情報処理学会研究報告
IPSJ SIG Technical Report

Vol.2012-SE-176 No.9
Vol.2012-EMB-25 No.9
2012/5/21

形式表現によるツールチェーンの構築

阿部 睦[†] 掛川 智央[††]

仕様記述のために，UML およびその派生言語が利用されることが多いが，その利用においてはいろいろな課題もある．そこで，報告者は仕様としての表現が求められる内容を網羅でき，実装に対する設計情報等も表現可能な形式表現の開発を行った．しかしながら，新たな表現方法を提案するだけでは，導入コストが大きい等の理由のため，実際の開発への適用が困難である．そこで，仕様記述のためだけでなく，開発プロセスの各工程で使用される各種ツールの連携に開発した形式表現を使用することによりツール間でのやりとりの自動化率を向上させることで開発効率の向上が期待できる．本報告では，開発した形式表現をツールチェーン構築に適用した結果について報告する．

Construct a Toolchain via Formal Description

MUTSUMI ABE[†] TOMOO KAKEGAWA[††]

UML and its derivative language are used for describing specification. However, there are various problems using these languages. we developed the description form that has capability to describe the specification and design information for system development. However, It is difficult to apply the new representation method to actual development because of the highly introduce cost, etc.Therefore, we applied the description form to tool chain construction for development automation. This paper reports the result of construct toolchain via developed description form.

1. はじめに

従来の仕様表現技術として UML[1]やその派生言語である SysML[2]，MARTE[3]等があるが，利用にあたっての課題があった．そのため，報告者は統一的に仕様が記述可能な形式表現（以下，開発した形式表現と呼ぶ）について検討を行い，例題を用いた評価を行った[4].

しかしながら，新たな表現方法を開発しただけでは導入コストや従来資産の扱い等の面で実際の開発への導入は難しい．そこで，表現済みの資産活用や開発した形式表現の導入コストの低減のため，開発した形式表現と UML との連携についても試みた[5]．このときの結果として，開発した形式表現からのコード生成や他のツールとの連携の可能性が得られた．

今回，開発した形式表現を仕様記述や単一のツールとの連携だけでなく，モデリングツールやその他の各種ツールに対して開発した形式表現を介して連携させることを行った. 連携方法としては, 開発した形式表現を中間表現とし，各種ツールとの連携を実現することでツールチェーンを構築することへの適用可能性について具体例を用いて検証を行った．

これにより，それぞれのツールを使う工程間での情報のやりとりが形式表現を介して行われることにより，工程間での情報伝達の誤りやツールへの入力工数の削減が見込める．また，用意されたツールチェーンを使うだけでなく，開発に適した必要なツール群を使ったツールチェーンの構築においてもその作業の軽減も見込める．

本報告では，開発した形式表現によるツールチェーンの構築にあたり，開発した形式表現による適用方法やそれに基づき試作した各種機能について説明し，具体例を用いて適用した結果について述べる．

２章では，開発した形式表現について概説する．

３章では，構築したツールチェーンについて，今回適用した具体例とツールチェーンで対応する領域を交えて述べる．

４章では，ツールチェーン構築にあたって試作した連携機能について述べる．

５章では，具体例にツールチェーンを適用した結果について述べる．

６章では, 今回の方法の有効性について述べ, まとめる．

2. 開発した形式表現の概要

まず，開発した形式表現[4][5]について概説する．

2.1 形式表現の構成要素

開発した形式表現ではデータを中心として表現する．その際，開発した形式表現では「データ」は機能や振る舞い等も含む広い範囲の物事/事物を対象としている．これは対象を表現する場合，何らかの手段により観測された結果，つまりデータであるということを意図して構成要素の名称として用いている．

上記の考え方に基づき，開発した形式表現で表現する際の基本的な構成要素はデータと制約の二つである．

データについては，上記で述べた通り，広い範囲の物事/事物を対象としている．制約については，開発した形式表

† 株式会社トヨタ IT 開発センター
Toyota InfoTechnology Center, Co., Ltd.
†† トヨタ自動車株式会社
Toyota Motor Corporation

ⓒ2012 Information Processing Society of Japan

現におけるデータの成立に必要な条件や規定となる．

これらを用いた表現作業としては，表現したい対象をまずデータと捉え，それを説明する形でデータに制約を接続する．その際，説明のために必要なデータを制約に接続することができ，そのデータを参照データと呼ぶ．これにより，定義したデータが他のデータと関係付けられる．

2.2 表現方法

表現のための追加の要素として複製データがあるが，それを含めて開発した形式表現の図形的な表現は以下の通りである．なお，それぞれの内容は文字列で記述する．

・データは矩形で表現

・制約は角括弧で表現

・データを説明する制約は説明するデータに対して矢印で接続

・参照データは制約に対して通常の線で接続

・複製データは破線の矩形で表現

複製データは同一のデータを表すため，図形表現として追加した．基本的にはデータと制約のみで表現可能であるが，複製データを導入することで，同じデータへの参照の線を減らすことができるため，記述のための領域を節約し，可読性を上げることができる．

図 1 に表現例を示す．「変数」という名前のデータは，抽象的なデータである．「名称」制約を持ち，「x」という名前になる．ここでは，x は複製データとして，どこか別の場所に定義されているものとする（もちろん，近傍にあれば，元となるデータ「x」から参照線を引くこともできる）．従って，x は参照データでもあり複製データでもある．x には「値」制約として，データ「0」を設定している．

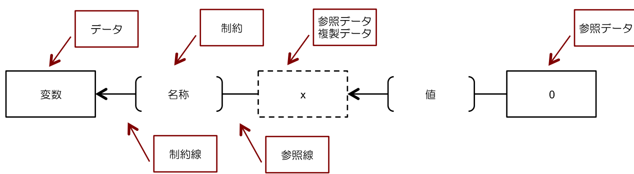

図 1　表現例

Fig 1 Expression example

2.3 類型

開発した形式表現では，データと制約の接続パターンを定義できる仕組みを用意した．これを類型と呼ぶ．接続パターンの登録即ち類型化により，記述の容易化や作成者間の表記の統一を図ることが可能となる．

今回の連携にあたっても連携のために必要な構造を定義するため，類型を活用している．

図 2 に類型の適用例を示す．図 2 で示す通り，類型とは，一つないしは複数の制約と無名データ（図 2 の左側の名称が記載されていないデータのこと）からなるパターンとして定義するものである．

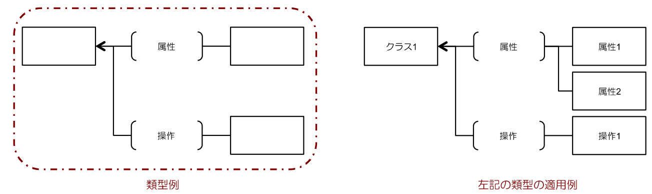

図 2　類型の適用例

Fig 2 Example of applying the type

類型の定義についても開発した形式表現で表現可能であり，定義も行っている[5].

3. ツールチェーンの構成

3.1 対象とする開発工程

今回構築するツールチェーンで対象とする開発工程としては，仕様（モデル）と仕様（モデル）検証とコード生成とする．これにより，検証された仕様（モデル）に基づいて実行コードを生成するという流れを確認する．

今回，開発した形式表現を介してツールを連携させるにあたり，適用領域としては状態遷移を対象とする．そのため，開発した形式表現で状態遷移を表現するため，有限状態機械（FSM）を開発した形式表現で構造を定義し，各ツールとのやりとりに利用する．

3.2 ツールチェーンを構成するツールと扱うデータ

ツールチェーンを構成するツールとしては以下のツールを使用した．

・仕様（モデル）ツールとして，UML ツールの Enterprise Architect（以下，EA）[6]および Simulink[7]

・検証ツールとして SPIN[8]

・実行用コード生成については試作

なお，今回は適用領域として状態遷移の部分であるため，EA については状態機械図を，Simulink ではステートマシン等のための拡張である Stateflow で記述された部分をモデルデータとして利用する．それぞれ，試作した取込み機能を用いて，開発した形式表現での FSM に変換される．

また，SPIN の利用においては検証対象が PROMELA という言語で記述されている必要があるため，検証用言語出力の検証用コードとして PROMELA のコードを生成し，モデル検査として SPIN での実行に用いる．

実行用コード生成については，実行用コードを動作させる機器としてタブレットデバイスを想定しており，そのデバイスの OS で動作するアプリケーションの開発言語が Objective-C を採用しているため，コード生成は Objective-C で出力を行う．なお，出力にあたっては，適用領域が状態遷移であるため，遷移部分のスケルトンを作成することでプログラムスケルトンを出力とする．

図 3 に対象とする作業工程おける作業内容や生成物等について示す．モデル検査以外のツールによる作業部分は試

ⓒ2012 Information Processing Society of Japan

作した機能が担う部分である．

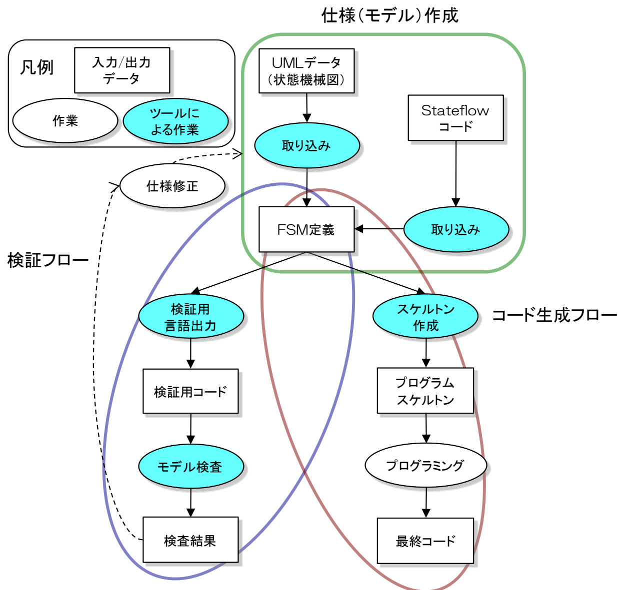

図 3　作業工程と生成物
Fig 3 Process and its product

4. 試作した各種機能

4.1 概要

今回のツールチェーン構築のため，次の 4 つの機能について試作を行った．

・UML ツールの状態機械図データの取込機能
・Simulink モデルの Stateflow 部分の取込機能
・PROMELA 生成機能
・Objective-C 生成機能

以下，それぞれについて述べる．

4.2 UML データ読込機能

UML ツールである EA で記述されたモデルデータの取込機能である．UML ツールからのデータ取込み機能については既に行っているが[5]，今回は状態機械図のデータを開発した形式表現で定義した FSM 定義に変換するところを試作した．

図 4 に EA で作成した例，図 5 に EA で表現した例を開発した形式表現の FSM に変換した結果を示す．

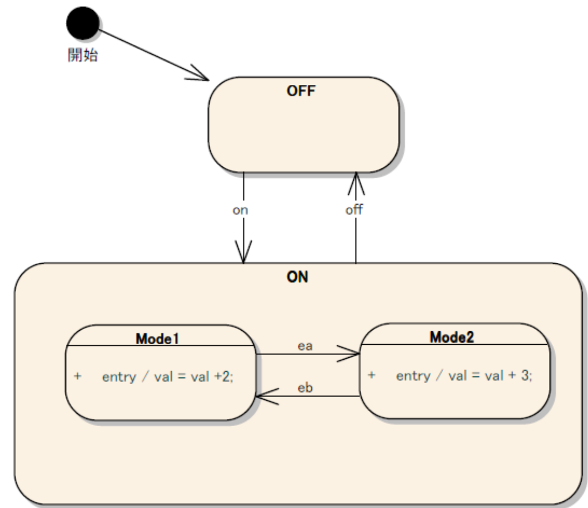

図 4 表現例
Fig 4 Expression example

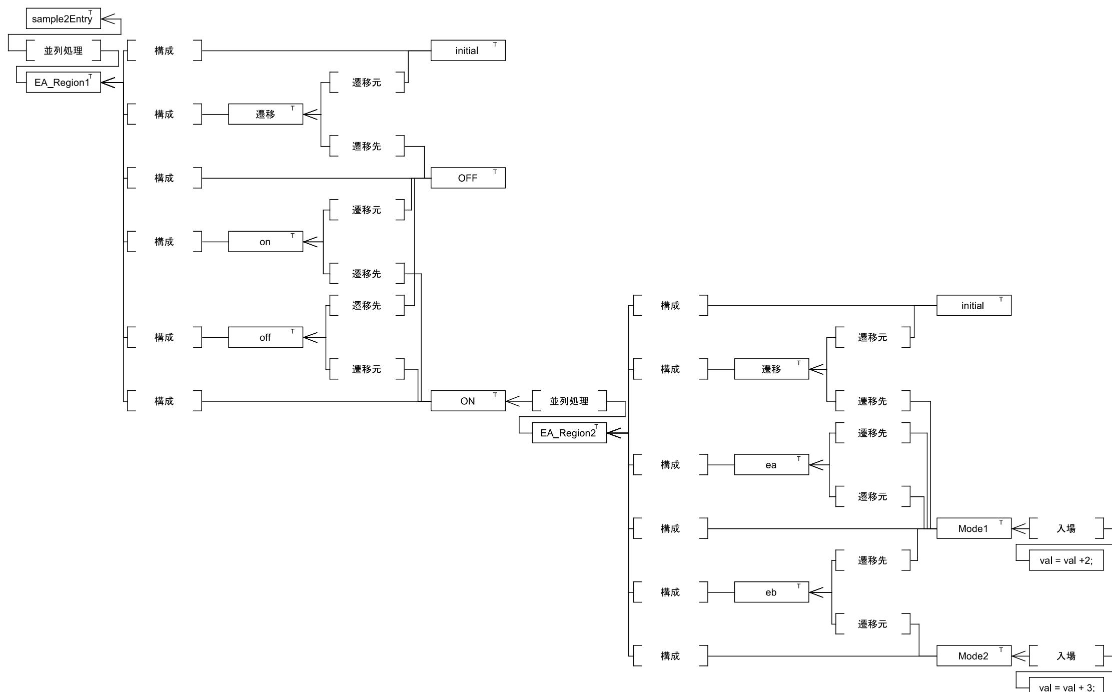

図 5 変換結果
Fig 5 Translation result

ⓒ2012 Information Processing Society of Japan

4.3 Stateflow 取込機能

Simulink モデルデータから状態遷移部分の記述を担う Stateflow 部分のデータの取込機能である．ここでは直接モデルデータから取込むのではなく，Simulink が持つ API を使って必要な情報を一旦 XML でファイルに出力した後に取込み, 開発した形式表現で定義した FSM に変換している．これは，今後のツールのバージョンアップ等でモデルデータのフォーマットが変更された場合を想定したためである．

4.4 PROMELA 生成機能

形式検証ツールである SPIN で処理できる言語である PROMELA のコードとして生成する機能である．図 6 に図 4 と図 5 で示した例に対して適用した結果を示す．

```
mtype =
{state_OFF,state_ON,state_ON_Mode1,state_ON_Mode2, nil};
mtype = {event_on,event_ea,event_eb,event_off};
mtype top_state_name = state_OFF;
mtype ON_state_name = nil;
chan ch_FSM = [0] of {mtype};
int val = 0;
active proctype proc_FSM() {
    mtype event;
    do
        ::ch_FSM?event->
            if
                ::(top_state_name == state_OFF && event
== event_on)->
                    top_state_name = state_ON;
                    val = val +2;
                    ON_state_name = state_ON_Mode1;
                ::(top_state_name == state_ON &&
ON_state_name == state_ON_Mode1 && event == event_ea)->
                    val = val + 3;
                    top_state_name = state_ON;
                    ON_state_name = state_ON_Mode2;
                ::(top_state_name == state_ON &&
ON_state_name == state_ON_Mode2 && event == event_eb)->
                    val = val +2;
                    top_state_name = state_ON;
                    ON_state_name = state_ON_Mode1;
                ::(top_state_name == state_ON && event
== event_off)->
                    top_state_name = state_OFF;
                ::else->skip
            fi;
    od
}
active proctype proc_main() {
    do
        ::ch_FSM!event_on;
        ::ch_FSM!event_ea;
        ::ch_FSM!event_eb;
        ::ch_FSM!event_off;
    od
}

#define p (val==99)
ltl p1 {<>p}
```

(a) PROMELA コード出力例

```
warning: for p.o. reduction to be valid the never claim must
be stutter-invariant
(never claims generated from LTL formulae are
stutter-invariant)
error: max search depth too small

(Spin Version 6.0.1 -- 16 December 2010)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of
claim)
        acceptance   cycles     - (not selected)
        invalid end states      - (disabled by never
claim)

State-vector 44 byte, depth reached 9999, errors: 0
    62161 states, stored
    48375 states, matched
   110536 transitions (= stored+matched)
        0 atomic steps
hash conflicts:        1340 (resolved)

Stats on memory usage (in Megabytes):
    4.268 equivalent memory usage for states
(stored*(State-vector + overhead))
    3.121 actual memory usage for states (compression:
73.12%)
          state-vector as stored = 25 byte + 28 byte
overhead
    4.000 memory used for hash table (-w19)
    0.458 memory used for DFS stack (-m10000)
    7.485 total actual memory usage

unreached in proctype proc_FSM
        sample_entry_action.pml:33, state 23, "-end-"
        (1 of 23 states)
unreached in proctype proc_main
        sample_entry_action.pml:42, state 8, "-end-"
        (1 of 8 states)
unreached in claim p1
        _spin_nvr.tmp:6, state 5, "-end-"
        (1 of 5 states)

pan: elapsed time 0.05 seconds
pan: rate    1243220 states/second
```

(b) SPIN での検証例

図 6　検証例

Fig 6 Verification example

なお，PROMELA コード出力例の最後の２行は検証用プロパティで，検証したい状態を追記したものであり PROMELA 生成機能を適用した段階では生成されない．上記では SPIN での検証用に追記している．もし，追記しない状態で SPIN にかけた場合は遷移しない状態等が情報として出力される．

4.5 Objective-C 生成機能

生成した実行コードをタブレットデバイスで動作させるため，Objective-C で実行用のプログラム生成を行う．

本機能により，ターゲットする OS が動作するタブレットデバイスでの動作が確認可能となっている．

図 7 に図 4 と図 5 で示した例に対して適用し，生成したプログラムの一部を示す．

```
#import "EA_Region1_StateMachine.h"
#import "EA_Region1_StateBase.h"
#import "EA_Region1_StateThread.h"
#import "common.h"
#import "EA_Region1_ON_Mode1.h"
#import "EA_Region1_ON.h"
#import "EA_Region1_OFF.h"
#import "EA_Region1_ON_Mode2.h"

static EA_Region1_StateMachine *myself_ = nil;

@implementation EA_Region1_StateMachine

@synthesize current = current_;
@synthesize threads = threads_;

+ (EA_Region1_StateMachine *)sharedInstance
{
    @synchronized (self) {
        if (myself_ == nil) {
            myself_ = [[super allocWithZone:NULL] init];
        }
    }
    return myself_;
}

+ (id)allocWithZone:(NSZone *)zone
{
    return [[self sharedInstance] retain];
}
```

図 7 生成したプログラムの一部

Fig 7 Code sample

5. ツールチェーンの適用

ここでは，Simulink で記述されたサービスモデルについて構築したツールチェーンを適用し，各作業工程での具体的な作業内容や得られる結果について述べる．

5.1 題材となるサービス

ここでは，ドアロックの開閉やエンジンの始動･停止等がキーの差し込み操作無しで可能な，いわゆるスマートキーシステムを題材とし，その中でもドアロック等の動作決定に関わる部分について適用した．図 8 に題材としたスマートキーシステムの Simulink と Stateflow のモデルの一部を示す．

図 8 のように Stateflow のモデルでは，状態機械を構成する複数の状態の定義（Stateflow モデル右上）や主に内部処理や複雑な条件判断をファンクションとして記述（Stateflow モデル上部以外の左側）している．

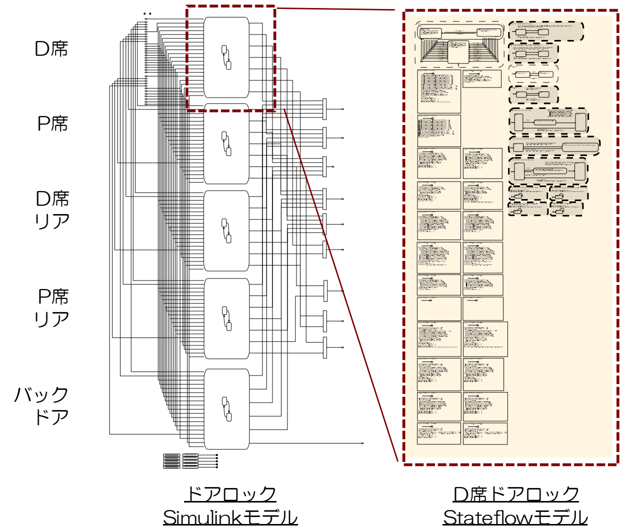

図 8 題材モデル

Fig 8 Material model

5.2 適用結果

図 9 に各ツールにおける生成結果を示す．

生成されるコードは検証フロー側では PROMELA，実行用コード生成フロー側は Objective-C である．

手作業によるコード追加は，検証フロー側である PROMELA では約 3 分の 1，実行用コード生成フロー側の Objective-C では GUI 部分も含めて約 4 分の 1 で済んでおり，多少冗長な面があるにせよ，全コードを手作業で作成するのに比べ，大幅な工数節約となっている．

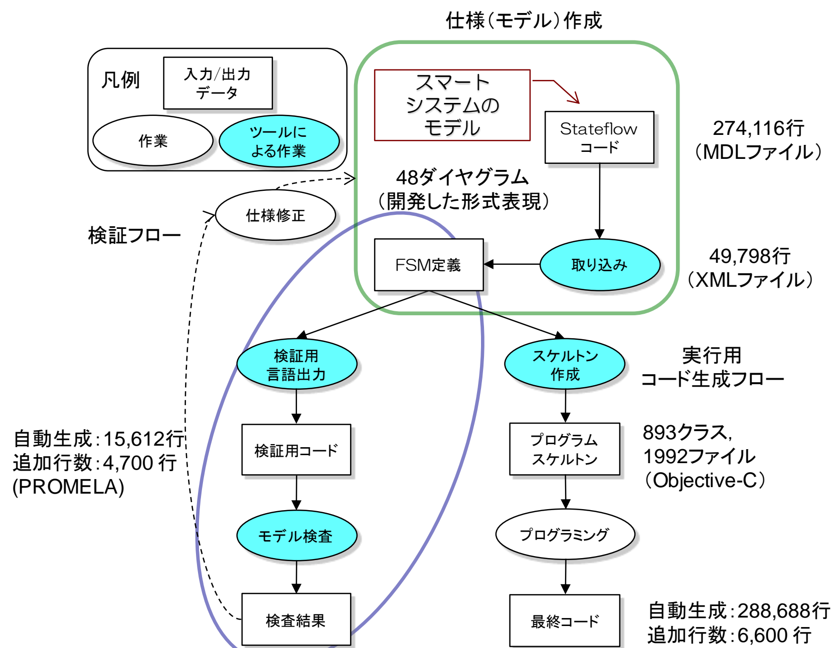

図 9 適用結果

Fig 9 Application result

ⓒ2012 Information Processing Society of Japan

なお，モデルデータの取込みや各コード生成における詳細については，次節以降で述べる．

5.3 モデルデータの取込み

Stateflow コードの取込みは，まずは MATLAB が持つ API を利用した XML 作成プラグインによりモデル全体の XML ファイルを得る．その上で，対象とした部分の Stateflow 部分を抽出する．今回の抽出では，48 のファイルを抽出した．その上で，抽出した XML ファイル毎，開発した形式表現に変換した．

5.4 SPIN による検証

開発した形式表現の形式となった 48 のファイルについてそれぞれ PROMELA のコードを出力した．

SPIN を適用するにあたっては，Stateflow コードでは変数宣言や定義等の情報が欠落しているため，それらの情報を補った．また，Stateflow で記述されている MATLAB 関数についても同等の内容で修正した．修正は修正前をコメント化した上で実施しており，図 9 では追加行数として記載している．

SPIN での検証においては，全状態空間探索を行った場合，今回の Stateflow でも大きめのものはメモリ不足で検証できなかった．そのため，オプションにより，省メモリモードで検証を行った．このモードでの検証で実行時間が一番大きかったケースでは約 50 分かかっていたが，同じものを全状態空間探索を適用すると約 2 分で割当可能なメモリを使い切り，処理が中断していた．

5.4.1 検証で発見した問題

（1）修正作業中に見つかった問題

接続を持たない状態があった．また，問題という程ではないが，同じ関数定義が複数個所で行われていたり，変数命名規則に一貫性が無い等の記述も見つかった．

（2）SPIN による検証で発見した問題

初期化処理の評価順序や遷移の重複，実行が保証されない処理，遷移条件の矛盾等が見つかった．

5.4.2 検証結果のフィードバック

SPIN による検証によって問題が見つかったとしても，SPIN について経験の無いモデル作成者ではモデルのどこに問題があったかを確認することが困難である．

この点については，ツールに精通した人材を置くことで対処可能ではあるが，工数削減という点ではツール側での対応が望まれる．

5.5 実行用コードの生成

実行用コードについて，開発した形式表現で取り込んだファイルから Objective-C のプログラムスケルトンを生成した．

実行用コードの生成におけるプログラムの追加作業としては，次の２点となる．

・Simulink で記述されている各 Stateflow の連携およびデータ変換処理

・アプリケーション化するための GUI 部分

5.5.1 実行用コードの動作

生成されたプログラムスケルトンに GUI 部分等のプログラムを追加し，実行可能としたアプリのタブレットデバイスでの実行例を図 10 に示す．このアプリケーションでは，ドライバーおよびスマートキーの車での位置の設定やロック/アンロック，ドアの開閉等を操作可能としている．

図 10 実行例

Fig 10 Execution example

5.5.2 コード生成による作業の軽減

5.2 で述べたように，自動生成部分に比べ，追加したプログラムの量は少なく，工数が節約されていることがわかるが，プログラムの追加にあたった開発者からも，ゼロからコーディングすることを考えるとずいぶん作業は楽だったとの報告があった．

ⓒ2012 Information Processing Society of Japan

6. まとめ

6.1 開発した形式表現を介したツールチェーン構築の有効性

今回の形式表現を用いたツールチェーン構築についての関連研究としては，モデルを PROMELA に変換し検証する研究[9]，Simulink モデルを UML モデルに変換するツールに関する研究[10]等がある．また，実行用のコード生成についてはSimulinkのベンダー自体が製品を出しているが，並列性を高めるための研究[11]がある．

これらの研究は基本的にモデルからモデル（コード）という流れであるため，そのモデル（コード）が関係する作業工程の関係で閉じてしまう．

本報告でのツールチェーン構築では，対象とするツール群で共通に使用するデータ構造を開発した形式表現で定義し，そこにツールで取り扱うデータの取込みや出力を行うことで連携を実現した．この方法では，共通に使用するデータ構造に対応したツールであれば，開発した形式表現との受け渡し機能を用意することにより，ツールチェーンに参加することが可能となることを意味する．

今回の試作では，FSM を共通のデータ構造としたが，各ツールが扱う情報を最大限取り込むことで，状態遷移以外の部分の連携も可能と考える．例えば，今回のプログラム生成では状態遷移のスケルトンのみだが，Simulink モデルの各状態で記述されている内部処理等も取り込み，スケルトン内部での処理プログラムに変換できれば自動生成で得られるプログラムの割合を増加させ，更なる省力化が可能となる．

また，上記で述べた関連研究でも変換にあたっては中間モデル等を用いていることから，それらの中間モデルを開発した形式表現で表現すること等により，関連研究で得られた成果を取り込むことも可能と考える．

本報告に関連したその他の研究として，モデル間の整合性についての検討[12]がある．今回のツールチェーン構築では各ツールでの作業を想定した連携となっているため，モデル間の整合性については明確である．また，作業工程に合せて連携させるツールの種類を増加させることでモデル群としてのスケーラビリティが得られると共に開発効率も向上すると考える．

謝辞　ツールの試作や評価作業等で協力いただいた株式会社ニルソフトウェアのみなさまに感謝します．

参考文献

[1] OMG UML, http://www.uml.org/
[2] OMG SysML, http://www.omgsysml.org/
[3] OMG MARTE, http://www.omg.org/omgmarte/
[4]阿部睦：統一的表現に向けた仕様記述方法，情報処理学会研究報告，Vol.2011-SE-171 No.24
[5]阿部睦：統一的表現に向けた形式表現と UML との連携，情報処理学会研究報告，Vol.2011-SE-174 No.10
[6] Enterprise Architect, http://www.sparxsystems.jp/ea.htm
[7] Simulink, http://www.mathworks.co.jp/products/simulink/
[8]Holzmann,G.J.:The Spin Model Checker : Primer and Reference Manual,　Addison-Wesley,2004.
[9]柳　翔太，小飼　敬，上田賀一，大久保　訓，高橋勇喜，中野利彦：情報制御システム記述モデルの検証項目記述と SPIN による確認，情報処理学会研究報告，Vol.2011-SE-171 No.10
[10]神山達哉，添田隆弘，兪 明連，横山孝典：組み込み制御ソフトウェア開発のための Simulink・UML モデル変換ツール：情報処理学会研究報告，Vol.2011-EMB-18 No.7
[11] 久村孝寛，枝廣正人，中村祐一，石浦菜岐佐，武内良典，今井正治：Simulink モデルにもとづいた並列 C コード生成，信学技報，CPSY2010-80(2010-03)
[12] 岸知二：スケーラブルなモデリング技法に関する考察，情報処理学会研究報告，Vol.2011-SE-173 No.10

ⓒ2012 Information Processing Society of Japan