Vol.2012-SE-175 No.6
2012/3/15

アスペクト指向状態言語の組込みソフトウェア制約への適用

川村峰大[†] 安倍昌輝[†] 長岡 拓弥[†] 谷川 郁太[†]
原 築良[‡] 小倉 信彦[‡] 渡辺 晴美[†]

組込みソフトウェアの複雑さは，時間，メモリ，環境などの制約に起因することが多い．制約によるソフトウェアの複雑さの問題に対し，我々はアスペクト指向状態遷移言語を提案してきた．本稿では，これまで提案してきた言語を拡張し，タイムスライス毎に状態遷移モデルを割り当てることでタイミング管理を可能にする．タイミング管理を，ポートの競合，ダイナミック点灯の遅延に関する制約問題に適用する．

An Example of Aspect Oriented State Machine Language for Embedded Software Constraints

Takahiro Kawamura[†], Masaki Anbai[†], Takuya Nagaoka[†],
Ikuta Tanigawa[†], Chikura Hara[††], Nobuhiko Ogura and[††]
Harumi Watanabe[†]

Complications of embedded software development commonly arise out ofconstraint on time, memory, and environment. We have addressed the problem by an aspect oriented language with state transition syntax. In this article, we extend, to handle time constraint, the language by assigning state transition models to each time slice. We also show an example as an application to resolve port confliction and constraint on timing of LED dynamic drive.

1. はじめに

組込みソフトウェアは，時間，メモリ，環境などの制約を強く受けることで知られている．これらの制約はソフトウェアを複雑にし，理解性低下，モデル駆動開発の困難さをまねいている．制約がソフトウェアを複雑にする原因は，機能に対し横断的関心事であるためである．横断的関心事を解決する代表的な技術としてアスペクト指向技術がある．一方，組込みソフトウェアはイベント駆動型であることから，その振舞いを状態遷移モデルに基づき設計，実装することが普及している．以上から，我々は，制約による複雑さの問題解決のために，アスペクト指向技術を状態遷移モデルに適用する方法に着目し，アスペクト指向状態遷移言語を提案してきた[15].

UML へのアスペクト指向技術の適用に関する研究は多数行われている[3] [4] [5] [6] [7] [8] [9]．これらの研究はモデルが中心であるが，我々はプログラミング言語に状態遷移モデルの概念とアスペクト指向の概念を取り入れたアスペクト指向状態遷移言語を開発した．アスペクトにより，状態遷移記述を関心事に応じて合成・分解し，周期を明示したタスクで動作させることが可能である．また，多数の横断的な制約や非正常系処理の問題についても，アスペクト指向の特徴であるウィーブにより，状態遷移の合成を行うことで，制約による状態モデルの変化の追跡可能性を用意にし，個々の非正常系を，正常系と分離して扱うことができるようにしてきた．

以上の解決方法は，並行タスクに依存した方法である．プロセッサの性能向上に伴い，リアルタイム OS を利用する機会は増しているが，安価で小規模なプロセッサは数多く存在し利用する場面も多い．我々は，より実践的な言語を目指し，OS を利用しない場合においても，制約の問題を扱えるようにアスペクト指向状態遷移言語を拡張する．この拡張部分では，タイムスライス毎に状態遷移モデルを割り当てることでタイミングの管理が可能になる．

本稿では，単純な電卓を制御する組込みソフトウェアを扱う．事例ではダイナミック点灯，ポートの競合が発生するため，これらの複数の遅延に関する制約を満たさなければならない．一見，単純な電卓を制御するソフトウェアは，これらの遅延を考慮することで複雑化する．この遅延問題を通し，提案部分について述べる．

以下，2 章ではこれまで提案してきた言語，および本稿で提案する拡張部分について述べる．3 章では，電卓の事例について述べる．

† 東海大学情報通信学部
Tokai University School of Information and Telecommunication Engineering
†† 東京都市大学環境情報学部
Faculty of Environmental and Information Studies Tokyo City University

2. 投稿まで

本章ではハードウェアの制約の問題を解決するために拡張した提案言語について述べる．下記の図 1 は提案言語の概要である．状態遷移言語とは状態遷移モデルをテキスト形式で表した言語であり，状態遷移モデル 1・状態遷移モデル 2 はそれぞれある処理について状態遷移言語を使って表している．またアスペクト記述が本稿の主題となる状態遷移モデルの実行タイミングを設定するファイルである．状態遷移言語とアスペクト記述をウィーバに通すことによって，状態遷移モデルをタイムスライス化する．以降，2.1 節ではアスペクト指向状態遷移言語について説明し，2.2 節では本稿の提案内容について述べる．2.3 節では変換機について説明する．

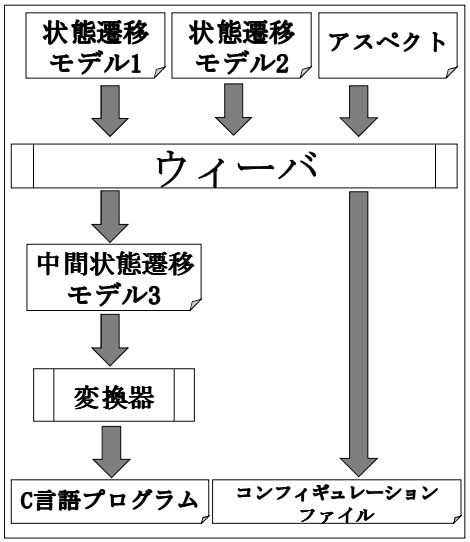

図 1　アスペクト指向状態遷移言語の構成

2.1 アスペクト指向状態遷移言語

アスペクト指向状態遷移言語は組込みソフトウェアにおいて非正常系処理による複雑化の問題や時間制約による問題を解決するために プログラミング言語に状態遷移モデルの概念とアスペクト指向の概念を取り入れた言語である．これにより状態遷移モデルを関心事に応じて合成・分解し，周期を明示したタスクで動作させることができた．[15]

2.2 提案内容

今回，我々は組込みソフトウェアにおける制約の問題を解決するために 2.1 節で述べたアスペクト指向状態遷移言語を拡張した．以降，2.2.1 節では状態遷移言語の記述方法を説明し，2.2.2 節では本稿において主題となるタイムスライスについて述べる．

図 2 に示すように，状態遷移言語はヘッダー部，状態遷移モデル部，関数・タスク部の 3 つの構成からなる．以下，各部を説明する．

2.2.1 状態遷移言語の記述方法

ヘッダー部

ヘッダー部を図 2 内の(a)に示す. ここには C 言語で記述する際に必要となる#include や#define などのプリプロセッサディレクティブを記述することができる．

状態遷移モデル部

状態遷移モデル部を図 2 内の(b)に示す．状態遷移モデルの宣言は「stm_def{} 状態遷移モデル名 1, 状態遷移モデル名 2, ... ; 」である．この際,”{”，“}”で囲まれた範囲に，状態定義，イベント定義，遷移定義を記述する．

(1)状態定義

状態定義を図 2 内の(c)に示す．状態定義は「状態名={ is_initial か is_final, "状態説明", マーク名}; 」である．is_initial と is_final は，それぞれ開始状態と終了状態を表している．状態名は状態遷移言語内で扱うための名前で，状態説明は自然言語による理解度向上のための名前である．マーク名には後述するアスペクトで宣言されたマーク名を記述し，カンマで区切って複数個指定できる．必要ない場合には省略することが可能である．マークについては 4.2.3 節にも記述してあるのでそこも参照する　.

(2) イベント定義

イベント定義を図 2 内の(d)に示す．イベント定義は「戻り型イベント名 (パラメータリスト) "イベント説明"」である．イベントではユーザの必要に応じて，戻り型とパラメータリストが指定できる．イベント名は状態遷移言語内で扱うための名前で，イベント説明は自然言語による理解度向上のための名前である．

(3)遷移定義

遷移定義を図 2 内の(e)に示す．遷移定義は「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 } 」である．ここでは状態・イベント・アクションの対応関係を記述する．イベント名には上記で定義したイベント指定し，ガード条件には C 言語を受理する形で条件式を記述する．状態の遷移は「->」を使って記し，アクション動作には状態が遷移した際に行う処理を記述する．ガード条件説明とアクション説明はそれぞれ自然言語による理解度向上のための名前である．

関数・タスク部

関数・タスク部を図 2 内の(f)に示す．関数・タスク部では通常の C 言語での処理をユーザが自由に記述できる．その中で，記述内の任意の場所でイベント名を「状態遷移モデル.イベント名() 」の形で記述することによって，イベント発行が可能である．

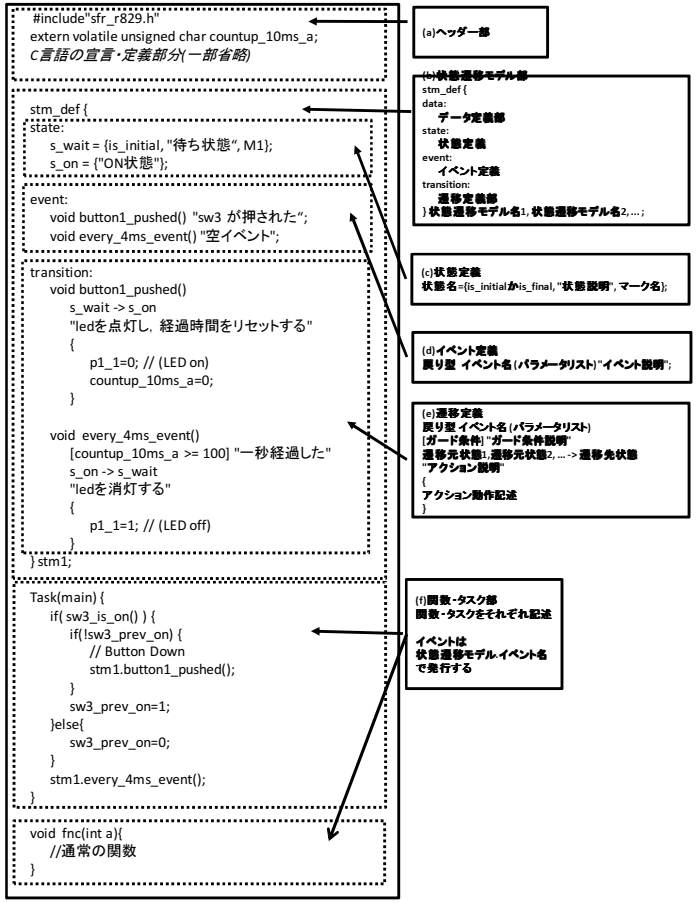

図 2　状態遷移言語の記述方法

2.2.2　タイムスライス

下記の図 3 は拡張前のアスペクト記述の例である．今回の提案では図 3 のような記述に加えて，状態遷移モデルのタイムスライスできるように拡張した．タイミング宣言部を図 4 の(d)に示す．タイミング宣言部とはどのような周期で管理したいかという情報を決めるためのタイムスライスを定義する箇所であり，宣言は「 Timing_def{ } 」であり，“{”, “}”で囲まれた範囲内に各タイムスライスの定義を行う．図 4 の(e)には各タイムスライスの定義を示す．タイムスライス定義は「 Time_Slice タイムスライス名{ } 」である．“{”, “}”で囲まれた範囲内には周期を決める PERIOD を指定する．

それら定義したタイムスライスにどの状態遷移モデルを割り当てるかを図 4 の(c)に示す. 割り当ての記述方法は「 assign_timing(状態遷移モデル名，タイムスライス名); 」である．状態遷移モデル名には既に状態遷移言語内で宣言されたものだけ指定可能である．

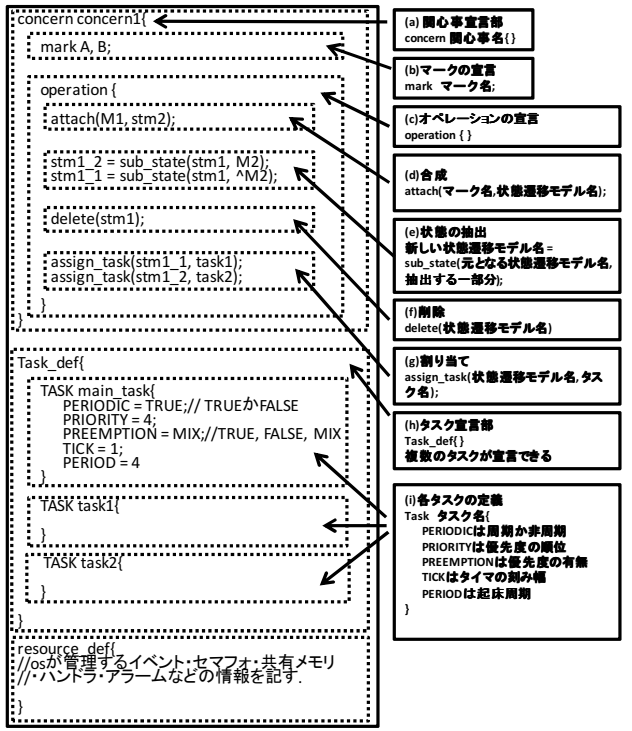

図 3　拡張前のアスペクト記述の例

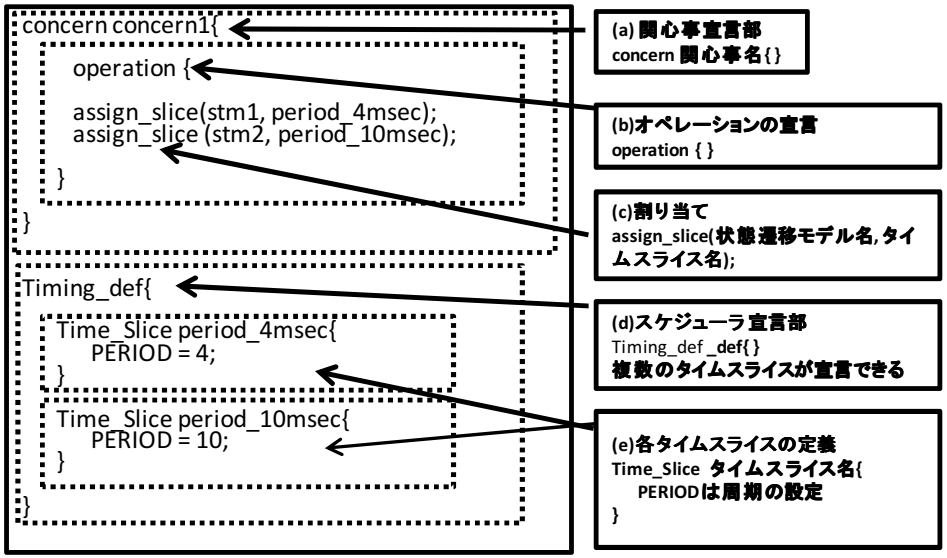

図 4　拡張後のアスペクト記述の例

2.3 変換機の動作

変換器は状態遷移言語で記述されたプログラムを C 言語のプログラムに変換する．

3. 適用

本章では組込みソフトウェアにおいて起こる制約の問題を 2 章で提案したタイムスライスに適用する．今回適用する制約の問題の例はマイコンを使用して作成した簡単な電卓アプリケーションに適用する．以降，3.1 節では電卓アプリケーションにおける制約の問題点をあげ，3.2 節ではその問題にタイムスライスを適用する

3.1 電卓アプリケーションにおける制約の問題

今回, 適用する電卓アプリケーションのハードウェアは数字ボタンが’0’から’9’まであり，演算子ボタンが’+’と’-’，’=’ボタン，’CL’ボタンの計 14 個のボタンと 4 桁の 7segLED という構成になっている．その構成を現したイメージを図 5 に示す．また数字ボタンと 4 桁の 7segLED との配線図を図 6 に示す．図 6 に示すようにボタンのポートと 7segLED のポートが競合している．そのため入力と出力が同時に行うことができないという制約の問題が発生する．この問題を解決するためには周期タイマによる処理の切り替えが必要となる．

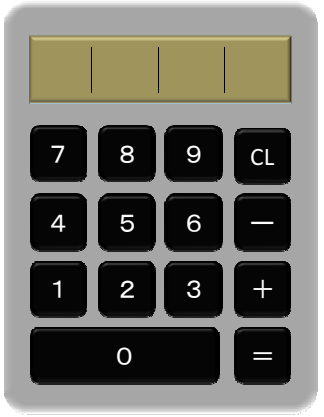

図 5　電卓の構成イメージ

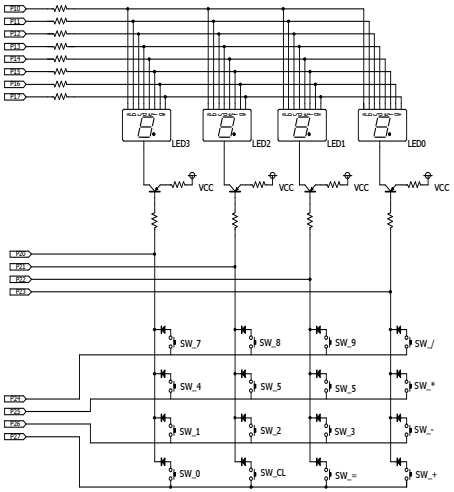

図 6　ボタンと 7segLED との配線図

3.2 問題への適用

3.1 節で述べた問題にタイムスライスを使用して，周期タイマによる処理に切り替えを行う．入力の処理を表した状態遷移モデルを図 7 に，出力の処理を表した状態遷移モデルを図 8 に示す．なお図 8 の状態遷移モデルの状態遷移言語を図 9 に示す．この二つの処理を周期タイマによって，切り替えを行うためにアスペクト記述を記述する．その記述を図 10 に示す．ウィーバにかけることによって，タイミング宣言部で宣言した周期にそれぞれ 4[msec]の周期には入力の処理を，10[msec]の周期には出力の処理をウィーブする．ウィーブ結果を図 11 に示す．図 11 の状態遷移言語をタイミングチャートで現したのが図 12 である．この通り，タイムスライスによって 2 つの処理を各周期に切り替えることができた．

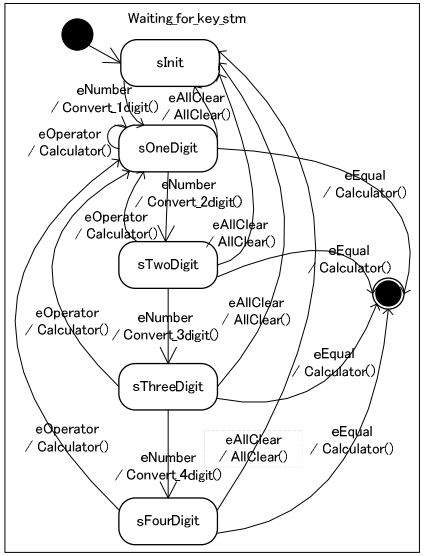

図 7　入力の状態遷移モデル

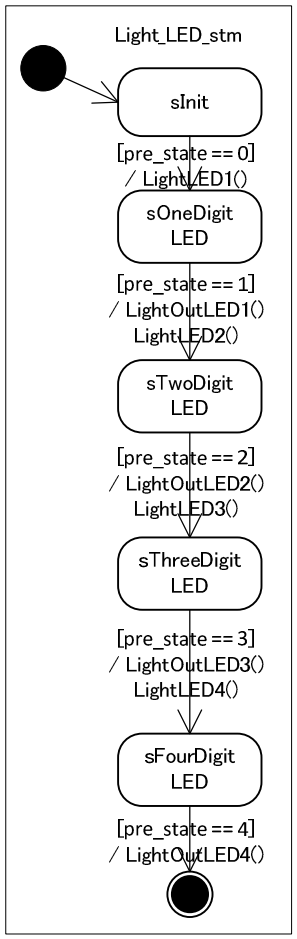

図 8　出力の状態遷移モデル

```
//グローバル宣言
void LightLED1(void);
void LightLED2(void);
void LightLED3(void);
void LightLED4(void);
void LightOutLED1(void);
void LightOutLED2(void);
void LightOutLED3(void);
void LightOutLED4(void);

stm_def
{
state:
 sInit = { is_initial, "0セグ表示" };
 sOneDigitLED = { "1桁目のLED" };
 sTwoDigitLED = { "2桁目のLED" };
 sThreeDigitLED = { "3桁のLED目" };
 sFourDigitLED = { "4桁目のLED" };
 sFinal = { is_final, "終了状態" };
event:
 void eNullEvent() "空イベント";

transition:
//初期
 void eNullEvent()
 [pre_state == 0]
 sInit -> sOneDigitLED
 "空イベント"
     {
        LightLED1();
     }
//1行目LED
 void eNullEvent()
 [pre_state == 1]
 sOneDigitLED -> sTwoDigitLED
 "空イベント"
     {
        LightOutLED1();
        LightLED2();
     }
//2行目LED
 void eNullEvent()
 [pre_state == 2]
 sTwoDigitLED -> sThreeDigitLED
 "空イベント"
     {
        LightOutLED2();
        LightLED3();
     }
//3行目LED
 void eNullEvent()
 [pre_state == 3]
 sThreeDigitLED -> sFourDigitLED
 "空イベント"
     {
        LightOutLED3();
        LightLED4();
     }
```

```
//4行目LED
 void eNullEvent()
 [pre_state == 4]
 sFourDigitLED -> sFinal
 "空イベント"
     {
        LightOutLED4();
     }

}Light_LED_stm;

void main(void){
 Light_LED_state.eNullEvent();
}
```

図 9　出力の状態遷移言語

```
concern concern1{

    operation{
        assign_slice(Waiting_for_key_stm.stmc, period_4msec);
        assign_slice (Light_LED_stm.stmc, period_10msec);
    }

}

Timing_def{
    Time_Slice period_4msec{
        PERIOD = 4;
    }

    Time_Slice period_10msec{
        PERIOD = 10;
    }
}
```

図 10　切り替えを行うためのアスペクト記述

Vol.2012-SE-175 No.6
2012/3/15

```
#pragma interrupt_handler Timer16_1_ISR
int timer_count = 0;

//stm1
typedef enum {
  NONE = 0,          // ボタンが押されていない
  NUMBER  = 1,       // 数値ボタン
  OPERATOR  = 2,     // 演算子ボタン
  EQUL = 3, // イコールボタン
  CLEAR = 4 // クリアボタン
} BUTTON_PATTERN;
void Convert_1digit(void);
void Convert_2digit(void);
void Convert_3digit(void);
void Convert_4digit(void);
void Calculator(void);
void AllClear(void);
BUTTON_PATTERN Check_Button(void);

stm_def
{
state:
  sInit = { is_initial, "初期状態" };
  sOneDigit = { "1桁目" };
  sTwoDigit = { "2桁目" };
  sThreeDigit = { "3桁目" };
  sFourDigit = { "4桁目" };
  sFinal = { is_final, "終了状態" };
event:
  void eNumber() "数値ボタン入力";
  void eOperator() "演算子ボタン入力";
  void eEqual() "Equalボタン入力";
  void eAllClear() "Clearボタン入力";
transition:
  void eNumber()
     sInit -> sOneDigit
     "数値ボタン入力"
     {
        Convert_1digit();
     }

     //以下省略

}Waiting_for_key_state;
```

```
//stm2
void LightLED1(void);
void LightLED2(void);
void LightLED3(void);
void LightLED4(void);
void LightOutLED1(void);
void LightOutLED2(void);
void LightOutLED3(void);
void LightOutLED4(void);
stm_def
{
state:
  sInit = { is_initial, "0セグ表示" };
  sOneDigitLED = { "1桁目のLED" };
  sTwoDigitLED = { "2桁目のLED" };
  sThreeDigitLED = { "3桁のLED目" };
  sFourDigitLED = { "4桁目のLED" };
  sFinal = { is_final, "終了状態" };
event:
  void eNullEvent() "空イベント";

transition:
//初期
  void eNullEvent()
      [(Tcount+10) <= Tcount] "10msec経過した"
      sInit -> sOneDigitLED
      空イベント"
      {
        LightLED1();
      }

      //以下省略
}Light_LED_state;
```

```
void Timer16_1_ISR(void){
    timer_count++;
}

void main(void){
    BUTTON_PATTERN button_pattern = NONE;
        while(1){
            if(timer_count == 4){
                button_pattern = Check_Button();
                if(button_pattern == NUMBER){
                    Waiting_for_key_state.eNumber();
                }else if(button_pattern == OPERATOR){
                    Waiting_for_key_state.eOperator();
                }else if(button_pattern == EQUAL){
                    Waiting_for_key_state.eEqual();
                }else if(button_pattern == CLEAR){
                    Waiting_for_key_state.eAllClear();
                }
            }else if(timer_count == 10){
                Light_LED_state.eNullEvent();
            }
        }
}
```

図 11　ウィーブ後の状態遷移言語

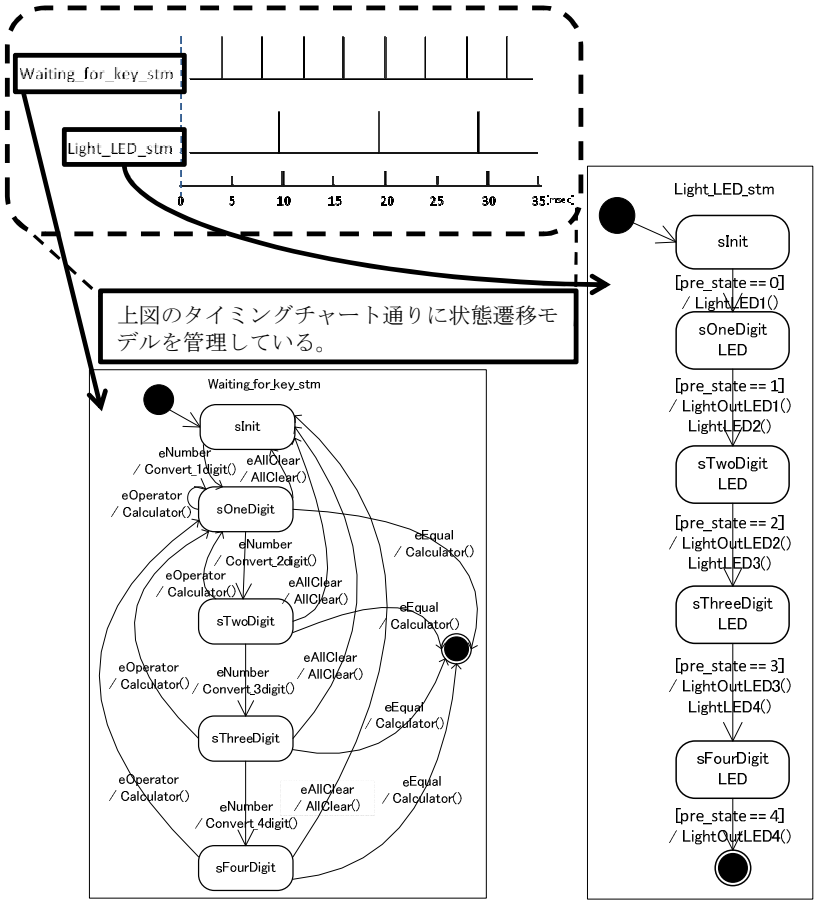

図 12　タイムスライスに基づく状態遷移モデルの切り替え

4. おわりに

本稿では，制約に起因した組込みソフトウェアの複雑化の問題に対し，これまで提案してきたアスペクト指向状態遷移言語を拡張した．本拡張では，リアルタイム OS やカーネルを搭載していないプロセッサにおいても，タイムスライス毎に状態遷移モデルを割り当てることでタイミング管理を用意にした．また，タイミング管理を，ポートの競合，ダイナミック点灯の遅延に関する制約問題を有する電卓の問題に適用し．今後の研究では,より多くの実際的な事例に適用し,評価を行い,機能を充実させたい．

参考文献

1) 佐々木心也，片山徹郎：組込みシステムの性能に応じた UML の変形について，組込みシステムシンポジウム 2005 論文集，情報処理学会 pp.140-pp.143, (2005).
2) 紫合治: 組込みソフトウェアにおける非正常処理の抽出，組込みシステムシンポジウム 2005 論文集，情報処理学会 pp.140-pp.143, (2005).
3) 鵜林尚靖，佐野慎治，前野雄作，村上聡，片峯恵一，橋本正明，玉井哲雄:アスペクト指向に基づく拡張可能な MDA モデルコンパイラ， 組込みシステムシンポジウム 2004 論文集，情報処理学会 pp.104 - pp.107(2004).
4) T. Elrad, O. Aldawud, A. Bader: Expressing Aspects Using UML Behavioral and Structual Diagrams, Aspect-Oriented Software Development, Addison Wesley, pp.459- pp.478(2005).
5) J. Zhang, T. Cottenier, A. van den Berg, J. Gray: Aspect Composition in the MotorolaAspect - Oriented Modeling Weaver, Vol. 6, No. 7, Journal of Object Technology, pp.89– pp.108 (2007).
6) N. Noda, T. Kishi: Aspect-Oriented Modeling for Embedded Software Design, 14th Asia-Pacific Software Engineering Conference (APSEC'07), pp.342– pp.349(2007).
7) L. Fuentes, P. Sánchez: Dynamic Weaving of Aspect-Oriented Excecutable UML Model, Transactions on Aspect-Oriented Software Development VI, pp.1 – pp.38(2009).
8) J. Whittle, P. Jayarman, A. Elkhodary, A. Moreia, João Araújo: MATA: A Unified Approach for Composing UML Aspect Model Based on Graph Transformation, Transactions on Aspect-Oriented Software Development VI, pp. 191-pp.237(2009).
9) A. Carton, C. Driver, A. Jackson, S. Clarke: Model-Drien Theme/UML, Transactions on Aspect-Oriented Software Development VI,
pp. 238-pp.266(2009).
10) M. R. Sleep, M. J. Plasmeijer, M. C. J. D van Eekelen: Term Graph Rewriting Theory and Practice, WILEY(1993).
11) 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-pp.48 (2010).
12) 小倉信彦，谷川郁太，渡辺晴美: 状態遷移言語による組込みソフトウェア開発，情報処理学会研究報告, Vol.2011-SLDM-149 No.48, pp.1 – pp.6(2011).
13) 満田成紀, MDD ロボットチャレンジ 2009 ワークショップ開催案内:審査員他によるワークショップ，組込みシステムシンポジウム 2009 論文集，情報処理学会, pp. 211-pp.213, (2009).
14) ET ソフトウェアデザインロボットコンテスト(ET ロボコン)http://www.etrobo.jp/
15) 安倍 昌輝，川村 峰大，長岡 拓弥，谷川 郁太，原 築良，小倉 信彦，渡辺 晴美，組込みソフトウェアのためのアスペクト指向による状態遷移言語の提案， 組込みシステムシンポジウム 2011 論文集，情報処理学会, 21-1 - 21-10, (2011).