

## 組み込みソフトウェアのための 構造化モデリングツールの開発

菅原秀之<sup>†</sup> 宮崎正俊<sup>†,††</sup>

近年、CPUの高機能化とともに組み込み機器、組み込みソフトウェアも大きく発展してきた。しかし、その発展によってコードの量が爆発的に増え、その管理が大きな問題となっている。またその一方でマイコン依存の開発環境による弊害も依然として残っている。そこでこれらの問題を解決する1つの方法として組み込みソフトウェアのための構造化モデリングツールの開発を行った。

### Development of structured modeling tool for embedded software

Hideyuki Sugawara<sup>†</sup> and Masatoshi Miyazaki<sup>†,††</sup>

In recent years, embedded products and embedded software have been extensively developed by development of CPU. The development makes exploding code and it causes major issue. On the other hand, negative effect of development environment depending on each micro controller is still exists. As a solution for these problems, we made structured modeling tool for embedded software.

### 1. はじめに

近年、CPUの高機能化とともに目覚ましい勢いで発展を遂げてきたソフトウェア開発もここ10年で成熟した段階にきている。言語の面ではアセンブラからCやLispのような高級言語、そしてC++やJavaのようなオブジェクト指向言語へと発展し、設計の面では構造化設計からオブジェクト指向設計へと発展してきた。だがそれ以降は設計、言語面でのパラダイムを変えるような大きな動きは出てきていない。

組み込み開発においては高機能化の流れが進み、32ビットマイコンにおいてはもはやPC上の開発と変わらない標準化された環境での開発を行えるようになってきている。一方で8ビットなどの低スペックのマイコンも依然として需要があり、こちらは消費電力などを除き、ほとんど性能が変わらないまま機種依存の環境で現在も開発が行われている。

他方、ホビー向け組み込みの分野に目を転ざると、PICマイコンやArduino、Gainerなどのホビー向けボード、あるいはレゴマインドストームのようなロボット教材などが急速に盛り上がってきている。これは組み込み開発の裾野が広がってきていることを意味する。

組み込み開発は高機能マイコン向けと、低機能マイコンまたはホビー向けに二極化、あるいは三極化した状態となっている。

### 2. 組み込みソフトウェア開発における問題点

組み込み開発において、よく取り上げられる問題は生産性である。ここ10年で作成するコードの量が爆発的に増えており、もはや従来の体制では管理しきれず、プロジェクトの遅延や障害などの影響が無視できなくなっている。一般にコードの量が倍に増えるとその保守のための管理工数は数倍に膨れ上がると言われる。開発要員についても同様である。管理の許容量を越えてしまったプロジェクトは、いわゆるデスマーチと呼ばれる状態になる。だがコードの量が爆発的に増えているのは携帯電話や車載組み込みなど、元々開発が大規模に行われているものである。マイコンの高機能化や開発環境の標準化によりコード生産性が上がったことで、現在のPC上の開発が抱える問題と同じ問題を組み込み開発も抱えることになったわけである。そこまでいかない中小規模の開発の場合、規模が小さいために大きく取り上げられることは少ないが別の問題を抱えている。

中、低スペックマイコン向けの開発に着目すると、開発環境の問題が見えてくる。

---

<sup>†</sup> アルゴソリューションズ株式会社  
Argo Solutions Co.,Ltd.

<sup>††</sup> 東北大学名誉教授  
Emeritus Professor, Tohoku University

一般に組み込み開発は対象のマイコンによって制御方法が異なる。言語こそ最近ではほぼ全てのマイコンがC言語を利用できるようになっているが、マイコン制御のためのレジスタはマイコン毎に異なり当然コンパイラも異なる。したがって開発のためのスキルがマイコン毎に異なってしまう。製造メーカーが異なれば全く別物である。高スペックマイコンではリアルタイムOS（以下RTOS）の普及やそれに伴う開発環境の発展で開発がかなり標準化されてきているが、中、低スペックマイコン向けでは容量の制約でRTOSは載せることができない。ここが標準化されない限り、この分野の開発の飛躍的な進歩は期待できず、技術者は対象のマイコンが変わるたびにその技術習得に時間を取られてしまう。それは技術者のスキル向上の大きな阻害要因の1つになり得る。

また上記が原因の1つでもあるが、組み込みソフトウェア技術者になるのは一般のソフトウェア技術者になるよりも敷居が高い。エンタープライズ系の開発であれば、1つの言語を習得し、DBアクセスのためのSQLを習得してしまえばほとんどの開発をこなすことができる。画面系のアクセス方法がわかればほぼ一式作れてしまう。それができるのは開発環境や実行環境が標準化されているからである。要件が理解できればあとは何をすればよいのかは技術者自身で判断できる。それに対して組み込み開発では、C言語習得は必須であるが、その上でマイコン毎に異なる仕様を把握するために膨大なデータシートを熟読しなければならない。しかもデータシートはマイコン開発者が作成するためにソフトウェア技術者にとっては読みにくく、慣れるまでに時間を要する。さらにソフトウェア技術者といっても組み込み開発では作業内容によって回路の知識もある程度必要になる。一般のソフトウェア技術者が組み込み開発に移行するのは容易ではない。

組み込みソフトウェアは簡単に作成できるべきものである。簡単とはいってもボタン1つでソフトウェアが完成するような簡単さではない。制御ロジック自体はユーザが作成するものである。それ以外のマイコンに依存するようなコードをユーザが記述しなくてよい、という意味である。組み込みソフトウェア自体は突き詰めて考えれば非常に単純なものである。組み込みソフトウェアの機能は、簡単に言えば外部のデバイスをマイコンのI/Oポートを通じて制御することである。対象のデバイスによって入出力信号の制御方法は変わってくるが、同種のデバイスであれば制御方法は共通している。制御を複雑にし、かつ難しくしているのは、制御の方法がマイコン固有の仕様に依存している点である。したがってマイコンの制御方法が共通になれば組み込みソフトウェアの作成も単純化できるはずである。

### 3. 解決方法としての開発ツール

そこで、上記の問題を解決する3つ手法を提案する。そして、その手法を実現する組み込みソフトウェアの開発ツールを作成した。

#### (1) マイコン制御処理を独立させて開発ツールとともに提供する

マイコン制御処理を独立させることにより、本来行いたい制御とマイコン自体の制御部分を切り分け、制御インタフェースを共通化する。さらにマイコン制御処理を開発ツールとともに提供することで、ユーザはマイコン固有の制御処理を意識することなく実現したい制御ロジックの作成を行うことができる。マイコン制御処理を切り分けることでマイコンの変更に対しても容易に対応できる。制御処理部分をデバッグ用のスタブで差し替えればPC上でのデバッグも可能になる。

#### (2) ソフトウェア駆動のフレームワークを自動生成する

マイコン上である機能を駆動させる場合、その駆動タイミングは常駐駆動、インターバル駆動、外部割込み駆動の3つに大別できる。その方法はメインループからの起動、マイコンのタイマ割込みによる起動、マイコンの外部割込みによる起動である。RTOSを使う場合はタスク登録によって起動させる。その仕組みはマイコン制御処理が共通化されていれば機械的に作成することができる。そこで(1)で提供される共通のマイコン制御処理を使い、ソフトウェア駆動のフレームワークを自動生成する。それにより一般のソフトウェア技術者が一から組み込み開発を行う場合でも混乱せずに開発を始めることができる。

#### (3) モデル駆動開発（以下MDD）の導入

(1)、(2)により、開発が標準化されていくと、制御ロジックに開発を集中できることで作成するコード量が増加してその管理が混乱することが考えられる。その対策としてMDDの導入を行う。これによって膨れ上がったコードの管理も視覚化した情報で容易に行える。また(1)、(2)と組み合わせることによりモデルを基にした自動生成が可能になる。MDDの導入は、コードベースで開発を行ってきた技術者に対しては逆に敷居を上げることになるが、保守や管理コストを考えると導入のメリットの方が大きい。導入についてはモデリングの方法を簡略化することで従来の組み込み技術者にも一般のソフトウェア技術者にも移行しやすい仕組みを構築する。モデリングの手法については構造化モデリングを採用する。構造化モデリングについては4.で説明する。

## 4. 構造化モデリング

本研究におけるモデリング手法は構造化モデリングをベースとする。MDDもしくはMDA（モデル駆動アーキテクチャ）は、オブジェクト指向設計の延長上に提案された設計手法で、ソフトウェアの「構造」や「振る舞い」を図や表を使って表現して開発を行う手法である。組み込みソフトウェア開発でよく使用されるモデリングの手法にはUMLモデリングと構造化モデリングがある。UMLモデリングはオブジェクト指向設計に基づいたモデル化を行う手法で、オブジェクト指向言語とは親和性が高い。それに対して構造化モデリングは組み込み開発で以前から行われている構造化設計の手

法をモデル化したもので、データの流れを表現したデータフローダイアグラム（以下DFD）によってソフトウェアの構造を決定する（図1参照）。UMLモデリングは、UML2.0以降になって多くの図が取り入れられ、構造化モデリングで表現できることのほとんどが表現可能になったが、その種類の多さと使用方法の柔軟さから逆に使いにくくなっている面もある。またUMLモデリングを行うためにはオブジェクト指向設計の抽象的な考え方を理解する必要があり、それが従来の開発者から敬遠される原因にもなっている。

そこで構造化モデリングを採用することで使用できる図を明確に限定し、従来の構造化設計ベース、手続きベースでモデルを構築できるようにした。DFDは機能の入力と出力を明確に記述するため、制御系の組込みソフトウェアの構造を表現するのに非常に適している。

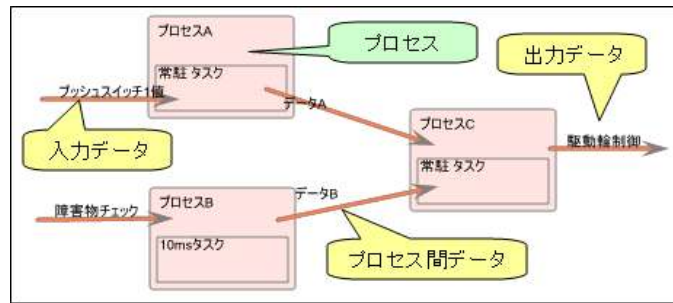


図1 DFD (データフローダイアグラム)

## 5. モデリングツール「Argo Embetics」

ツールの名称は「Argo Embetics (アルゴ エンベティックス)」といい、組込みソフトウェア開発向けのモデリングツールとして提供する。以下に詳細を記述する。

### 5.1 システム構成

Argo Embetics のシステム構成は、モデリングを行うエディタとツール内部で保持するモデル実体、そしてコードを生成するジェネレータの3つで構成される。モデル実体はモデルデータファイルという形でツール外部にXML形式で保存され、次の起動時にはモデルデータファイルを読み込んでモデル実体を構築する（図2参照）。

Argo Embetics を使用した開発は、プロジェクト単位で行う。1つの開発対象に対し1つのプロジェクトを作成する。プロジェクトの共通情報はプロジェクトファイルに

格納し、モデルの情報はモデルデータファイルに格納する。プロジェクトという考え方はソフトウェアの統合開発環境では一般的であり、従来の開発形式に慣れたユーザが混乱することはない。モデルの各要素は個別にインポート、エクスポートすることができ、部分的に再利用することも可能である。Argo Embetics はMDDにより開発を行う。ユーザはモデルを構築することで設計、製造の両方の工程をツール上で行う。構築されたモデルは、ツールによりC言語のソースコードに変換され出力される。

開発に使用するモデルの詳細は5.2で説明する。

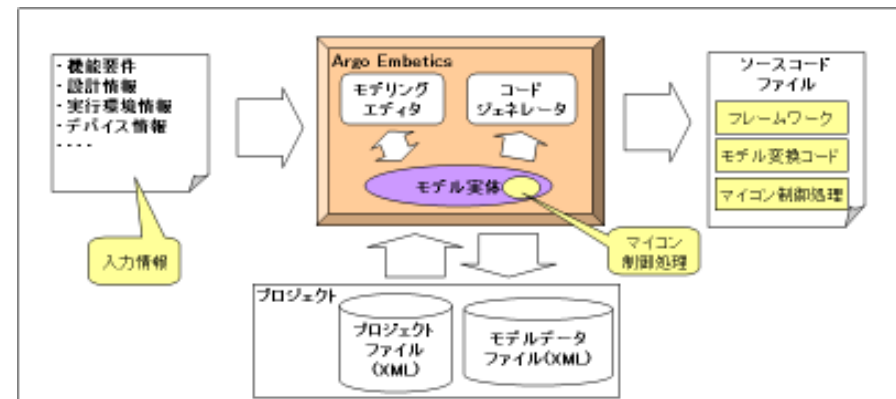


図2 システム構成図

### 5.2 モデルの構造

Argo Embetics は構造化モデリングをベースとしたモデリングを行う。モデルはその表現する内容に応じて以下に示す5つのグループに分かれ、その中の各要素が組み合わされて全体のモデルを構築する。モデルの要素の構成を図3に示す。モデル表現と実行環境との関連イメージを図4に示す。

- ハードウェアモデル  
 センサやアクチュエータなど、制御対象のデバイス、あるいはデバイスの組み合わせによるハードウェア機能を表す。
- 要求モデル  
 対象ハードウェアの制御方法、いわゆる機能要件を表す。他に制約条件や応答タイミングなどの非機能要件も表す。
- 機能モデル

ソフトウェアの構造や振る舞いを表す構造化モデリングの核となるモデル。基本設計から詳細設計、そしてコード記述まで含んだ情報を表現する。ソフトウェアの構造はDFDを用いて表す。ソフトウェアの振る舞いは状態遷移図、真理値表、それから手続き記述と呼ぶ処理手順表現を組み合わせて表す。このモデルでは構造と振る舞いは決定されるが実行するタイミングは決定されない。

- タイミングモデル  
 機能モデルにおける振る舞い、つまり処理が実行されるタイミングを表す。
- プラットフォームモデル  
 マイコンやデバイスといった実行環境に依存する情報を表す。このモデルは他のモデルから独立していて実行環境へのインタフェースの役割をはたす。

構造化モデリングを表現するのはハードウェアモデル、要求モデル、機能モデルである。タイミングモデル、プラットフォームモデルは5-1(1), (2)を実現するためにモデルに組み込んだ。

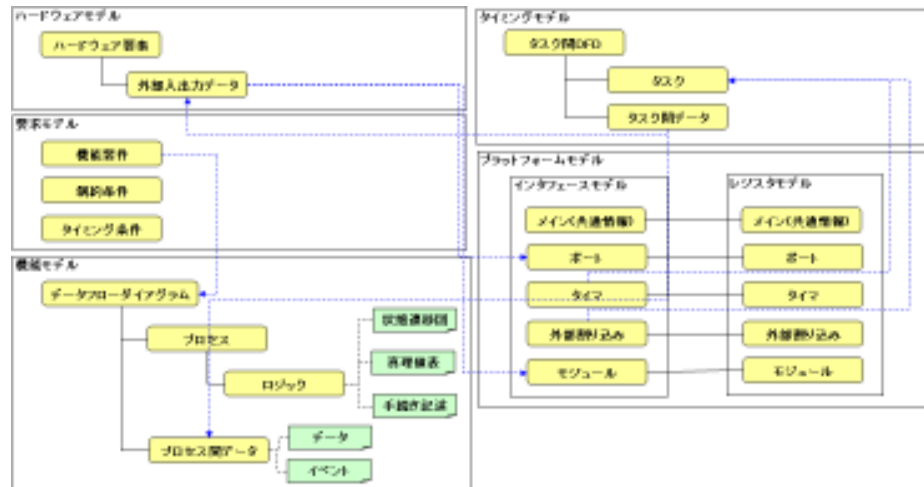


図 3 モデル要素の構成図

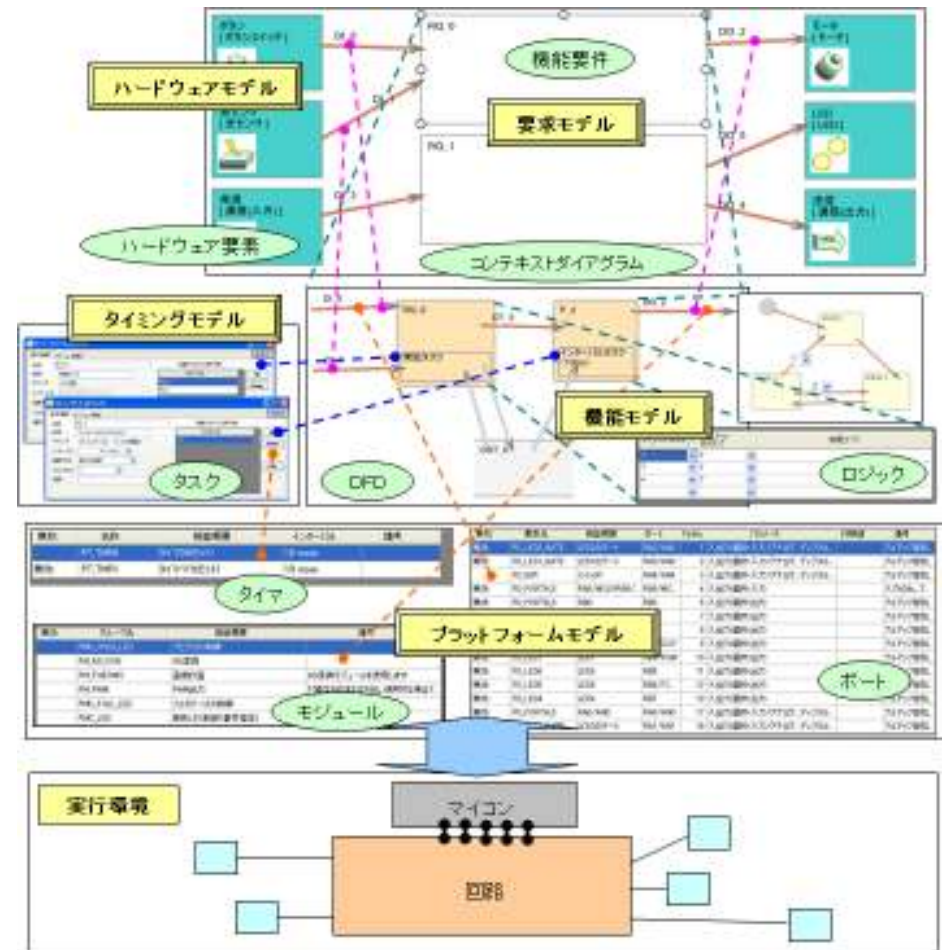


図 4 モデルの構造

ハードウェアモデルと要求モデルは組み合わせられてコンテキストダイアグラムを構成する。コンテキストダイアグラムとは、ソフトウェア本体と外部の入出力対象を記述したものでソフトウェアから見たシステムの全体像を表す。ソフトウェア本体は1つなので機能要件の記載は不要だが、機能要件と外部入出力との関係を明確にする

ために拡張して表現している。ハードウェアモデルを構成するハードウェア要素は外部の入出力データを管理する。外部入出力データはバーチャルなものだが、実行時には実デバイスを制御するデータとなる。

機能モデルは構造や振る舞いを表し、一般に設計と呼ばれる部分からコードと呼ばれる部分まで表現する。ただし、静的でプラットフォーム非依存の情報であるため、マイコン制御部分や実行されるタイミングを表す部分、つまりシステムが「処理」を駆動する部分は表現しない。ここでの「処理」とはDFDに記述されたトップレベルの処理を表す。このトップレベルの処理のことをプロセスと呼ぶ。プロセスとプロセス間を流れるデータによってDFDを構成する(図1参照)。

プロセスの振る舞いはロジックという要素によって表現される。ロジックの表記法は現在3種類あり、それぞれ状態遷移図、真理値表、手続き記述と呼ばれる表記法である(図5参照)。状態遷移図は、ある状態を持ち、その状態と入力データの内容により出力を決定していく振る舞いである。真理値表は入力データの内容だけで出力が決まる振る舞いを表現する。入力データの条件のパターンを表形式で記述し、パターン毎に処理を記述していくものである。手続き記述は状態遷移図と真理値表で表せない処理手順をそのまま手続きとして記述する。手続き記述の文法にはC言語の文法を使用する。同様に状態遷移図や真理値表内に記述する処理部分のコードもC言語文法を使用する。



図5 ロジック要素の3つの表現

タイミングモデルは機能モデルにおけるプロセスが実行されるタイミングを表す。実行タイミングにはメインループからの常駐駆動と定期的なインターバル駆動、外部の割込みをトリガとするトリガ駆動の3種類あるが、実際の起動はマイコンの動作に依存する。同一の実行タイミングのプロセスはタスクという要素で管理する。タイミングモデルはタスクの集合として構成される。

プラットフォームモデルはマイコンやデバイスといった実行環境に依存する情報を表す。共通情報、ポート、タイマ、外部割込み、モジュールの5種類の要素で構成

され、マイコン固有のレジスタアクセスの処理を内部に隠蔽する。またプラットフォームモデルは2階層構造をとる。レジスタモデルと呼ばれる下階層はマイコンだけに依存する情報を持ち、インタフェースモデル(以下I/Fモデル)と呼ばれる上階層は、レジスタモデルをラップする形でクロックやタイマ周期などのマイコン機能を使用するための設定情報を管理する(図3参照)。またセンサ制御やモータ制御など、マイコンの固有機能ではないがマイコン制御を使用する情報もここで管理する。プラットフォームモデル、特にレジスタモデルはマイコン毎に使いまわしができるため、プロジェクト毎に作成するのではなく提供されたモデルをインポートして使用する、という使い方を想定している。

機能モデルとプラットフォームモデルは、Argo Embeticsのモデリングの核をなすモデルであるため、5.3, 5.4で詳細を記述する。

モデルの構造は図4に示す形となる。最上位にシステム全体像であるコンテキストダイアグラムが配置され、その中の1つの機能要件が1つのDFDを形成する。DFDの内容は、対応する機能要件をブレイクダウンしたものとなる。DFD内の1つのプロセスはロジック要素にブレイクダウンされて機能を構成する。ハードウェア要素が管理する外部入出力データはそのまますプロセスの入出力データとなり、最終的にはロジック要素内でアクセスされる。それ自体はバーチャルなデータであるが、外部入出力データはプラットフォームモデルのポート要素やモジュール要素の関数アクセスとリンクして実際の外部デバイス制御を実現する。またプロセスの実行タイミングを管理するタスクは、プラットフォームモデルのタイマや外部割込み要素とリンクして、実際の割込み時にプロセスを起動する。このようにしてシステム全体像からソフトウェアの構造、振る舞い、実行環境の情報までくまなくモデリングすることができる。

### 5.3 機能モデル

DFDはソフトウェアの構造を表す。振る舞いや実行タイミングは表現しない。したがってDFDはコードを生成することではなく、ソフトウェアの振る舞いにも影響しない。振る舞いはプロセスに割り当てられたロジック表現にしたがって実行される。ロジック要素同士は関数呼び出しの形で入れ子にすることができる。それにより関数の呼び出し構造を入れ子ロジックだけで作ることができる。

DFDとロジックにより、ソフトウェアの構造は図6左図のように縦と横に構築される。DFDが横の構造、入れ子ロジックが縦の構造を構築する。縦の構造は関数呼び出しの構造のため理解しやすいが、横の構造は慣れていないと理解しにくい。横の構造、つまりプロセスは、機能モデル上では独立して駆動するとみなす。実際の動作は実行環境依存であるが、機能モデルは論理的なバーチャルな仕組みを表すため、機能モデル上でのプロセスは独立駆動である。そのためDFD上のプロセスの位置やデータの流れの順番は、プロセスの起動順や起動タイミングに一切影響を与えない。

DFD 以下は縦と横の構造をもつが、縦構造の間の直接のつながりはない。共通関数呼び出しやデータストアアクセスを除き、プロセス毎の構造は縦のみである。プロセス同士は横の構造をもつが、それはデータの受け渡しを行うだけのつながりである。DFD の横の構造であるプロセスの分割は、この点を踏まえた上で、処理内容の意味や駆動タイミングの切り分けによって行う。ただし、プロセスは必ず分割しなければならないものではない。プロセスの実行タイミングが一致していれば、1つのプロセスだけで振舞いを作成することも可能である。その場合は DFD の作成も不要である。DFD 内でプロセスを切り分ける理由は機能を分割することであるが、手続き的な表現方法に慣れている場合は1プロセスによる振る舞い設計でもモデリングは行える。

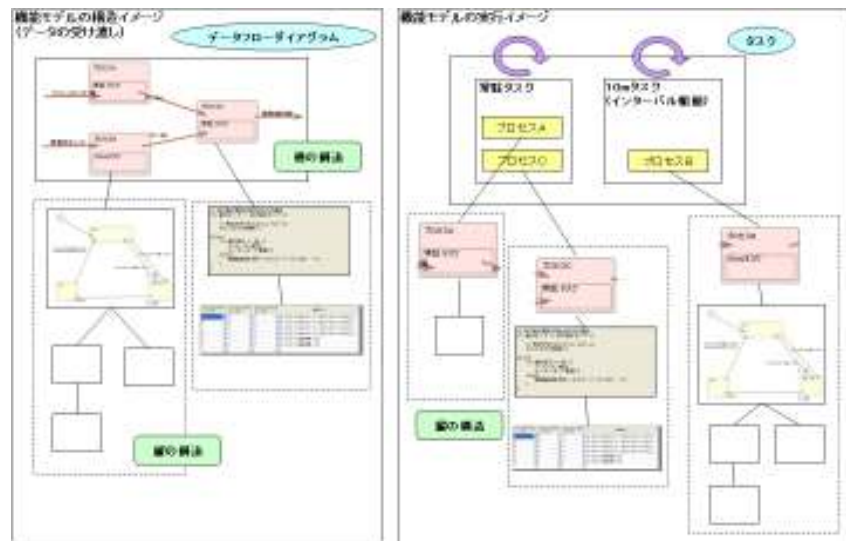


図 6 機能モデルの構造と実行イメージ

#### 5.4 プラットフォームモデル

プラットフォームモデルは前述のとおり、共通情報、ポート、タイマ、外部割込み、モジュールの5種類の要素で構成され、レジスタモデル、I/Fモデルという2階層構造をもつ。それにより実行環境であるマイコン制御のインタフェースを提供する。レジスタモデルはI/Fモデルの基になる情報を提供し、実際のマイコン制御はI/Fモデルを通じて行われる。I/Fモデルを使ったマイコンの制御には2つの種類があり、どちらの設定も必要である。1つは要素内のパラメータという形をとり、その選択内容に応じ

てマイコンの動作クロックやポートの入出力属性など、マイコンの使用方法を決定する静的な情報である。もう1つは関数呼び出しの形で提供され、ある機能の実行や外部の値の取得、設定などを行う動的な情報である。この2つの情報を使用することでマイコンの固有情報を直接アクセスせずに制御を実現している。詳細は図7を参照。

プラットフォームモデルを独立させたことでモデリングの際には対象の実行環境を意識しない形で開発を行うことできる。プラットフォームモデル以外のモデルは実行環境依存の情報をもたないバーチャルなモデルであるため、実行環境が別のマイコンに変更になる場合でもモデルの切り替えだけで対応することができる。この仕組みは通常、OSを通して行えるものであるが、プラットフォームモデルによってOSを使用しない開発やOSを搭載できない環境でも実行環境情報の切り離しが可能になる。

プラットフォームモデルで採用しているパラメータという仕組みはその選択内容に応じてマイコンの固有レジスタの設定処理を作成するものだが、この仕組みはレジスタ設定に限らず様々な用途に使用できる。たとえばパラメータの選択値を他のパラメータの設定値に組み入れる方法や、コンパイルスイッチングをパラメータの選択内容によって行う方法などがある。パラメータであらかじめ出力するコードが決定できれば、よく見るようなコンパイルスイッチだらけの見づらいコードから解放される。つまり、パラメータによるコードのカスタマイズが可能になる。

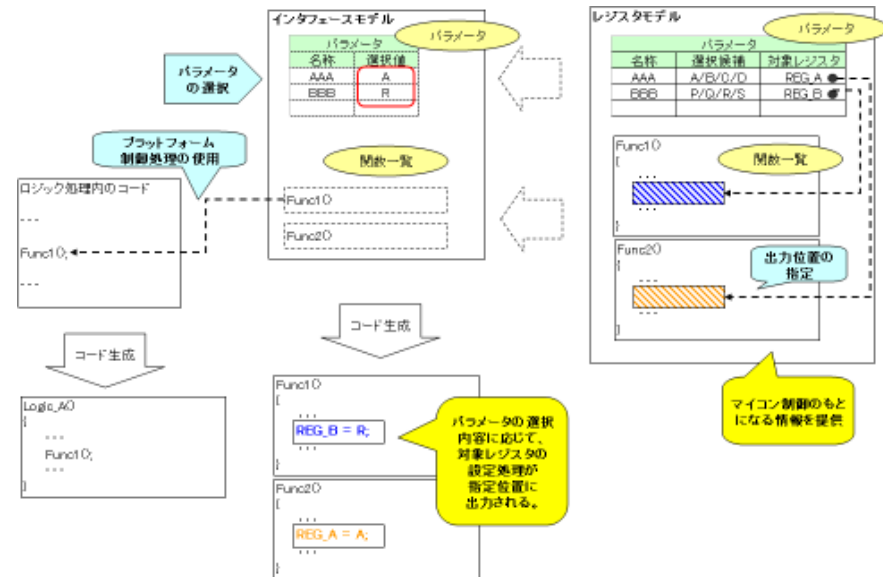


図 7 プラットフォームモデルにおけるマイコン制御の仕組み

## 5.5 生成コード

モデルから生成されるコードは C 言語のコードである。C 言語を採用した理由は現在の組込み開発で最も使用されているという点と、その文法がなじみやすい点である。C の文法は、C に限らず C++, Java, C#, php, JavaScript, Processing など様々な言語で使用されているために組込み開発の未経験者でも理解しやすい上、他の C 系の言語の出力も行いやすい。

生成されるコードの主な内容は、各プロセスに割り当てられたロジック関数と、プラットフォームモデルで定義されたマイコン制御処理である。ロジックには 3 種類あるが、状態遷移図は状態遷移を case 文で置き換えたロジック、真理値表は条件パターンを if 文で置き換えたロジックが作成される。手続き記述は基本的に記述された内容のまま出力される。

入れ子のロジックは関数の呼び出しの形で出力される。プロセスに割り当てられた最上位のロジック関数は、そのプロセスに設定された実行タイミングに応じて起動される。実際にはプロセスはタスクにより管理されるため、最上位ロジックはプロセスの管轄タスクから呼び出され、そのタスクは実行タイミングの種類により、メインループ、またはマイコンのタイマ割込み、外部割込みの関数から呼び出される。タスク管理を行う RTOS を使用する場合は駆動フレームワーク生成を行わず、各タスクを OS のタスク管理に登録して使用する。

外部入出力データについてはリンクにしたがってプラットフォームモデルのポートや関数呼び出しをマクロ定義の形で出力する。DFD 内部のプロセス間のデータについては、生成時にその設定内容にしたがってツールで内部定義した変数を出力する。

実行タイミングが異なるプロセス間のデータのやり取り、つまりタスク間のデータのやり取りについては、割込みによるデータの衝突や不正アクセスを抑制するため、コピーデータをやり取りするようにコードを生成し、データコピー時の割込みの禁止を実施するコードを出力する。

生成されるコードは一部を除いてユーザが画面上で作成したモデルの内容に基づくものである。駆動フレームワーク、あるいはタスク間のデータコピーと割込み禁止処理は独自に作成される。モデルの実行イメージは図 6 を参照。

## 6. Argo Embetics が提案する設計手法と開発手法

Argo Embetics はモデリングツールである。設計手法は基本的に MDA の手法をとる。ただし構造化モデリングを採用するためオブジェクト指向的なアプローチを取る必要はない。また全体像や構造にこだわる必要もなく、ボトムアップで構築していくことも可能である。コードベースに慣れたユーザであれば、ボトムアップで構築しながら

徐々に構造、特に DFD を作りこんでいくことができる。

作成したモデル要素はインポート/エクスポート機能により再利用が可能である。モデル資産を積み上げていけば、必要な処理をインポートしてつなぎ合わせるだけでソフトウェアが構築できる。もちろんユーザ作成のものだけでなくツール提供のモデル要素も使用可能である。それによって MDA でありながらモジュール型アーキテクチャの手法を実現することができる。

開発手法には、その開発サイクルの長さからウォーターフォール型、スパイラル型、反復型、そして最近話題のアジャイル型などがある。個々の説明は割愛するが、Argo Embetics は MDA の採用によりプロトタイプ作成までの時間が短く、なおかつ仕様変更やプラットフォーム変更にもすばやく対応できるため、アジャイル型開発が最も適していると思われる。日本では明確でない仕様書からすり合わせを行うことで製品開発を行うため、厳密に仕様を確定する海外の開発手法はそのままでは適用できないといわれているが、Argo Embetics を使えばすり合わせ型の開発でも効果を上げることができる。Argo Embetics はイメージベースでもモデルの作成が可能である。しかもモデルのシミュレーション機能を使用すれば、イメージベースのモデルでも動作検証を行うことができる。

## 7. Argo Embetics がもたらす効果

Argo Embetics は約 4 年の期間をかけて研究開発を続けている。はじめはモデリングとコード生成ができるだけのものではあったが、プラットフォームモデルの仕様を明確化して対象プラットフォームを広げ、さらに使い勝手の向上を図ってようやくモデリングツールとして使える製品となってきた。だが開発ツールとしての効果はまだまだ限定的である。現状の機能で効果が期待できるのは、当初の目的に挙げてきた、組込みソフトウェアを簡単に作れることである。それにより組込みソフトウェア技術者層の幅を大きく広げることができる。技術者に限らず多少のプログラミング知識のある一般ユーザが組込みソフトウェアを作成できるようになり、同時にプログラミングスキル不足に悩むハードウェア系技術者が自在に組込みソフトウェアを作成できるようになる。さらに既存の組込みソフトウェア技術者にとってもマイコン仕様依存の制約から解放されることになる。それによって制御ロジックに重点を置いて開発することも、逆にプラットフォーム依存処理に特化して開発を行うことも可能になる。

Argo Embetics は普及の目的で機能限定版を Web サイト上で無料提供している。開発ツールの販売方法としては常套手段であるが、英語版も海外サイトで提供しているためダウンロード数が一定量を超えれば一気に普及することが見込まれる。当然専門外的一般ユーザもツールを手にすることができ、ユーザ層は大きく広がる。Argo

Embetics が普及すると、組み込み開発のみならず MDD の手法も一般的になり、誰もが組み込み機器制御を行えるようになる。将来は現場技術者ではない一般ユーザが組み込み開発をリードしていくようになるかもしれない。

## 8. Argo Embetics の可能性

上述のとおり、Argo Embetics の効果はまだまだ限定的である。将来的には以下に示すような機能を実装し、モデルの表現力を拡張する予定である。それによりソフトウェア開発手法の新しい形を提案できる。

オブジェクト指向言語は、その継承機構により差分拡張を利用したクラスの再利用やオーバーライドによるポリモルフィズムの実現という、生産性に大きく貢献する仕組みを提供した。MDD または MDA という技法はオブジェクト指向設計の流れから登場したものである。同じようにモデルの差分拡張という機能を実装できるはずである。これができるモデル全体というアーキテクチャレベルでの差分拡張が実現できる。これはモデリングという設計機構がオブジェクト指向言語のような言語機構の仕組みを実装できることを意味する (図 8 参照)。

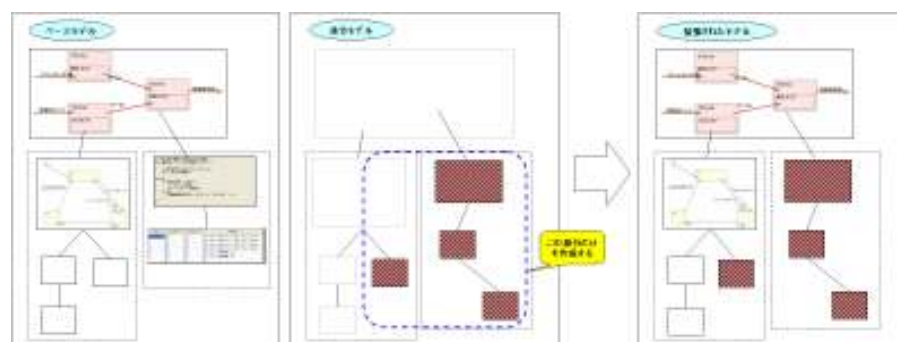


図 8 差分拡張モデル

そして、ソフトウェアの「振る舞い」のモデリングを拡張する。振る舞いのモデリングはモデリングツールの限界を示している。モデリングツールはソフトウェアの構造についてはモデル化することに長けているが、振る舞いのモデル化についてはほとんど行えていない。唯一モデル化に成功しているのが状態遷移図と真理値表、あるいはディシジョンテーブルと呼ばれるもので、実現できた理由は図や表をそのまま case 文や if 文に落とせるからである。それ以外の膨大な種類のロジックやアルゴリズムに

ついては MDA といえども手続き的なコードによって記述を行っている。それが何を意味するかというと、MDA を使用したところで結局は SE が構造設計、プログラマがコーディング、という従来の開発体制が何も変わらないということである。

Argo Embetics はこの振る舞いの部分についてもモデル化と視覚化を試みる。それはフローチャートのように論理構造だけを視覚化するものではなく、ロジックを意味のある形で視覚化するものである。それにより MDA によって近づいた SE とプログラマの境界をほぼなくすることができる。開発においては設計する対象に応じて技術者を配置するような開発体制をとることもできる。

これらの機能を実装することで、モデリングという手法がソフトウェアの単なる構造表現の方法ではなく本格的に言語機構を置き換える仕組みとなる可能性が出てくる。

## 9. おわりに

現在まで、人と組み込み機器や機械との関係はつぎのような形で推移している。

### (1) 全て機械仕掛けの機器

全て目で見ることができる。

### (2) 電気回路を使用した機器

回路や電子デバイス部分については見てもわからないが、電子機器技術者が見ればわかる。

### (3) マイコンを使用した機器

マイコンの中身がソフトウェアのため制御機構は全くわからない (⇒ブラックボックス化)。

それによって、元来身近でわかりやすかった機器が高機能化されて中身が全くわからないものになった。身近な例では、昔、街の電気店で修理できていた家電製品が、いまではメーカーの修理工場に送らないと直せなくなっている。車の電気制御部分についても同様で、街の修理工場では修理することができない。

将来的に、Argo Embetics を使用して作成した組み込み製品は、完成して出荷した後でもソフトウェアの修正やカスタマイズを行うことができるようになる。それはソフトウェアの開発者でなくても、組み込み開発の専門家でなくても可能である。モデルがあれば、街の電気店でソフトウェアを直すことも可能になる。それにより人と組み込み機器の関係を昔のような、人が修理、カスタマイズできる状態に戻すことができる。

他方、ハードウェアまでも柔軟につなげる動きがある。「Wiring」という開発環境が「Arduino」というマイコンボードで提供され、そこではハードウェアまで含めた組み込み開発のオープンソース化が推進されている。

Argo Embetics の普及は、組み込みソフトウェアというブラックボックスからの解放を意味する。組み込み開発は、今までのように製造メーカーの中にクローズしたものではな

く、多くの人々の間で共有され発展していくようなオープンな開発へと広がっていくものとなる。

**謝辞** 本研究開発にあたり、様々な形で指導を頂いた、東北大学、東北学院大学、株式会社エス・キューブ、NECソフトウェア東北株式会社、株式会社ファインアークの皆様に深謝いたします。本研究開発の一部は、宮城県地域中核IT企業成長支援事業、宮城・仙台富県チャレンジ応援基金事業、みやぎe-ブランド確立支援事業、仙台ものづくり中小企業製品開発緊急支援補助金によるものである。

### 参考文献

- 1) セサミワーキンググループ, “組込みソフトウェア開発のための構造化モデリング”, 翔泳社, 2006
- 2) T. Gardner, L. Yusuf, “Explore model-driven development (MDD) and related approaches: A closer look at model-driven development and other industry initiatives”, Mar 2006  
<http://www.ibm.com/developerworks/ibm/library/ar-mdd3/>
- 3) B. Portier, L. Ackerman, “Model Driven Development Misperceptions and Challenges”, Jan 2009  
<http://www.infoq.com/articles/mdd-misperceptions-challenges>
- 4) 高橋知伸,南光孝彦, “組込み分野へのモデル駆動開発手法の適用”, Panasonic Technical Journal Vol. 56 No. 3, Oct 2010
- 5) Wiring  
<http://wiring.org.co/>
- 6) Arduino  
<http://www.arduino.cc/>
- 7) Gainer  
<http://gainer.cc/>
- 8) レゴ マインドストーム  
<http://www.legoeducation.jp/mindstorms/>