

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

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モデルは、システムの本来の役割を表す正常系の処理に加え、膨大な非正常系の処理が表されるため複雑になることが多い。実装に近い下流のモデルは、組込みソフトウェア特有の制約により、さらに複雑になる。制約は多くの場合、横断的関心事となることから、このような問題を解決するために、アスペクト指向技術が期待されている。本論文では、アスペクト指向状態遷移言語を提案する。本言語は、組込みソフトウェア特有の制約を状態遷移モデル上でアスペクトとして扱うことが可能で、状態遷移モデルを周期タスクへ割り当てる機能を有する。また、ETソフトウェアロボットコンテストのNXTロボットに適用し、その効果を評価する。

A State Transition Machine Language Based on Aspect-Oriented Concept for Embedded Software

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

Models of Embedded software are generally complicated, by many kind of irregular behavior. The complexity makes it difficult to understand the regular behavior which is essential role of the system. Additionally, many constraints of embedded software make more complicated the model. As some constraints often become crosscutting concerns, aspect oriented technologies are expected to solve these problems. In this paper, we propose an aspect oriented state transition machine language, which provides composition and decomposition of state transition models and also has a periodic task assignment mechanism to cope with the timing constraint. For evaluation of our work, we will apply the language to NXT robots software for ET Software Design Robot Contest.

1. はじめに

組込みソフトウェアのモデルは、システムの本来の役割を表す正常系の処理に対し、膨大な非正常系の処理が付随するため、正常系部分の理解を損なうほど複雑であることが多い。さらに、実装に近い下流のモデルは、組込みソフトウェア特有の制約により、さらに複雑になる。従って、本来の役割を表す正常系部分の理解が困難になり、上流モデルと下流モデルや実装とが関連付かなくなるといふ追跡可能性の問題をまねくこともある。このような状況は、機能追加を困難にし、誤りの修正に時間を要する問題を引き起こす。以上の非正常系処理による複雑化、制約による追跡可能性の低下の問題解決には、アスペクト指向技術の適用が有効である。非

正常系処理や制約は、しばしば横断的関心事であり、時には、システム全体に分散波及することさえあるためである。

組込みソフトウェアの振舞いを記述する際には、状態遷移モデルが用いられることが多いことから、我々は、アスペクト指向技術の状態遷移モデルに適用する方法に着目した。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。組込みソフトウェアの制約に関する問題は、実装と密接に関係する。モデルを中心に開発していくモデル駆動開発では、実装情報を加味するためには、プラットフォーム定義モデル(PDM)が必要となる。小規模で周辺機器が多様であるシステムにとっては、PDM の構築は現実的でない開発負荷となる。また、モデル駆動開発のためのツールの使用方法や、アクションの記述、モデル変換のため

† 東海大学情報通信学部

‡ 東京都市大学環境情報学部

の言語の習得も必要となる。

一方で、モデル駆動開発ツールは使用せず、状態遷移に基づいた設計実装を行う、または状態遷移モデルと実装が対応付く部分のみにツールを使用するというのが、広く行われている。このようなツールを使用しない、あるいは限定的な使用では、非正常系や制約によるモデルや実装の変更を場当たりのに行うため、冒頭で述べた、複雑化、追跡可能性の低下をまねく。

以上から、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提案する。本言語では、アスペクトにより、状態遷移記述に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。状態遷移モデルとプログラムは容易に対応付き、制約をアスペクトとして表現可能なため、制約に応じて、場当たりのな変更、それに伴う追跡可能性低下の問題を削減できる。すなわち、従来、状態遷移設計後の実装時に、タイミングの問題が生じたり、メモリの問題が生じた際に、タスクを分解・合成したり、遅延時間に合わせてコードを追加する等の処理を、状態遷移モデルを変更するアスペクトとして記述することができる。

また、多数の横断的な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、個々の非正常系を、正常系と分離して扱うことが可能となる。

以下、本論文の構成は次の通りである。まず 2 章ではアスペクトが必要となる組込みシステムの問題について、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 2010)の難所を例に、説明する。3 章ではアスペクト指向の概念に基づいた状態遷移言語を提案する。4 章では、3 章で提案した方法を用い 2 章の問題が解決できることを示す。5 章では関連研究と本研究との比較を行う。

2. 背景

本章では、組込みソフトウェア開発で起こりうる横断的関心事の問題を ET ロボコン 2010 で用いる NXT ロボットを例に述べる。

以下、2.1 節では ET ロボコンについて概説し、2.2 節ではアスペクト指向技術の代表的な例であるロギングの問題について述べる。本事例は、一般的なアスペクト指向の概念が実現可能であることを後章で確認することを目的としている。2.3 節では、非正常系処理として脱線処理の問題について記す。本事例により、本研究の目的の一つである非正常系処理によるモデルの複雑化に

ついて明らかにする。2.4 節では本論文の特徴である時間制約に関する事例として障害物検知について論じる。

2.1 NXT ロボットの概要

ET ロボコン 2010 は、ロボットを図 1(a)に示すコースの黒線上で走らせ周回タイムを競う大会である[14]。NXT ロボットの構成は図 1(b)に示すとおり、黒線を識別する光センサ、倒立を保つジャイロセンサ、障害物を検知する超音波センサからなる。

図 1(c)をもとに、以下の章で事例として紹介する難所について述べる。(a)に示すとおり、コース上には難所が複数存在する。(c)の難所では、障害物であるペットボトルの配置パターンを検知し、(a)右側の二連ループ部分を通る。



(c). ET ロボコン2010 超音波センサを使用する難所

1. NXT ロボットは黒線上を走行し、灰色のラインを検知後、左旋回しボールの検知を開始(A,B,Cの地点にボールが配置される)
2. ボールの有無を検知後右旋回し黒線上の走行を始める。
3. この一連の動作をB,C地点でも行う。

図 1 ET ロボコンコース、一部機能

2.2 横断的関心事の問題:ロギング

アスペクト指向技術で解決できる代表的な問題にロギングがある。この問題は、すべてのクラスにロギングの処理が横断することにある。横断する処理を各クラスに加えると、修正工数が増加し、変更誤りを招く恐れがある。ET ロボコンの事例においてもロギングの問題が生じる。図 2 は ET ロボコンにおける主要な機能を抽出したク

ラス図である。このクラス図に、テストやチューニングのため、それぞれロギングの処理を加えて、各種センサの値を獲得する必要がある。しかし、図 2 で示す通り、ロギングの処理は各クラスに分散するので、ET ロボコンにおいてもロギングの問題が発生する。なおクラス図を用いて記しているが、実装ではクラス概念をファイルに対応づけて C 言語で行っている。

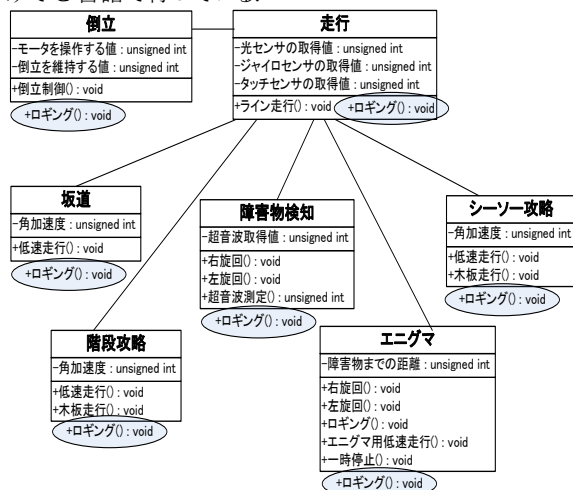


図 2 各機能にロギングが横断しているクラス図

2.3 非正常系処理問題: 脱線処理

1 章で述べた通り、組込みソフトウェアは多数の非正常系を処理するため、システムが複雑になることが多い。ET ロボコンの事例でも同様な問題が発生する。ET ロボコンにおいて、非正常系の処理には脱線や光センサのノイズ、エンコーダの誤差などがある。その中で本論文は正常系である通常走行処理に、非正常系の脱線処理が加わる問題に着目する。

図 3 は正常系の通常走行を表す状態遷移モデルである。これはエッジ走行を表して、NXT ロボットがコース上の黒なら右に、白なら左に曲がる。図 4 は、この図 3 に非正常系の脱線処理を表す状態遷移モデルを加えた図である。脱線処理はまず脱線したのが判定されたらバックする。その後 NXT ロボットは旋回し、前進する。このときにコース上に復帰しない場合は、再度バックから始めて、コースに復帰するまでこの動作を繰り返す。図 4 に示すとおり、このような処理が 1 つ、通常走行に加わるだけで状態遷移モデルは複雑になる。またこの図 3 に加える非正常系の処理は脱線処理以外にもあり、状態遷移モデルがさらに複雑になることは容易に予想される。さらに、脱線処理は通常走行だけでなく、図 1 に記した二連ループにも表れる横断的関心事でもある。

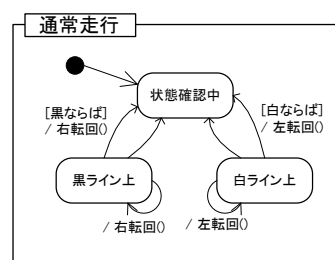


図 3 正常系の通常走行における状態遷移モデル

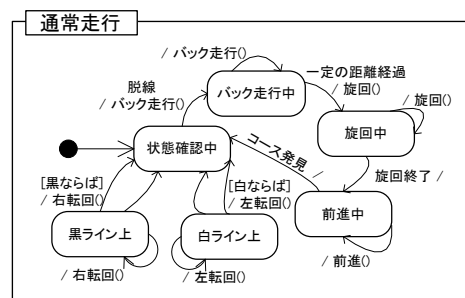


図 4 正常系の通常走行に非正常系を追加した状態遷移モデル

2.4 時間制約問題: 障害物検知

時間制約は組込みソフトウェアにおける特徴的な問題であるが、NXT ロボットにおいても各センサの時間制約を受ける。図 5 は図 1(c)で示したペットボトルの障害物を超音波センサで検知する機能の状態遷移モデルであり、NXT ロボットは倒立しながら黒線上を走行し灰色を検知したら 90° 左旋回し超音波センサで障害物を検知し 90° 右旋回して再度黒線上を走行する。組込みソフトウェア開発では、システムに追加するハードウェアを決定した際に、特性を測りモデルの設計を行う。NXT ロボットに搭載する超音波センサの特性を計測すると、50[msec]の待ち時間を要する。NXT ロボットではどの状態でも倒立制御を 4[msec]間隔で行う必要がある。図 5 では障害物検知中状態時に超音波センサによる 50[msec] の遅延が生じ、倒立を保つことができず倒れてしまう。

この時間制約により状態遷移モデルにも変更が生じる。NXT ロボットを正常に動作するための、状態遷移モデルの実現方法を二つ取り上げる。

(1) 並行タスクを扱えるならば、超音波センサの障害物検知時の状態を新たに作成した 50[msec]のタスクに対応づけ並行に動作させることで解決する。

(2) 並行タスクを利用しない場合、プログラムの処理間隔に 4[msec]毎に倒立制御のプログラムを入れ込むことでも解決できる。

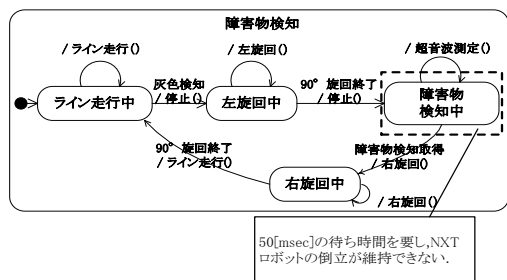


図5 障害物検知の状態遷移モデル

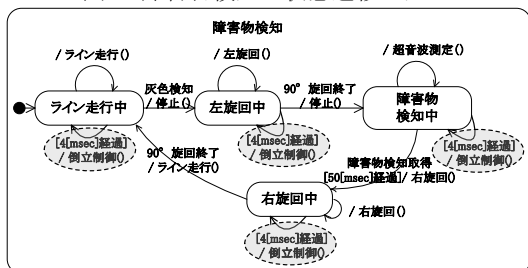


図6 倒立制御仕様を付加した状態遷移モデル

3. 提案

本章では2章の問題を解決するためにアスペクト指向状態遷移言語を提案する。提案言語では、アスペクト記述により、状態遷移モデルに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る。以降、3.1節では提案言語の概要を、3.2節では状態遷移言語の記述方法を説明する。3.3節は、本論文の主題である状態遷移モデルの合成・分解を可能にする部分の記述に関する説明である。3.4節では、提案言語のウィーバで可能な状態遷移モデルの合成・分解について例を用いて説明する。3.5節では変換器によっての状態遷移言語からC言語への変換規則を説明する。

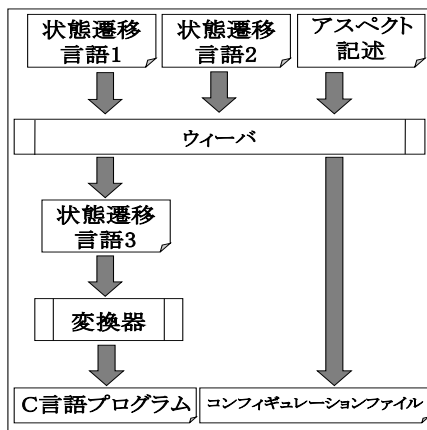


図7 アスペクト指向状態遷移言語の構成

3.1 提案言語の概要

上記の図7はアスペクト指向状態遷移言語の概要である。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、状態遷移言語1・状態遷移言語2はそれぞれある状態遷移モデルを表している。またアスペクト記述とは複数の関心事とタスクの設定をまとめたものである。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、元の状態遷移モデルを合成・分解した新たな状態遷移モデルを表す状態遷移言語3を作成する。それと同時にタスクの各設定が書かれたコンフィギュレーションファイルも生成する。状態遷移言語3を変換器に通すとその状態遷移モデルを表すC言語が生成される。

3.2 状態言語の記述方法

図8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、タスク・関数部の3つの構成からなる。以下、各部を説明する。

3.2.1 ヘッダー部

ヘッダー部を図8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。

3.2.2 状態遷移モデル部

状態遷移モデル部を図8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8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あり、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節にも記述しているのでそちらも参照する。

(2) イベント定義

イベント定義を図8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ーの必要に応じて、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 8 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。

3.2.3 関数・タスク部

関数・タスク部を図 8 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ー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。

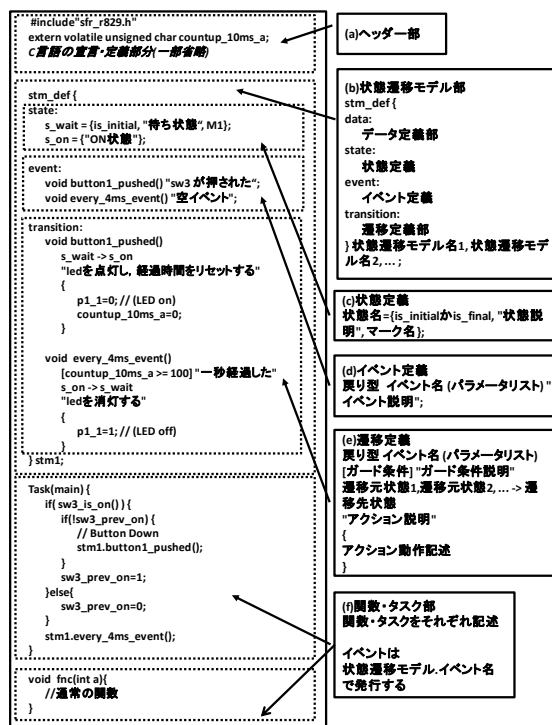


図 8 状態遷移言語の記述方法

3.3 アスペクト記述の方法

アスペクト記述の方法を図 9 に示す。アスペクトは複数の関心事とタスクの設定をまとめたファイルであり、関心事宣言部とタスク宣言部の二つから成る。

3.3.1 関心事宣言部

関心事宣言部を図 9 の(a)に示す。関心事宣言部は「concern 関心事名{ }」であり、“{”, “}” で囲まれた範囲はマークとオペレーションの 2 つの宣言を記述する。

(1) マークの宣言

図 9 の(b)にマークの宣言を示す。マークとは状態に印をするための記述であり、複数の状態を 1 つの集合とすることができる。これは、アスペクト内で使用するのここに宣言をする。マークの宣言は「mark マーク名 1, マーク名 2, ...;」である。

(2) オペレーションの宣言

図 9 の(c)にオペレーションの宣言を示す。オペレーションの宣言は「operation { }」である。“{”, “}”で囲まれた範囲内には関心事に対して、合成・状態の抽出・削除・割り当ての4つの操作が可能である。以下それぞれの操作についての意味と記述方法を説明する。

・**合成**: 図 9 の(d)に合成を示す。合成とは指定した複数の状態に対して、別の状態を張り付ける操作とする。記述方法は「attach(マーク名,状態遷移モデル名);」である。マークで指定した複数の状態に状態遷移モデルがそれぞれ加わる。

・**状態遷移モデルの抽出**: 図 9 の(e)に状態の抽出を示す。状態の抽出とは指定した状態遷移モデルから一部分を抽出して、新しい状態遷移モデルとして宣言する操作である。記述方法は「新しい状態遷移モデル名 = sub_state(元となる状態遷移モデル名, 抽出する一部分);」である。新しい状態遷移モデル名は任意の文字列を指定でき、元となる状態遷移モデル名はすでに宣言してある状態遷移モデル名のみである。抽出する一部分はマークが指定でき、マークの前に^を置くことによってマーク以外の状態を指定する。

・**削除**: 図 9 の(f)に削除を示す。削除とは指定した状態遷移モデルを削除する操作である。記述方法は「delete(状態遷移モデル名);」である。指定できる状態遷移モデル名はすでに宣言されているものだけである。

・**割り当て**: 図 9 の(g)に割り当てを示す。割り当てとは指定した状態遷移モデルを新しいタスク上に割り当て

ることである。記述方法は「`assign_task`(状態遷移モデル名, タスク名);」である。状態遷移モデル名には既に宣言されたものだけ指定可能である。タスクには新しいタスクの名前を指定する。

3.3.2 タスク宣言部

図 9 の(h)にタスク宣言部を示す。タスク宣言部は「`Task_def{ }`」であり, “{”, “}”で囲まれた範囲内には各タスクを定義する。

(1)タスク定義

図 9 の(i) にタスク定義を示す。タスク定義は「`Task スク名{ }`」である。“{”, “}”で囲まれた範囲内に `PERIODIC` は周期か非周期を, `PRIORITY` は優先度の順位を, `PREEMPTION` は優先度の有無を, `TICK` はタイマの刻み幅を, `PERIOD` 起床周期をそれぞれ指定する。

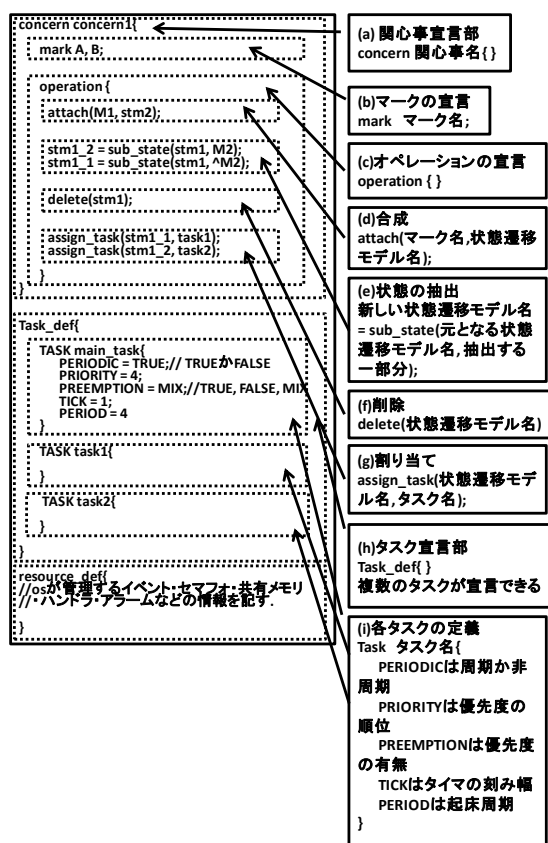


図 9 アスペクト記述の例

3.4 ウィーバによる合成・分解

ウィーバではオペレーションで指定した操作に従っ

て, 状態遷移言語で記述された状態遷移モデルを合成・分解する。

図 10 はアスペクト記述の各操作の結果がどのようになされるか状態遷移モデルを用いて, 説明したものである。状態遷移モデルでマークを使用するために「状態名 : マーク名」と記述することで状態遷移モデルにマークを導入する。

図 10 の(1)は `attach` 操作によって指定されたマーク名 `M1` に状態遷移モデル `stm2` を合成している。その結果, 状態を 4 つ持つ状態遷移モデル `stm1` になる。図 10 の(2)では `sub_state` 操作によって, マーク名 `M2` が印された状態を `stm1_2`, そうでない状態を `stm1_1` としてそれぞれ状態遷移モデルに分解する。その後, `delete` 操作に従って, 状態遷移モデル `stm1` を削除している。図 10 の(3)は `assign_task` 操作によって, 周期を明示した `task1` に `tm1_1` の状態遷移モデルを割り当て, `task2` に `tm1_2` の状態遷移モデルを割り当てる。

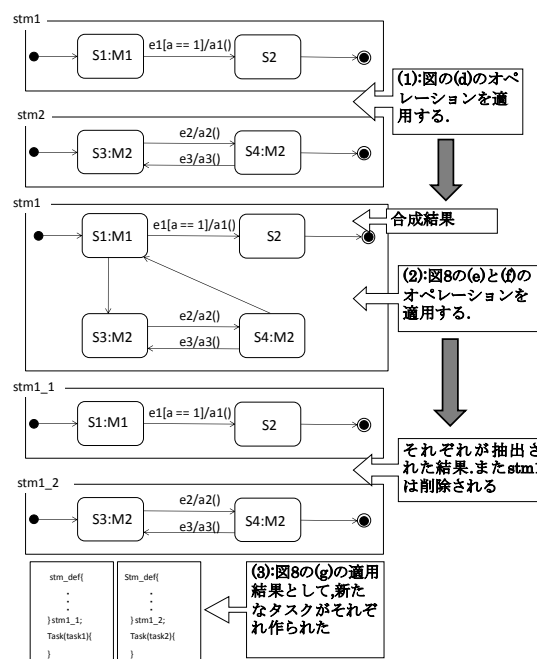


図 10 ウィーバ適用例

3.5 変換器の動作

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。図 10 には変換器の適用例を示す。この図 11 は上記の図 7 で示した状態遷移言語を変換器に通した場合の結果である。

```
#include"sfr_r829.h"
volatile unsigned char countup_10ms_a;
//C言語の宣言・定義部分(一部省略)

// definition of stm1
struct tag_stm1{
    int state;
} stm1;

void stm1__init(struct tag_stm1 *p_stm){
    p_stm->state=0;
}

void stm1__button1_pushed(struct tag_stm1 *p_stm){
    switch(p_stm->state){
    case 0:
        p1_1=0;
        countup_10ms_a=0;
        p_stm->state=1;
        break;
    default:
        // default case should be trapped !
    }
}

void stm1__every_4ms_event(struct tag_stm1 *p_stm){
    switch(p_stm->state){
    case 1:
        if(countup_10ms_a >= 100)
        {
            p1_1=1;
            p_stm->state=0;
        }
        // else should be trapped!
        break;
    default:
        // default case should be trapped !
    }
}

Task(main){
    sw3_prev_on=0;
    char c;
    if(SW3_IS_ON){
        if(!sw3_prev_on){
            stm1__button1_pushed(&stm1);
        }
        sw3_prev_on=1;
    } else {
        sw3_prev_on=0;
    }
    stm1__every_4ms_event(&stm1);
}

void fnc(int a){
    //通常の関数
}
```

図 11 状態遷移言語を変換器に適用した後の C 言語

4. 適用例と評価

本章では 2 章で述べた組込みソフトウェア開発で起こりうるロギング、時間制約に関する関心事に 3 章で提案したアスペクト指向状態言語を適用し、評価する。

4.1 ロギングの問題に状態言語を適用

2.2 節では、ロギングの処理がすべてのクラスに横断する問題である。そこでロギングの処理を関心事として、アスペクト指向状態遷移言語を使って、各クラスにロギングの処理を合成する。図 12 に示したプログラムは各クラスの状態遷移言語に記されている LIGHT, GYRO, SONAR マークにロギングの処理を追加するためのアスペクト記述の一部である。このプログラムと各クラスを表した状態遷移言語をウィーバに通して、各状態遷移モデルにロギングの処理を合成する。その結果、得られた状態遷移モデルを図 13 に示す。図 13 に示す通り、分散していたロギングの処理をアスペクトでまとめることができた。これにより一括でロギングの処理が合成できるので修正工数が減少し、また自動なので変更誤りを招く恐れが低下した。なおここで示した図 13 は実際にウィーバによって作成される状態遷移モデルを元にクラス図に書き直した図である。尚、アスペクトによって変更を施す各状態へマークを付随させるのではなく、状態宣言時に状態の種類特徴を表す印として付ける。

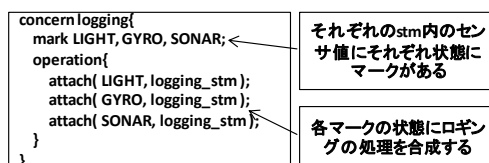


図 12 ロギングのアスペクト記述の一部

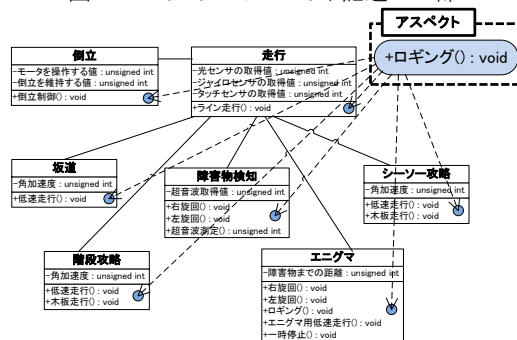


図 13 ロギング処理を合成した後のクラス図

4.2 脱線処理の問題に提案言語を適用

2.3 節では非正常系の脱線処理を正常系の通常走行に加えると状態遷移モデルが複雑になり、通常走行の処理が不明確になり、理解性が低下する問題である。そこで脱線処理を関心事として、アスペクト指向状態遷移言語を使って、通常走行に合成する。図 14 に示したプログラムは通常走行の状態遷移言語に記されている OUT マークに脱線処理を追加するためのアスペクト記述の一部である。このプログラムと通常走行を表した状

状態移行言語をウィーバに通して、通常走行に脱線処理を合成する。その結果、得られた状態移行モデルを図 15 に示す。図 15 に示す通り、複雑であった状態移行モデルがアスペクトにより隠蔽され、正常系の通常走行が明確になる。これにより理解性が保たれる。

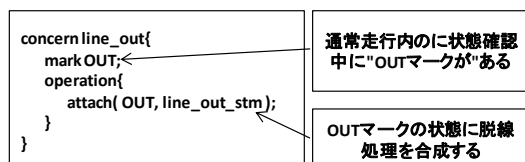


図 14 脱線のアスペクト記述の一部

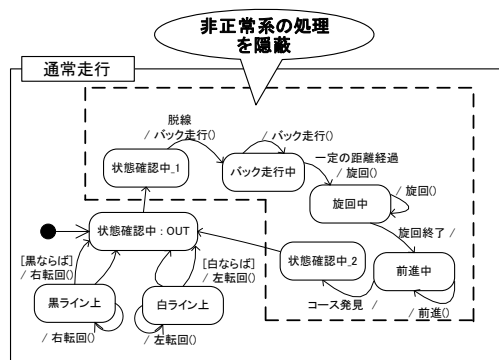


図 15 アスペクトにより非正常系のみを隠蔽化した状態移行モデル

4.3. 時間的制約の問題に状態言語を適用

(1) 並行タスクに実現する場合、障害物検知の状態移行モデルから遅延が生じる状態移行モデルを分解するためのアスペクト記述を図 16 に示す。図 17 はアスペクト指向状態言語を適用し、2.4 節図 5 で示した超音波センサの遅延を表す状態移行モデルをタスクに配置した図である。これは超音波センサの遅延が関心事となり、アスペクトで分離している。これらのタスクは並行に動作する。この図から、時間制約であるセンサの遅延を関心事として分離できることがわかる。

(2) タスク 1 つで実行する場合、指定した状態に倒立制御の状態を合成するためのアスペクトを図 18 に示す。図 19 では 2.4 節図 5 で示した倒立制御中の関数が各機能の状態移行モデル上に分散し理解性が低下することを防ぐためにアスペクト指向状態言語に適用する。図 19 は図 18 のアスペクト記述により、倒立制御の機能を個別に管理し、主要な状態の箇所を明確にすることを可能にした。

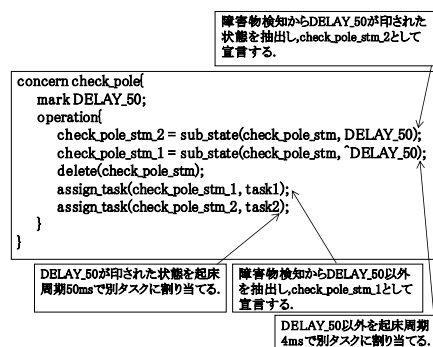


図 16 障害物検知のアスペクト記述例

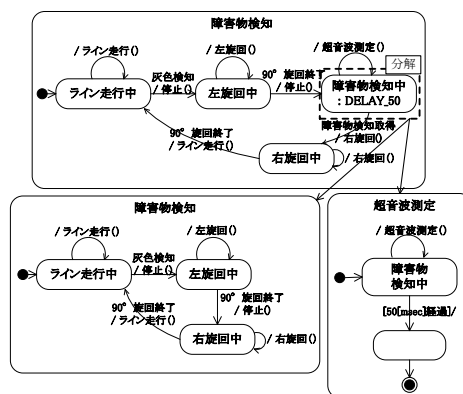


図 17 障害物検知に関するウィーブ後の状態移行モデル

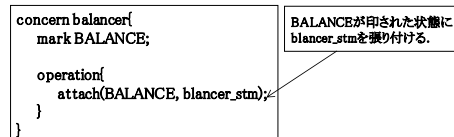


図 18 倒立制御のアスペクト記述例

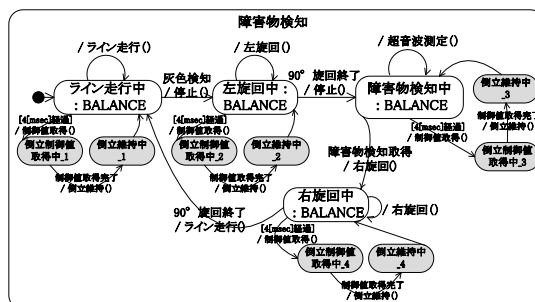


図 19 倒立制御をウィーブ後の状態移行モデル

5. 関連研究

組込みソフトウェアにおける非正常系や制約が引き起こすモデルに関する問題,我々が解決策として適用するアスペクト指向技術に関し様々な研究が行われている。佐々木, 片山らは性能制約に応じた UML の変形について論じている[1]。紫合は, 正常処理の振舞い状態マシンをもとに, 非正常処理の状態遷移を検出・追加していく方式について述べている[2]。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おり[3][4][5][6][7][8][9], 鶴林らは, 組込みソフトウェア開発への適用を提案している[3]。また, 状態遷移モデルの合成が可能なアスペクト指向技術もいくつか提案されている。先進的な状態遷移モデルの合成が可能なアプローチに MATA(Modeling Aspects Using a Transformation Approach)[8]がある。このアプローチでは, 項グラフ書き換え理論のクリティカルペア解析[10]を適用することで, 状態単位での詳細な状態遷移モデルの合成を実現している。以上の技術, アプローチのように UML に基づき標準化されたモデル駆動開発技術はモデル変換を検討する際の研究基盤として有用であり, 我々の提案方法でも取り入れている。

一方で UML より実装に近いモデルを記述する言語では, プロセッサや OS, 実装依存の問題が顕在化するため, 組込みソフトウェアのより実地的な問題の検討に有用であることが期待される。岡山, 片山の研究では, 組込みソフトウェア向け言語について, 状態遷移構文, テスト構文を持たせ, 処理系により異なる型のチェックを強化している[11]。

我々も, C 言語に状態遷移構文を持たせることが可能な言語を提案した[12]。本言語の特徴は, 既存の C 言語に, 提案した状態遷移記述を施せば, 変換器を通すことで, 状態遷移構文を解釈し, 既存の C 言語に変換することができることにある。また, グラフ形式の状態遷移モデルをリバースすることも可能である。以上の言語を MDD ロボットチャレンジ[13]の飛行船開発や ET ロボットコンテスト[13]等の組込みソフトウェア開発に適用し評価した, 最大規模は 1 万行程度である。また, 状態遷移モデルを苦手としている学生でも容易にコーディングすることができた。

本論文で提案した言語は, この状態遷移言語に, アスペクト指向の概念を取り入れ, 状態遷移記述の合成・分解により, 関心事を分離する言語を提案した。提案言語では, アスペクトと呼ぶ記述により, 状態遷移記述に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を実現する。本言語における新しい点

は, 非正常系や制約によりモデルが複雑化・追跡可能性が低下する問題を, このアスペクトに記述した操作により緩和することにある。

6. おわりに

組込みソフトウェアは, 多数の非正常系, 制約を持ち, これらは横断的な関心事であることから, モデルやプログラムが複雑になる, 追跡可能性が低下するという問題が発生する。モデル駆動開発により, これらの問題を解決することは可能であるが, 小規模で周辺機器が多様なシステムにとっては, 開発負荷が大きい。

本論文では, これらの問題を解決するために, プログラミング言語に状態遷移とアスペクト指向の概念を取り入れたアスペクト指向言語を提案した。提案した方法により, 非正常系の問題, 制約の問題が扱えることを述べた。

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] 鶴林尚靖, 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 Structural 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 Motorola Aspect - 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 Executable UML Model, Transactions on Aspect-Oriented Software Development VI, pp.1 – pp.38(2009).
- [8] J. Whittle, P. Jayarman, A. Elkhodary, A. Moreira, 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/>