより生成された UML モデルは，プリエンプティブなマルチタスク環境での動作を考慮したものとはなっていない．そこで，算出・更新するデータの一貫性を SPIN を用いて検証することで，タスク間の同期や通信の機構が必要になる箇所を検出する．提案する手法では，データの算出・更新の回数に着目してデータの整合性の検証を行う．そして，シーケンス図の振る舞いから検証用の PROMELA コードを自動生成するツールを開発した．本ツールは算出処理を疑似的に表現した PROMELA コードを生成することで，データ算出の振る舞いを SPIN によって検証可能とする．

2. ソフトウェア設計工程

2.1 全体工程

我々が提案する組み込み制御ソフトウェアの開発工程を図 1 に示す．全体の開発工程は制御設計工程，ソフトウェア設計工程，実装（プログラミング）工程からなる．ソフトウェア設計工程では，制御設計で作成した Simulink モデルを元に，ソフトウェアの構造と振る舞いを UML で設計する．また，ソフトウェア設計工程は機能設計とタイミング設計の 2 段階からなる．

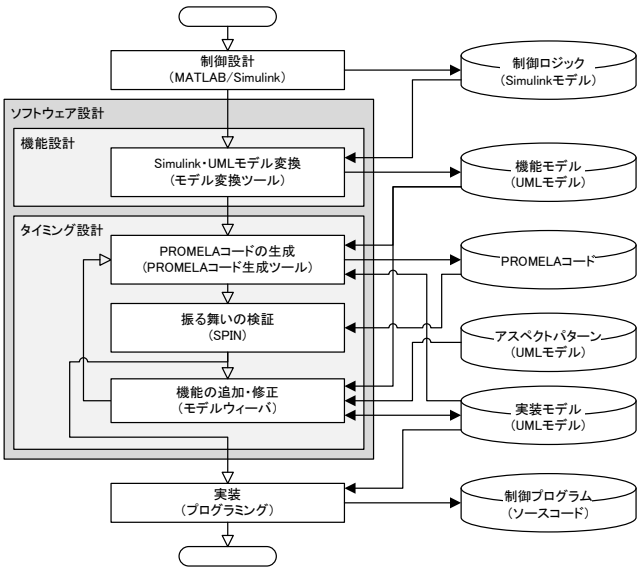

図 1 組み込み制御ソフトウェア開発の流れ
Fig. 1 Development Flow of Embedded Control Software

2.2 制御設計工程

制御設計工程では，MATLAB/Simulink を用いて制御ロジックを Simulink モデルとして設計する．Simulink モデルはソフトウェアとしての実装については考慮せず，制御機能のみを考慮して設計するものとする．例として自動車車間距離システムのうち，スロットル制御（Throttle Controller）の Simulink モデルを図 2 に示す．図 2 の Simulink モデルは入力されたエンジン回転数とエンジン状態，アクセル開度からスロットル開度を算出し出力する処理を表している．この Simulink モデルはエンジン回転数（Engine Revolution），エンジン状態（Engine Status），アクセル開度（Accelerator Opening）の入力を表す 3 つの Inport ブロック，トルク（Torque），スロットル開度（Throttle Opening）を算出する 2 つの Subsystem ブロック（Torque Calculation と Throttle Opening Calculation），算出したスロットル開度の出力を表す 1 つの Outport ブロックからなる．図 2 の上半分は階層化された Simulink モデルの上位階層であり，トルク，スロットル開度を算出する具体的な処理は Subsystem ブロックの下位階層に記述されている．この例では初めに，エンジン状態とアクセル開度のデータからトルクを算出する．次に，算出したトルクとエンジン回転数，エンジン状態のデータを用いてスロットル開度を算出し出力する．

2.3 機能設計

機能設計では，モデル変換ツールを用いて Simulink モデルをソフトウェア設計に適した形の UML モデル（機能モ

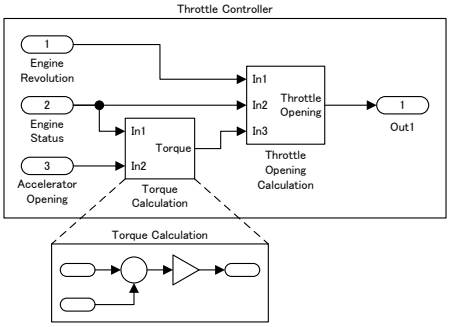

図 2 Simulink モデルの例
Fig. 2 Example Simulink Model

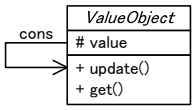

図 3 データ値オブジェクトの基底クラス
Fig. 3 Base Class of Data Object

デル）に変換する．我々のモデル変換方法は時間駆動オブジェクト指向ソフトウェア開発法 [4] に基づいている．この開発法では，入力値，出力値，観測値，推定値，目標値，制御パラメータ等の制御ロジック上重要なデータ（意味を持つ物理量に対応するデータ）をオブジェクトに対応付ける．そして，それらのデータとその値を算出する処理とをまとめてカプセル化し，1 つのクラスとする．

UML モデル上では，データを表すオブジェクトをデータ値オブジェクトとして記述する．データ値オブジェクトの基底クラス（ValueObject）を図 3 に示す．ValueObject はデータ値を記憶する属性 value，データ値を算出・更新する update メソッド，データ値を読み出す get メソッドを持つ．データ算出時（update メソッド実行時）に他のオブジェクトが持つデータが必要となる場合，そのオブジェクトの get メソッドを呼び出してデータを参照する．また，制御ロジックのブロック線図が表すデータフローをオブジェクト間のデータの参照とみなす．関連 cons は始端側クラスが終端側クラスのデータを参照することを表す．

図 2 の Simulink モデルから変換したクラス図を図 4 に示す．このクラス図はエンジン回転数，エンジン状態，アクセル開度，トルク，スロットル開度，スロットル制御の 6 つのクラスからなる．トルクはエンジン状態とアクセル開度，スロットル開度はトルクとエンジン回転数とエンジン状態のデータをそれぞれ参照する．なお，Inport ブロックのデータに対応するオブジェクトは update メソッドは持たない．スロットル制御は図 2 の Simulink モデル全体を表すクラスである．スロットル制御はトルクとスロットル開度の update メソッドを呼び出す exec メソッドを持つ．

図 2 の Simulink モデルから変換したシーケンス図を図 5 に示す．このシーケンス図はスロットル制御の exec メソッドの処理を表している．update メソッドの呼び出し順は

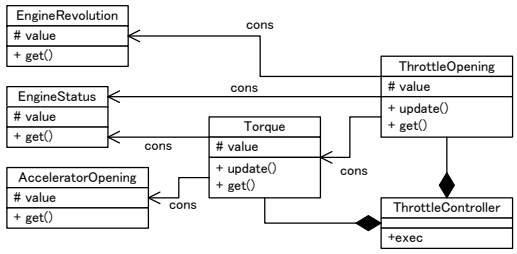

図 4 機能モデルの例（クラス図）
Fig. 4 Example Class Diagram of Functional Model

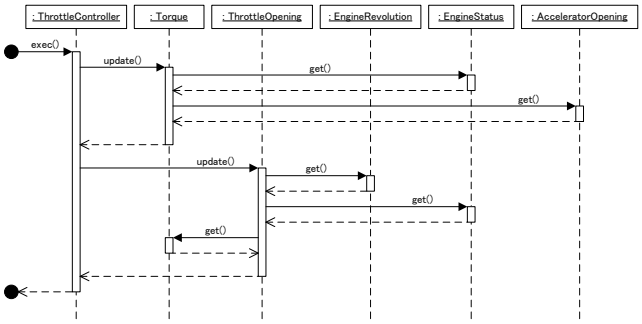

図 5 機能モデルの例（シーケンス図）
Fig. 5 Example Sequence Diagram of Functional Model

Simulink モデルのデータフローに基づいて決定する．スロットル制御の exec メソッドは，図 2 の Simulink モデルのデータフローに従い，トルク，スロットル開度の順に，update メソッドを呼び出している．また，トルクの update メソッドはエンジン状態とアクセル開度の get メソッドを，スロットル開度の update メソッドはエンジン回転数，エンジン状態，トルクの get メソッドを呼び出してデータを参照している．この exec メソッドの処理を制御周期に基づいて周期的に実行することで，制御処理が実現される．

2.4 タイミング設計

機能モデルは制御に関する機能のみを記述したものであり，ソフトウェアとして実装する上で必要な，タスク間の同期や通信の機構は考慮していない．タイミング設計では，プリエンプティブなマルチタスク環境で動作が可能なように，機能モデルに同期や通信の機構の追加・修正を行う．まず初めに，UML モデルを PROMELA コードに変換し，SPIN を用いて振る舞いの検証を行う．この検証によって，実装時に起こりうるデータの不整合を発見する．そして，検証によって発見されたデータの不整合を回避するための機構を機能モデルに追加する．

プリエンプションによるデータの不整合の例として，図 4 のシステムにおいてエンジン回転数，エンジン状態，アクセル開度の更新とトルク，スロットル開度の算出が異なる周期のタスクで実行される場合を考える．前者のタスクの方が後者のタスクよりも高優先度の場合，後者のタスクは前者のタスクによってプリエンプションされる可能性があ

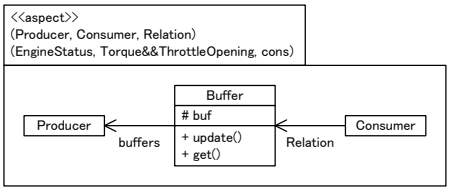

図 6 バッファリングのアスペクトパターン
Fig. 6 Aspect Pattern of Buffering

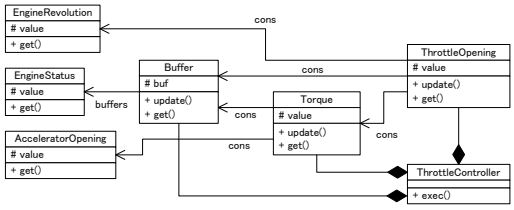

図 7 実装モデルの例（クラス図）
Fig. 7 Example Class Diagram of Implementation Model

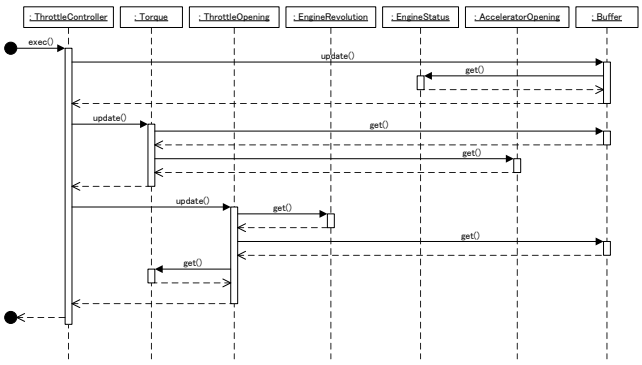

図 8 実装モデルの例（シーケンス図）
Fig. 8 Example Sequence Diagram of Implementation Model

る．このとき，後者のタスクのトルクの算出終了とスロットル開度の算出開始の間でプリエンプションが発生した場合，トルクの算出とスロットル開度の算出の間で使用するエンジン状態の値が異なり，データの整合性が取れなくなる．そのため，データの整合性を保つための排他制御やタスク間通信等の機構が必要となる．

このようなリアルタイム性などに関する非機能的な要求はモデルレベルのアスペクトパターンとしてライブラリ化されており，モデルウィーバを用いることにより UML モデルに織り込むことができる [9]．発見されたデータの不整合の解決法の 1 つにエンジン状態の値をバッファリングする手法がある．後者のタスクの実行開始時にエンジン状態の値をバッファに記憶しておき，トルクの算出とスロットル開度の算出で使用するエンジン状態の値をバッファから読み出すことでデータの整合性を保持できる．Producer オブジェクトと Consumer オブジェクト間を繋ぐ，バッファリングのアスペクトパターン（クラス図）を図 6 に示す．Buffer クラスはデータ値を記憶する属性 buf と Producer オブジェクトからデータ値を読み出して属性 buf に記憶する update メソッド，Consumer が属性 buf に記憶されたデータを読み出すための get メソッドを持つ．また，このクラス図がアスペクトパターンであることを表現するために，<<aspect>>という名前のパッケージでクラス図を囲う．織り込み先のモデルとの関連付けは<<aspect>>の下部の記述によって行う．アスペクトパターン中の要素の内，織り込み先のモデルに依存する部分を変数として記述する．図 6 では，Producer，Consumer，Relation が変数となる．その後に織り込み先のモデルが実際に持つ要素を記述することで，織り込み先のモデルと変数との対応付けを行う．図 6 では，変数 Producer には Engine Status が，変数 Consumer には Torque と Throttle Opening が，Relation には関連 cons がそれぞれ対応している．本論文では，実装を考慮した機構を加えた UML モデルを実装モデルと呼ぶ．図 4，図 5 のクラス図，シーケンス図にバッファリング機能のアスペクトパターンを織り込むことで，図 7，図 8 のような，バッファ（Buffer）が追加された実装モデルが得られる．

そして，機能追加後の実装モデルに対して再度検証を行い，問題が正しく回避されたか，新たな問題が発生していないかを確認する．これを問題が発見されなくなるまで繰り返すことで，実装を考慮した UML モデルを設計する．

2.5 実装工程

実装工程では，最終成果物となる制御プログラムのソースコードを作成する．ソースコードは，Simulink モデルから Embedded Coder[1] を用いて生成した制御ソースコードと，実装モデルから UML モデリングツールを用いて生成したクラスのスケルトンのソースコードを合成することによって生成する [10]．

3. 振る舞いの検証

3.1 振る舞いの検証方法

本論文で提案する振る舞いの検証は，UML モデルが表すデータの算出・更新の処理中でこのようなデータの不整合が起こりうるかを検証することを目的とする．PROMELA では検証対象となる動作モデルをプロセスの並行動作として表現する．生成する PROMELA コードにはシーケンス図が表すデータ算出の処理をプロセスとして記述する．UML モデルではメソッド内部の処理などはカプセル化されており，処理の詳細は隠蔽されている．そのため，UML モデルからデータを算出する処理の詳細なコードを生成するのは難しい．しかし，データの整合性を検証する際に着目すべきは，データの値そのものではなく，データがどのタイミングで更新されたものかということである．そこで，本研究で生成する PROMELA コード上ではデータの算出・更新をデータの更新回数の増加として扱い，データの更新回数を変数（以下更新カウンタ）に記憶させる．ま

た，データの読み出しを，対象データの更新カウンタの値を別の変数（以下参照データ）に記憶する処理に置き換える．これにより，データの算出処理を PROMELA コード上で疑似的に表現し，更新回数の比較からデータの整合性を検証する．

3.2 PROMELA コード

図 5 のシーケンス図から生成した PROMELA コードを図 9 に示す．この例では，エンジン回転数（EngineRevolution），エンジン状態（EngineStatus），アクセル開度（AcceleratorOpening），トルク（Torque），スロットル開度（ThrottleOpening）の 5 つのデータを更新カウンタに置き換えている．また，トルクからエンジン状態とアクセル開度へのデータの読み出し（get メソッドの呼び出し）を，2 つの参照データ（EngineStatus_Torque，AcceleratorOpening_Torque）に置き換えている．スロットル開度についても同様に，エンジン回転数，エンジン状態，トルクへのデータの読み出しを 3 つの参照データ（EngineRevolution_ThrottleOpening，EngineStatus_ThrottleOpening，Torque_ThrottleOpening）に置き換えている．

データの算出（update メソッドの処理）は inline での記述に置き換える．この例ではトルク，スロットル開度の update メソッドを inline に変換している．各 inline 内では，読み出し対象となるデータの更新カウンタの値を参照データに記憶後，update メソッドで算出するデータの更新カウンタをインクリメントする．これにより，参照データにはデータ算出時に使用されたデータの読み出し時点での更新回数が，更新カウンタにはデータの更新回数が記憶される．

シーケンス図に記述されたデータ算出処理を行うプロセス型（proctype）を定義する．データの算出は周期的に繰り返し行われるため，プロセスの処理は反復構造として記述する．反復構造は do-od で囲まれた部分を::の後に指定された条件を満たす限り繰り返すことを表す．本手法では::の後に条件を指定せず，無条件ループとして記述する．また，シーケンス図上で更新されていないデータ（この例ではエンジン回転数，エンジン状態，アクセル開度）は，実際の制御システムではコントローラモデルの外部で更新が行われているため，それらのデータを更新する別のプロセスを生成する．

一連のデータ算出処理後，データの整合性を検証するために読み出し先のデータが同じ参照データを比較し，値が等しいかを確認する．この例では，エンジン状態がトルクとスロットル開度から読み出されるため，対応する参照データを比較している．値が異なる場合，読み出されたデータは異なるタイミングで更新されたものであり，データの不整合が発生したことが確認される．比較に用いる assert 文は，与えられた条件式が満たされなかった場合にエラーを出力する．

```
int EngineRevolution = 0;
int EngineStatus = 0;
int AcceleratorOpening = 0;
int ThrottleOpening = 0;
int Torque = 0;

int EngineStatus_Torque = 0;
int AcceleratorOpening_Torque = 0;
int EngineRevolution_ThrottleOpening = 0;
int EngineStatus_ThrottleOpening = 0;
int Torque_ThrottleOpening = 0;

inline update_Torque()
{
        EngineStatus_Torque = EngineStatus;
        AcceleratorOpening_Torque = AcceleratorOpening;
        Torque = Torque + 1;
}
inline update_ThrottleOpening()
{
        EngineRevolution_ThrottleOpening = EngineRevolution;
        EngineStatus_ThrottleOpening = EngineStatus;
        Torque_ThrottleOpening = Torque;
        ThrottleOpening = ThrottleOpening + 1;
}

active proctype ThrottleController()
{
        do::
          update_Torque();
          update_ThrottleOpening();
          assert(EngineStatus_Torque==EngineStatus_ThrottleOpening);
        od;
}
active proctype update_EngineRevolution()
{
        do::
          EngineRevolution = EngineRevolution + 1;
        od;
}
active proctype update_EngineStatus()
{
        do::
          EngineStatus = EngineStatus + 1;
        od;
}
active proctype update_AcceleratorOpening()
{
        do::
          AcceleratorOpening = AcceleratorOpening + 1;
        od;
}
```

図 9 PROMELA コードの例
Fig. 9 Example PROMELA Code

図 9 の PROMELA コードで SPIN によるシミュレーションを行った結果を図 10 に示す．下線部分で assert 文がエラーを出力しており，エンジン状態についてデータの不整合が発生したことが確認できる．

3.3 PROMELA コード生成ツール

我々は第 3.2 節で述べたルールに基づき，シーケンス図から PROMELA コードを自動生成するツールを開発した．PROMELA コード生成ツールの内部処理を図 11 に示す．本ツールはシーケンス図が保存された XMI ファイルを入力とする．XMI ファイルは UML モデルの標準的なファイル保存形式である [11]．まずツールは，入力された XMI ファイルを解析し，変換に必要なデータを抽出した UML モデルデータを生成する．続いて，生成した UML モデルデータを元に PROMELA データ生成処理を行い，変数や inline，proctype などの内容を含んだ PROMELA データを生成する．最後に，PROMELA データを元に pml ファ

```
 0:    proc - (:root:) creates proc 0 (ThrottleController)
 0:    proc - (:root:) creates proc  1 (update_EngineRevolution)
 0:    proc - (:root:) creates proc  2 (update_EngineStatus)
 0:    proc - (:root:) creates proc  3 (update_AcceleratorOpening)
spin: warning, pan_in, global, 'int   AcceleratorOpening_Torque' variable is never used
spin: warning, pan_in, global, 'int   EngineRevolution_ThrottleOpening' variable is never used
spin: warning, pan_in, global, 'int   Torque_ThrottleOpening' variable is never used
 4:    proc  1 (update_EngineRevolution) pan_in:43 (state 3)       [.(goto)]
 5:    proc  3 (update_AcceleratorOpening) pan_in:59 (state 3)     [.(goto)]
11:    proc  2 (update_EngineStatus) pan_in:51 (state 3) [.(goto)]
12:    proc  1 (update_EngineRevolution) pan_in:43 (state 3)       [.(goto)]
15:    proc  1 (update_EngineRevolution) pan_in:43 (state 3)       [.(goto)]
18:    proc  3 (update_AcceleratorOpening) pan_in:59 (state 3)     [.(goto)]
20:    proc  2 (update_EngineStatus) pan_in:51 (state 3) [.(goto)]
23:    proc  1 (update_EngineRevolution) pan_in:43 (state 3)       [.(goto)]
24:    proc  2 (update_EngineStatus) pan_in:51 (state 3) [.(goto)]
29:    proc  1 (update_EngineRevolution) pan_in:43 (state 3)       [.(goto)]
spin: pan.in:33, Error: assertion violated
spin: text of failed assertion: assert(EngineStatus_Torque==EngineStatus_ThrottleOpening)
#processes: 4
30:    proc  3 (update_AcceleratorOpening) pan_in:59 (state 3)
30:    proc  2 (update_EngineStatus) pan_in:51 (state 3)
30:    proc  1 (update_EngineRevolution) pan_in:39 (state 2)
30:    proc  0 (ThrottleController) pan_in:33 (state 10)
4 processes created
```

図 10 シミュレーションの結果
Fig. 10 Result of Simulation

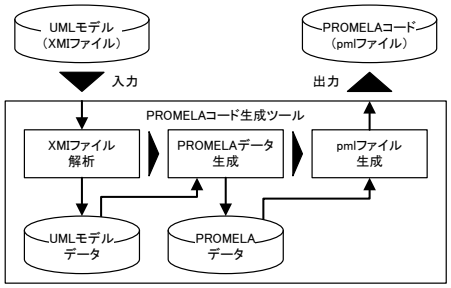

図 11 PROMELA コード生成ツール
Fig. 11 PROMELA Code Generation Tool

イル生成処理を行い，PROMELA コードのファイル保存形式である pml ファイルを出力する．

4. 適用実験

モデル変換ツールによって Simulink モデルから変換した UML モデルに対して，本検証手法の適用実験を行う．適用実験に用いる Simulink モデルは，制御設計者が実装を考慮せずに作成したものである必要がある．そこで，MathWorks 社 [1] が提供している燃料噴射システム，ハイブリッド駆動システム，ステッピングモータ制御システムの Simulink モデルを用いて適用実験を行った．最初に，それらの Simulink モデルをモデル変換ツールを用いて UML モデルに変換した．続いて，PROMELA コード生成ツールを用いて検証用の PROMELA コードを生成し，SPIN による検証を行った．そして，検証によって不整合の発生が確認されたデータをバッファリングするよう UML モデルを修正して再度検証を行い，それによってデータの不整合が回避されたことを確認した．

以上のように，ソフトウェアとしての実装を考慮していない Simulink モデルから変換した UML モデル上でのデータの不整合の発生を SPIN での検証により発見可能であることを確認した．

5. まとめ

本論文ではプリエンプティブなマルチタスク環境での実装を考慮したソフトウェア設計を効率的に行うため，Simulink モデルをソフトウェア設計に適した UML モデルに変換し，変換した UML モデル上でのデータの一貫性をモデル検査ツール SPIN によって検証する手法を提案した．また，UML モデルから検証用の PROMELA コードを自動生成するツールを開発した．これらにより，制御のみを考えた Simulink モデルを元に UML でのソフトウェア設計を効率的に行うことを可能とした．

本研究で提案した振る舞い検証は，データの不整合が発生する可能性の有無を確認するのみで発生条件の特定はできない．よって，タスクの周期や優先度など，より多くの条件を加えた検証を可能とすることが今後の課題となる．

謝辞 本研究の一部は JSPS 科研費 24500046 の助成を受けたものである．

参考文献

[1] The Math Works Inc.: http://www.mathworks.com/.
[2] Sangiovanni-Vincentelli, A. and Di Natale, M.: Embedded System Design for Automotive Applications, *IEEE Computer*, Vol.40, No.10, pp.42–51 (2007).
[3] 田村雅成, 神山達哉, 添田隆弘, 兪明連, 横山孝典: Simulink モデルと UML モデルを用いた組み込み制御ソフトウェア開発のためのモデル変換環境，情報処理学会論文誌, No.53, No.12，pp.2660–2670 (2012).
[4] 横山孝典，納谷英光，成沢文雄，倉垣智，永浦歩，今井崇明，鈴木昭二：組込み制御システムのための時間駆動オブジェクト指向ソフトウェア開発法，電子情報通信学会論文誌，Vol.34，No.2，pp.338–349 (2003).
[5] Gerard J.Holzmann: The Model Checker SPIN, *IEEE Transactions on Software Engineering*, Vol.23, No.5 (1997).
[6] D. Latella, I. Majzik, and M. Massink: Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker, *Formal Aspects of Computing*, pp. 637-664 (1999).
[7] Prasanta Bose: Automated Translation of UML Models of Architectures for Verification and Simulation Using SPIN, *Proceedings of the 14th IEEE international conference on Automated software engineering*, (1999).
[8] Li Jing, Li Jinhua: Zhang Fangning, Model checking UML activity diagrams with SPIN, *Proceedings of the International Conference on Computational Intelligence and Software Engineering*, (2009).
[9] Soeda, T., Yanagidate, Y. and Yokoyama T.: Embedded Control Software Design with Aspect Patterns, *Journal of the Chinese Institute of Engineers*, vol.34, Issue 2, pp.213–225 (2011).
[10] 神山達哉, 兪明連, 横山孝典: モデル変換とコード生成機能を有する組み込み制御ソフトウェア開発支援ツール, 情報処理学会第 74 回全国大会講演論文集, 1L-5 (2012).
[11] Object Management Group: *XML Metadata Interchange Specification,* Version 2.0.1 (2005).