

状態遷移言語による組込みソフトウェア開発

小倉信彦[†] 谷川郁太^{††} 渡辺晴美^{††}

組込みソフトウェアでは外部刺激へ反応する振る舞いの記述に状態遷移モデルが有効である。我々は、一般のC言語と混在した形式で状態遷移モデルを記述できる実践的な言語を開発し、これまで、複数の異なるターゲットの開発例題へ適用してきた。本発表では、使用言語、開発事例について紹介する。

A Case Study of Embedded Software Development based on State Machine Programming Language

Nobuhiko Ogura[†], Ikuta Tanigawa^{††},
Harumi Watanabe^{††}

The article presents an extension to the programming language C. The language has additional syntax to define state transition models which often used to represent behaviors of reactive systems. This extension narrows the gap between implementations and state models and reduces difficulties of managing both models and programs and learning special tools on models or model description language. We also show case studies with several different target processors, toolchains, and operating systems.

1. はじめに

組込みシステムではしばしば、外部刺激へタイムリーに反応することを要求され、その刺激の順序やタイミングは外部環境によって左右される。このような場合、状態遷移モデルに基づく動作の記述が必要となる。これまで状態遷移モデルを GUI エディタや、特別な言語を用いて記述することにより、設計やプログラムの自動生成する方法やツールが提案されてきた[1][2][3][4][5]。これらのツールは十分な実用性を持つものも多いが、

- プログラミングで用いる物とは異なる特別な文法やツール使用方法の学習が必要となる、
- ツールのコンフィギュレーションが複雑でシステム記述の一部に用いる場合他の部分との連携に手間が掛かる、
- 既存プロジェクトの統合開発環境や構成管理方法への適合が問題になることがある、

などの理由で導入に課題がある。そのため、我々は、一般のC言語と混在した形式で記述できる実践的な言語を開発した。

提案言語ではC言語の「薄い拡張」として状態遷移マシンを記述可能とすることにより、システムの一部、必要な部分へ容易に導入できる言語を目指した。まず状態遷移マシンの定義をC言語類似の構文で行い、C言語の構文の宣言、定義部と複数の状態遷移マシンの定義を混在して記述する。状態マシンに関わるイベント発行等の操作を、処理を記述するC言語の文中に混ぜて記述する。また、トランスレータにより(C言語のコンパイル単位と同じ)ソースファイル単位でC言語へ変換、コンパイルし、他の言語で書かれたモジュールやライブラリとリンクする。

記述されたソースファイルから、自動レイアウトによるモデル図の生成、およびC言語を通じた実行形式への変換を行うトランスレータを作成した。C言語への変換を通じた拡張言語を様々なプラットフォームへの適用を考える場合、コンパイラの処理系に依存する非標準の拡張が問題となるため、ヘッダファイルなどから除去拡張構文を避けて変換を行う仕組みを持つことにより様々なターゲットに容易に適用可能となるよう工夫した。

また、組込みソフトウェアの開発例題である ET ロボットコンテスト課題(2008 年)

[†] 東京都市大学 環境情報学部

Tokyo City University, Faculty of Environmental and Information Studies

^{††} 東海大学 情報通信学部 組込みソフトウェア工学科

Tokai University, Department of Embedded Technology

やその RTOS 拡張課題(同 2010 年)の開発[6], MDD ロボットチャレンジ(2008-2009 年)における飛行船ロボット開発[7][8], 著者らの所属する東海大学組込みソフトウェア工学科の組込み課題(通信プログラム)等を通して, H8, ARM(TOPPERS/ATK), M16C, x86(Windows), R8C などの複数のターゲットへ適用した. 本発表では, 言語, 開発事例について紹介する.

2. 状態マシン言語の概要

提案言語は C 言語を拡張することにより, C 言語と混在した形式で状態遷移マシンを記述可能としたものである.

トランスレータを用いて状態遷移マシンの記述を含む本言語ソースコードより C 言語へ変換を行い, 通常のコモジュールとリンクさせプログラムが動作する. また, 別のトランスレータを用いて, 自動的にレイアウトされた状態遷移図を本言語ソースコードより生成することも可能である. 図 1 に状態遷移マシン記述言語からの変換の構成を記す.

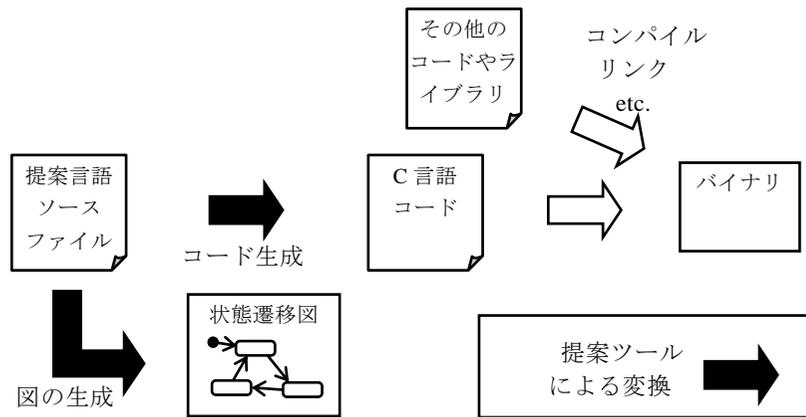


図 1 提案ツール(トランスレータ)の概要

構文拡張の主たる点は以下の通りである.

- C 言語のソースコードはトップレベルでは宣言(変数定義を含む)と関数定義を並べた構造となっているが, 提案言語ではこれらと併置して状態マシンの宣言定義を行う.
- C 言語の処理は複合文の中に文のリストを置き記述するが, 文の一つの式の中に,

初期化やイベント発行などの状態モデルの操作を記述する.

また, 以下により教育, 各種プラットフォームへの適用が容易となるように工夫している.

- 状態遷移マシンの理解性を向上させるために状態, イベント, ガード条件, アクションのそれぞれに, 自然言語の表記を併記する補助的な構文を追加した.
- C 言語処理系の独自拡張構文に則った記述がしばしばヘッダファイル中に表れ, 本言語のような拡張言語を様々なプラットフォームへ適用する際の障害となるため#include 指令を迂回するための pragma 指令を追加した.

最も簡単なプログラムの例を示す. また, ツールにより生成された状態遷移図を図 2, 図 3 に示す. 各構文は次節で説明する.

```
#include"sfr_r829.h"  
extern volatile unsigned char countup_10ms_a;
```

C 言語の宣言・定義部分(一部省略)

```
stm_def  
{  
state:  
s_wait = {is_initial, "待ち状態"};  
s_on = {"ON 状態"};  
event:  
void button1_pushed() "sw3 が押された";  
void null_event() "空イベント";  
transition:  
void button1_pushed()  
s_wait -> s_on  
"ledを点灯し, 経過時間をリセットする"  
{  
p1_1=0; // (LED on)  
countup_10ms_a=0;  
}  
void null_event()  
[countup_10ms_a >= 100] "一秒経過した"  
s_on -> s_wait  
"ledを消灯する"  
{  
p1_1=1; // (LED off)  
}  
}
```

```

} stm1;

void main(void)
{
    init_devices();
    stm1.init(); // 状態マシン初期化

    sw3_prev_on=0;

    while(1)
    {
        if (SW3_IS_ON)
        {
            if (!sw3_prev_on)
            { // Button Down
                stm1.button1_pushed();
            }
            sw3_prev_on=1;
        }
        else
        {
            sw3_prev_on=0;
        }
        stm1.null_event();
    }
}

```

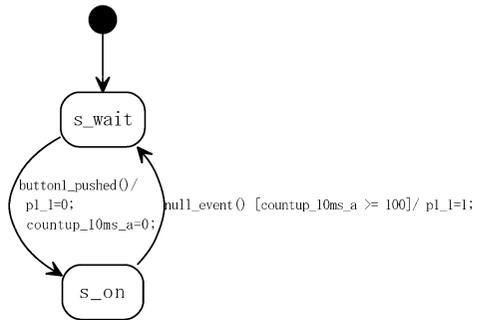


図 2 生成された状態遷移図

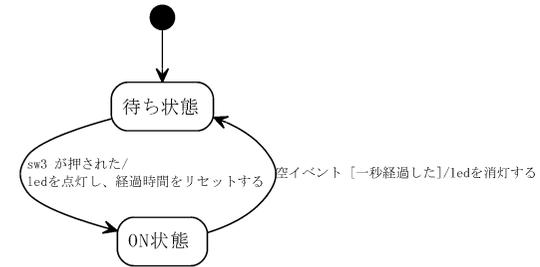


図 3 生成された別の形式の状態遷移図

3. 状態マシン言語の構文

本節では、提案言語の、状態遷移マシンの定義、定義された状態遷移マシンの操作の構文を説明し、また構文外のコンパイル指令についても説明する。

3.1 状態定義部

状態の定義は以下のように行う。記述できる場所は、C言語の関数定義が可能な場所と同じである。

```

stm_def
{
    data:
        データ定義部
    state:
        状態定義部
    event:
        イベント定義部
    transition:
        遷移定義部
} 状態遷移マシン名 1, 状態遷移マシン名 2, ... ;

```

この定義のために `stm_def` を予約語として追加した(外部宣言のための `extern_stm` も予約語としている)。

データ定義部には次のようにして、状態遷移マシンに属する変数を定義できる(状態を表す変数を明示的に定義する必要はない)。

```
struct {変数定義リスト};
```

状態定義部では、状態遷移マシンの状態を全て列挙する。各状態は、状態名とその属性を記述する。is_initial, is_final を書くことによってそれぞれ該当状態が初期状態、終了状態であることを示す。

```
状態名={"状態説明", is_initial, is_final};
```

イベント定義部では、次のような関数宣言と似た形で、状態マシンが受け入れるイベントを列挙する。

```
戻り型 イベント名 (パラメータリスト) "イベント説明";
```

遷移定義部では、次の形式で遷移を列挙する。ガード条件にはC言語の条件式と同等の表現を、アクション動作記述は複合文の文リストを記述することが出来る。また、遷移元の状態は複数指定することが出来る。

```
戻り型 イベント名 (パラメータリスト)
[ガード条件] "ガード条件説明"
遷移元状態 1, 遷移元状態 2, ... -> 遷移先状態
"アクション説明"
{
    アクション動作記述
}
```

3.2 状態操作記述

状態マシンに対する操作は次の4種類あり、

```
状態マシン名. イベント名 (パラメータ)
状態マシン名. init()
状態マシン名. is_final()
状態マシン名. state_is (状態名)
```

それぞれ、イベントの発行、初期化、終了状態の判定、状態の判定を行う。全て式と見なされる。(ただし、状態マシンがどのような状態を持ちうるか、どの状態であるか、外部から分かるべきではない、という考えに立つと、4番目の操作は許されないことが望ましい。)

3.3 構文外の指令

C言語のコンパイラは独自の型や予約語、ビット参照構文、割り込みハンドラ関数定義構文、メモリバンクの指定、属性記述など、処理系依存の非標準な拡張を持つことがある。組込みシステムではターゲットによって様々なコンパイラを用い、また、ハードウェア機能を有効に使うための拡張がされていることが多い。C言語と混在した記述の可能なC言語の拡張を行う提案言語では、ソースファイル中に非標準の構文が用いられている場合、個別に拡張構文に対応しない限り、取り扱うことが出来ない。プリプロセッサを掛けて、ヘッダファイルをインクルードすることにより、初めて正当な構文となるC言語の特性から、特に、ハードウェアサポート、ライブラリ、ミドルウェア、OS等のヘッダファイル中に含まれている場合に問題となる。

しかしながら、ヘッダファイル中の関数や変数の宣言は構文解析や変換に影響しないことや、多くの変数型や関数型のマクロが変数や関数として扱っても構文解析上問題ない形で使用されていることから、提案言語では、問題となるヘッダファイルのインクルードをC言語へ変換するまで遅らせるコンパイル指令を用意して、この問題を回避している。

遅延指示する場合 include 指令の代わりに次のように記述する。

```
#pragma stmc lazy_include <ヘッダファイル名>
```

この記述は変換時に include 指令に変換される。

また、ヘッダファイル内で typedef 宣言による型定義が行われた場合、これを無視すると、構文解析、変換が不可能となるので、次のようにどの名前の型が定義されるべきか明示する記法を用意した。

```
#pragma stmc lazy_include <ヘッダファイル名> with_type 型名,
型名, ...
```

ad-hoc な対応であるが、この指令により、適用例で扱った限り全ての拡張構文の回避が可能であった。

4. 事例

本節では、2節で紹介した状態マシン言語およびツールを使用し、開発した適用事例を紹介する。

提案言語は元々、2008年にETロボットコンテストの開発課題をターゲットとしモデルの学習手段として作成された。2008、2010年のETロボットコンテストのライントレースロボット例題、2008-2009年のMDDロボットチャレンジの屋内用飛行船ロ

ロボット開発例題，通信を題材とした組込みシステム教育例題等で用い，それぞれ異なるターゲット(表 1 参照)に対し，提案言語とツールを適用した。

表 1 適用事例のターゲットの一覧

CPU	OS	コンパイラ	サイズ
H8/3292 (8bit)	BrickOS	gcc	約1000行
R8C/29 (16bit)	なし	NC30	約500行
M16C/26 (16bit)	なし	gcc	約500行 (ソースコードの一部として)
AT91SAM7S256 (32bit ARM7)	Toppers/ATK	gcc	200行 (ソースコードの一部として)
Intel x86 (32bit)	WindowsXP	Microsoft Visual C++	約1000行 (ソースコードの一部として)

4.1 屋内用模型飛行船システム

屋内用模型飛行船ロボットシステムの開発を課題とする MDD ロボットチャレンジにおいて，参加チームのシステム開発の一部に提案言語を用いた。

コンテストへの 2008-2009 年の参加を通して，図 4 に示す，地上設備と連携して手動，自動で航行する飛行船システムを作成した。

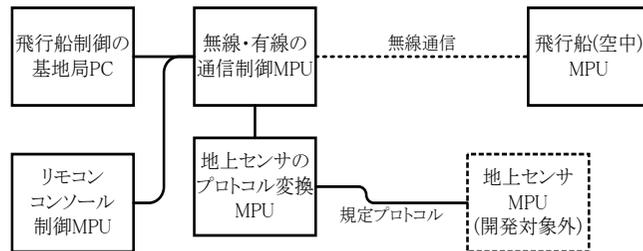


図 4 飛行船システムの構成概要

この事例の開発対象は飛行船上のサブシステムも含めると，M16C，C8051，PIC 等様々なマイコンや PC を用いそれぞれのファームウェアを複数人で分担して作成しており，総計でおよそ 10000 行程度であるが，図 4 のリモコン装置のファームウェアの一部，および基地局 PC のソフトウェアの一部を提案言語により作成した。全体に占める提案言語記述部分の割合は 15%程度(行数による)である。

リモコン装置のファームウェアは，操縦者とのやりとりの入出力，データ送受信，

一定間隔の処理，A/D 変換などのハードウェアの完了通知などを，最下層のイベントを用いて最下層の状態遷移マシンが記述されている。ハードウェアはルネサスエレクトロニクス社製 M16C/26，コンパイラは同社製 NC30 を用い，OS は用いていない。

基地局 PC は自律航行競技で飛行船の制御を行うコンピュータで，オペレータによる設定，開始，中断，再開などの操作を受け付け，地上や空中のセンサ情報を受信し，飛行船へ制御命令を送信する。インテル社 x86，32bit プロセッサ上で動く WindowsXP 向けのソフトウェアを提案言語と Visual C++を使って作成した。

4.2 LEGO マインドストームによるライトレースロボット

ET ロボットコンテストによる組込み開発課題のプラットフォームとして，教育用マイコンである LEGO マインドストームが用いられている。

LEGO マインドストームには RCX と NXT の 2 種類がありはそれぞれ ET ロボコン 2008，ET ロボコン 2010 で用い，課題であるライトレースロボットの作成に，紹介した言語を用いた。



図 5 ET ロボコンの走行体

4.2.1 RCX によるライトレースロボット

本事例の対象はセンサ数が光センサのみの 1 個と少なく，ユーザからの入力もボタン数個のみであり，全体が 1000 行程度の小規模なものである。

光センサ 1 個で外乱に影響されずに走行するために，走行前の周囲環境の計測を適切に行う必要があるため，限られたボタンで走行前の設定操作を可能とするために状態マシンモデルが記述されている。また走行中は，走行状態の推定，検知をイベントとして，状態に合わせた動作を行っている。

ルネサスエレクトロニクス社製 H8/3292，および BrickOS を用い，コンパイラとしては gcc を用いた。

4.2.2 NXT による倒立振り子ライトレースロボット

本事例は ATMELE 社製 ARM7 コア 32bit プロセッサ AT91SAM7S256 上で動く，リアルタイム OS である Toppers/ATK を対象とし，またマインドストーム NXT のハードウェアを制御するライブラリを使用している。コンパイラには gcc を用いた。

本システムでは，光センサ，モータエンコーダー，距離センサ，角度センサ等より多くのセンサを持ち，倒立振り子制御を行うなど高度化した制御が必要な課題となっている。

4.3 マイコン教育課題

東海大学 組込みソフトウェア工学科のマイコン講座で通信を扱うプログラムの例題として、500 行程度の簡単な通信プログラムを課題とした。ルネサスエレクトロニクス社の R8C/29 にボタンとシリアル通信、数種類の出力を持つハードウェアで、時間の処理中の受信や動作の中断などを行う課題を、直接状態遷移モデルによって記述し、動作させ、通信プログラムを学習することにより、通信プログラムにおける状態遷移モデルの使い方の学習を行った。

5. 議論

提案言語およびツールがいくつかのプラットフォームに問題なく適用可能であり一定の実用性を有することは、事例により確認した。

本ツールの特徴はプログラミング言語の要素と同様の形で複数の直交する状態遷移マシンをプログラム中に容易に記述できることにある。提案ツールではイベントによる遷移の処理を開数へと変換する。このことは(遷移中のアクションの完了を待つ)同期的な処理のみが可能であることを意味する。このことは(i)OS の同期機構やキュー、割り込み、変数変化など様々なイベント実現方法がある、(ii) 状態遷移マシン間の並行性があり得るため問題である。ただし(i)についてはイベント生成側での対処が可能であるため問題は少ないと考える。(ii)についてはソフトウェアによる状態遷移モデルで不可避な問題であり、解決方法を探る必要がある。

事例では複数人数でコンテスト課題開発を行ったが、ソースコード形式での記述を用いることにより、構成管理、版管理が容易であった。変更部分を行単位の差分で把握し、またモデル図によるレビューが開発の役に立った。開発メンバーの中にはプログラミング初心者も含まれていたが、課題開発を通じた状態遷移マシンの作成に関わる試行錯誤の過程が版管理システム上に残ったため、状態遷移による動作概念の獲得過程を調べる助けになるのではないかと期待している。

状態遷移モデル記法の多くは、階層化の概念を持っているが、提案言語では持っていない。しかしながら階層化が有効であるいくつかの場合に、階層がなくともプログラム形式の記法で簡潔に記述できることがあり、階層化とは異なる状態記述の構造を構築できる可能性がある。図 6 では、階層化された状態 (コンポジット状態などと呼ばれる) がモデルの簡略化に有効であるが、提案言語では以下のように複数の遷移元を指定することで、図 6 の太線で書いた遷移を、階層化を用いることなく簡潔に記述することが出来る。

```
void event1()  
  s2_1, s2_2, s2_3 -> s1  
{
```

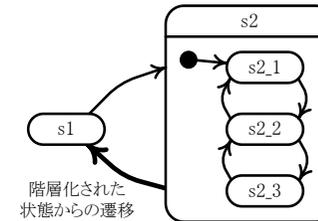


図 6 階層化の有用な例

6. おわりに

本稿では、組込みシステム開発のために開発した状態マシンに基づいたプログラミング言語について紹介し、本言語を使用した事例を紹介した。

謝辞 状態マシンプログラミング言語および事例の開発に協力いただいた、武蔵工業大学(現校名 東京都市大学) 大学院環境情報学研究科を修了した本宮茂雄氏、同大学 環境情報学部を卒業した市川清美氏、本多祐美子氏に感謝の意を表す。

参考文献

- 1) 岡山直樹, 片山徹郎: 状態遷移構文とテスト構文を導入した組込みソフトウェア向けプログラミング言語の開発, 組込みシステムシンポジウム 2010 論文集, 情報処理学会, pp. 43-48, (2010).
- 2) 藤本洋, 渡辺政彦, 福田晃: 制御設計用振舞いモデルと組込み設計用振舞いモデルの双方向変換手法, 組込みシステムシンポジウム 2010 論文集, 情報処理学会, pp. 131-136, (2010).
- 3) D. Harel: Statecharts: A visual formalism for complex systems, Science of Computer Programming 8, pp.231-pp. 274(1987).
- 4) 野村昌男, 久保秋真, 伊藤恵, 片山卓也: 組み込みシステム設計のための ObTS に基づく記述支援環境に関する研究, ソフトウェア工学研究会報告, 情報処理学会, pp91-98 (2000).
- 5) 伊藤恵, 片山卓也, オブジェクト指向方法論のための動的モデル ObTS, コンピュータソフトウェア, Vol.14, No.2, pp.22-37, Mar.(1997).
- 6) ET ロボコン(ET ソフトウェアデザインロボットコンテスト) <http://www.etrobo.jp/>
- 7) 二上貴夫, 鷲崎弘宜, 小林靖英, 乾裕紀, 大槻博之, 仲久保正人, 久保寺勇氣, 川縁幸平, 羽田千織, 三橋祐仁, 沼里京介: MDD ロボットチャレンジ 2006 開催報告, 研究報告「ソフトウェア工学」 No.2007-SE-156, pp.79-86, 情報処理学会 (2007).
- 8) 鷲崎弘宜, 久保秋真, 小林靖英, 渡辺晴美, 小倉信彦, 飯田周作: MDD チャレンジにみる組込みソフトウェアモデル中心開発の工学と教育, 研究報告「ソフトウェア工学」 No.2007-SE-156, pp.95- 102, 情報処理学会 (2007).