オブジェクト指向組み込み制御システムのモデルベース開発法

吉村 健太郎[†] 宮崎 泰三[†] 横山 孝典[††,☆]

本論文では，オブジェクト指向とコード自動生成という 2 つの技術を統合化した，組み込み制御システムのモデルベース開発方法について述べる．組み込み制御システムの大規模化にともない，再利用性に優れたオブジェクト指向技術が本格的に注目されてきた．また近年，制御モデルからプログラムを自動生成するオートコードの技術が実用段階に近づいている．しかし，これら 2 つの技術の統合化には課題が多い．そこで我々は，オブジェクト指向組み込み制御システム向けの，コード自動生成を援用したモデルベース開発法を提案する．本開発法の特徴は，制御モデルから自動生成したソフト部品をラッパでオブジェクト化するとともに，そのラッパを自動生成可能としたことである．本提案方式により，モデルベース開発における実装用コード自動生成率を従来方式の 69%から 96%に向上させることができた．また，本開発法の自動車用エンジン制御システムへの適用事例を紹介する．

A Model-based Development Method for Object-oriented Embedded Control Systems

Kentaro Yoshimura,[†] Taizo Miyazaki[†] and Takanori Yokoyama[††,☆]

We present a model-based development method for embedded control systems. The development method integrates object-oriented software development and automatic program generation. The amount of embedded software is increasing dramatically, and there is an increasing demand for improving development efficiency. Object-oriented software development, which excels in the reuse of software components, has been gaining a great deal of attention. Recently, the quality and efficiency of the automatic program generation from a controller model is becoming production level. However, an integrating these methods has some difficulties. The objective of our research is to establish a model based development method for object-oriented embedded control systems adopting automatically generated software. The feature of this method is that a wrapper wraps an automatically generated function which is handled as an object, and the wrapper is automatically generated too. We develop a software that generate the wrapper from the automatically generated function. Therefore, the rate of automatically generated function is improved from 69% to 96%. We also explain a proposed development method with an application example to an engine control system for automobiles.

1. はじめに

自動車，産業用機器，家電製品など，広い分野で，組み込み制御システムが用いられている．これら組み込み制御システムでは，コンピュータ制御による高機能化・高付加価値化が進み，ソフトウェアが飛躍的に大規模・複雑化している．これにともない，新規制御ソフトウェアの開発効率を向上させるとともに，ソフトウェアの再利用が重要な課題になっている．

一般に，組み込み制御ソフトウェアの開発は，制御設計者が制御仕様（制御ロジック）を開発する制御開発と，制御仕様に基づいてソフトウェア開発者が制御プログラムを作成するソフトウェア開発の 2 つの段階からなる．

従来の開発プロセスでは，まず制御開発において，自然言語で記述した制御仕様書を作成し，その後ソフトウェア開発の段階において，制御仕様書をもとに制御プログラムを作成してきた．しかし，自然言語による仕様記述には，曖昧な記述や詳細仕様の記載漏れが発生しやすく，ソフトウェア開発者が制御設計者の意図どおりの制御プログラムを作成するのは容易なことではない．また，制御仕様をそのままコーディングす

† 株式会社日立製作所日立研究所
Hitachi Research Laboratory, Hitachi, Ltd.
†† 株式会社日立製作所オートモティブシステムグループ
Automotive Systems, Hitachi, Ltd.
☆ 現在，武蔵工業大学
Presently with Musashi Institute of Technology

るのみでは，部品化・再利用に適したプログラム構造にはならず，ソフトウェアの再利用が困難であるという問題もあった．

ところが最近，制御開発においてモデルベース開発が主流となりつつある．この手法では，制御系CAE/CADツールを使用し，制御モデルをブロックダイアグラムで表す[1]．このため，正確で可読性が良い仕様記述が可能なうえ，プログラム作成前にシミュレーションにより制御ロジックの検証が可能であるという大きな利点がある．

こうした流れを受け，ブロックダイアグラムで書かれた制御モデルから制御プログラムを自動生成するツールも開発されている[2]．人手によらずプログラムを自動生成することで，制御設計者とソフトウェア開発者との間の仕様理解の不一致や，コーディングミスによるヒューマンエラーを排除できるという利点がある．このため，制御プログラム自動生成機能はモデルベース開発における不可欠な要素になりつつある[3]．

加えて，システム内の一部の制御への適用はともかく，制御アプリケーション全体を自動生成プログラムで置きかえるのは容易なことではない．ツールにより自動生成されたプログラムは構造化されておらず，システムへの組み込みに工数がかかる．また，組み込み制御システムでは実制御対象を用いた検証コストが高く，モデル変更のたびに制御アプリケーション全体を再生成するのは適切ではない．しかし，自動生成プログラムは人間が理解しやすいモジュール構造になっていないため，部品化・再利用も容易ではない．これら課題のため，制御プログラム自動生成ツールは，主としてラピッド・プロトタイピング向けに使用され，量産ソフトウェアでの実用化には至っていないのが現状である．

一方，再利用性の良いソフトウェアを開発する手法として，オブジェクト指向プログラミングやアプリケーションフレームワークの導入が進められている[4]．しかし，ブロックダイアグラムを用いたモデルベース制御開発と，オブジェクト指向分析・設計とのギャップが大きく，従来のオブジェクト指向開発をそのまま適用するのは困難である．

そこで我々の研究の目的は，モデルベース制御開発との継続性が優れるとともに，オブジェクト指向技術とプログラム自動生成技術を統合化した，組み込みソフトウェア開発法を確立することである．

上記目的を達成するための第1の課題は，ブロックダイアグラムで書かれた制御モデルから，オブジェクト指向に基づくソフトウェア構造を容易に構築できる設計法の開発である．我々は，制御モデル中に現れるデータのうち制御ロジック上重要なデータに着目し，それらのデータを計算するブロックからなるように，制御モデルを階層的に構成する．そして，それらのデータをオブジェクトの単位とし，それらからなるアプリケーションフレームワークを構築する．これにより，ブロックダイアグラムで記述される制御モデルから，オブジェクト指向に基づくソフト構造へと，自然に移行できるようになる．また，制御ロジックの詳細部分の変更はオブジェクト内のメソッドの処理の変更に納められるので，安定したフレームワーク構成を実現できる．

第2の課題は，制御プログラム自動生成ツールにより，制御モデルから自動生成したプログラムコードのオブジェクトへの組み込み手法の開発である．我々は，アプリケーションフレームワーク（オブジェクト構成）の構築と，オブジェクト内の処理プログラムの作成を別の工程として扱う．そしてその後者向けに，制御プログラム自動生成ツールを適用する．具体的には，先に述べたブロック単位でプログラムを自動生成し，それにラッパをかぶせることでオブジェクト化する．これにより，自動生成した制御プログラムを，オブジェクト指向に基づくソフトウェア構造に違和感なく取り込むことができる．

第3の課題は，上記手法に基づいて効率的にソフトウェアを作成できる開発環境の開発である．メソッド内の処理については既存の制御プログラム自動生成ツールにより生成できるが，自動生成コードをラッピングしてオブジェクト化する工程や，制御モデルからアプリケーションフレームワークを作成する工程も，できるだけ自動化したい．そこで我々は，自動生成コードを解析してラッパを生成するツール，および，制御モデルを解析してフレームワークのプロトタイプを生成するツールを開発した．これによりアプリケーションプログラムの大部分を自動生成できる．

以上により，モデルベース制御設計との継続性が良く，また，プログラムの大部分を自動生成可能な，オブジェクト指向に基づく組み込み制御ソフトウェアの開発が可能になる．

以下本論文では，まず2章で，制御開発からソフトウェア開発への設計の流れとオブジェクトの抽出法について述べる．次に3章で，ソフトウェアの具体的構成と，自動生成プログラムを組み込んだオブジェクトの実装方法について述べる．そして4章で，制御プログラムのほとんどを自動生成可能とする開発環境について説明する．最後に5章で，本手法を自動車エンジ

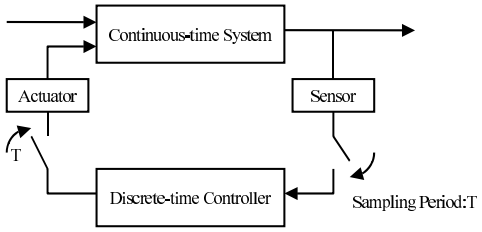

図 1 組み込み制御システム
Fig. 1 Embedded control systems.

ン制御ソフトウェア開発に適用して，評価を行う．

2. 設計の流れ

2.1 対象システム

設計法の議論の前に，本論文で対象とする組み込み制御システムについて簡単に述べる．

制御とは，JIS の自動制御用語では「ある目的に適合するように，対象となっているものに所要の操作を加えること」と定義される[5]．本論文で対象とする組み込み制御システムは，ディジタル制御システム[6] の一種である．連続時間系の対象に離散時間系のコントローラを組み込み，対象機器を制御するシステムを指す．

組み込み制御システムの概念図を**図 1** に示す．対象物は連続時間系，すなわち機械系やプロセス系である．コントローラは離散時間系であり，マイクロプロセッサなどを用いたコントローラである．コントローラは，一定サンプリング周期でセンサ計測，目標値演算，アクチュエータ出力という一連の制御処理を実行する．このような組み込み制御システムの例として，自動車用エンジン制御，空調機や冷蔵庫の温度調節制御などがあげられる．

2.2 設計の全体の流れ

本論文で提案する設計手法の全体の流れを**図 2** に示す．まず，制御開発により，ブロック線図型の制御モデルを作成する．制御開発の詳細は 2.3 節で述べる．次に，ソフトウェアの部品化を行う（(1)-1）．その詳細は 2.4 節で述べる．部品化した制御モデルをオブジェクト指向制御モデルと呼ぶことにする．

オブジェクト指向制御モデルから，個々のオブジェクトと，アプリケーションフレームワークを作成する．そのソフトウェア構成については，3.1 節で説明する．

個々のオブジェクトを作成するには，まず，既存ツールによるモデルベースプログラム自動生成により制御データ演算用関数を生成する．その詳細は 4.1 節で述べる．次に，新規開発したラッパ生成ツールによりラッパを生成し（(3)-1），演算用関数と組み合わせて，

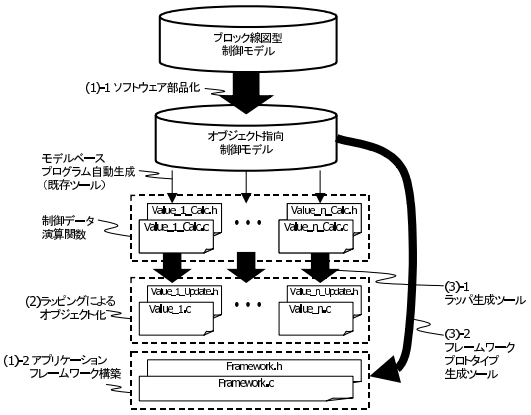

図 2 設計の流れ
Fig. 2 Design flow.

オブジェクトのソースコードを生成する（(2)）．オブジェクトのソフト構造の詳細 3.2 節で，ツールの詳細は 4.2 節で述べる．

一方，アプリケーションフレームワークを作成するには，まず，フレームワークプロトタイプ生成ツールにより，フレームワークのプロトタイプを生成し（(3)-2），必要により同時性保障のための処理を追加する．そのソフト構造の詳細は 3.3 節で，ツールの詳細は 4.3 節で述べる．

以上のように本手法は，制御モデルから制御プログラムを生成する既存ツールのみを用いる従来方法とは異なり，制御モデルの段階で部品化を行うとともに，既存ツールに加え，新規開発したラッパ生成ツールやフレームワークプロトタイプ生成ツールを使用する．これにより，部品化・再利用にに適したオブジェクト指向に基づくソフトウェア構造を実現する．

2.3 制 御 開 発

一般に，1 つの制御システムは複数の制御機能の組合せで構成される．したがって制御モデルも，制御機能ごとに作成するのが普通である．

制御モデルは，制御系 CAE/CAD ツールにより設計する．作成した制御モデルはツール上で実行可能であるため，シミュレーションによってロジックの妥当性を検証することができる．

図 3 に，制御モデルを表すブロックダイアグラムの例を示す．図中のブロック（方形）は処理を，接続線（矢印）はデータを表している．この例では，目標エンジントルク（TargetTorque）や目標スロットル開度（ThrottleOpening）の演算を行うブロックが記載されている．この制御機能にとっての入力データであるエンジン回転数（Engine Revolution），エンジン状態（Engine Status），アクセル開度（Accelerator

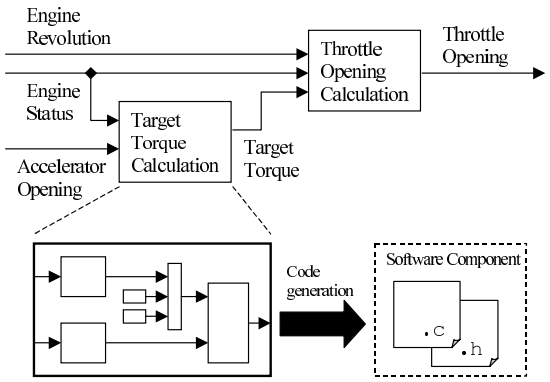

図 3 制御ブロック図の例
Fig. 3 An example of block diagram.

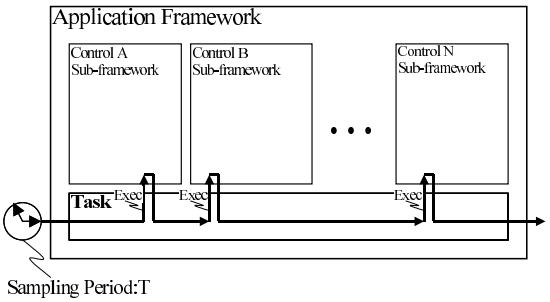

図 4 アプリケーション構造
Fig. 4 Structure of application software.

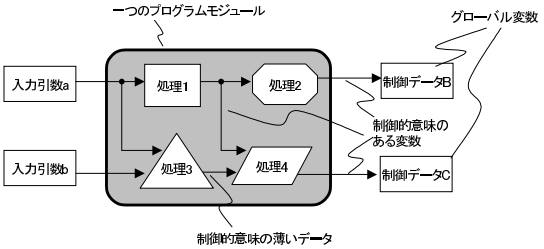

図 5 従来のソフトウェア部品
Fig. 5 Conventional software component.

Opening）は，他の制御機能によって計算される．このような制御ロジックで表された処理を周期的に実行することによって，制御が行われる．

設計者が理解しやすい制御モデルとするため，ブロックダイアグラムは階層的に表現することが多い．図 3 の例では，目標トルク算出ブロック内の処理が，下位層のブロックダイアグラムで表現されている．この階層化については，できるだけ設計者が理解しやすく，修正も容易な形にすべきである．

そこで我々は，ブロックダイアグラム中に現れるデータに注意して，ブロック化および階層化を行うこととした．すなわち，ブロックダイアグラムの上位階層には，制御として意味のあるデータのみが現れるように階層化する．制御として意味のあるデータとは，たとえば，制御目標値や，システムの状態などである．計算中に現れる一時的なデータは対象としない．そして，ブロックダイアグラムの上位階層では，それらのデータを算出する単位でブロックを構成し，それらブロック内の具体的算出処理は，その下位層のブロックダイアグラムとして記述する．

一般に，制御として意味のあるデータは制御ロジック変更時でも削除・追加が少ないため，上位階層のブロックダイアグラムの変更が必要となる場合は少なく，多くの場合，下位層の修正のみで対応できる．

2.4 ソフトウェア開発

ソフトウェア開発では，制御開発で作成した制御モデルをもとに，アプリケーションフレームワーク，およびそれを構成するオブジェクトを実装する．

アプリケーションフレームワーク（以下，フレームワーク）は，アプリケーションを実現するための標準的なオブジェクトの組合せを定義したものである[7]．我々はすでに，組み込み制御システム向けの時間駆動に基づくオブジェクト指向モデルを提案しており[8)~11)]，本論文で述べるフレームワークはこのモデルに基づいている．

前述のように，一般に制御システムは複数の制御機能から構成されるので，フレームワークも制御機能ごとに分割することができる．これを，サブフレームワークと呼ぶことにする．すなわち，**図 4** に示すように，アプリケーションフレームワークは複数のサブフレームワークの集合からなる．

サブフレームワークは制御機能ごとにオブジェクト構成を表したものである．オブジェクトは，制御開発により作成された上位階層のブロックダイアグラムを参照して抽出する．抽出にあたっては，部品化・再利用性に適したオブジェクト構成とすることに留意する．そのためには，再利用しやすいサイズにするとともに，制御的意味を考慮して，独立性の高いオブジェクトとすることが重要である．

従来の組み込み制御ソフト開発では，制御ロジックにおける処理に着目してソフトの部品化を行ってきた．しかし処理単位の部品化は，再利用の観点ではなく，その時点での制御ロジックの実装に都合の良い単位で区切ってしまうことが多い．設計者がとらえる機能の単位は個々の設計者により異なるのが普通で，1 つの機能が複数の処理を含むことも多い．このため，**図 5** に示すように，部品が大きくなる傾向があり，複数の制御項目が混在したり，出力変数が複数存在したりす

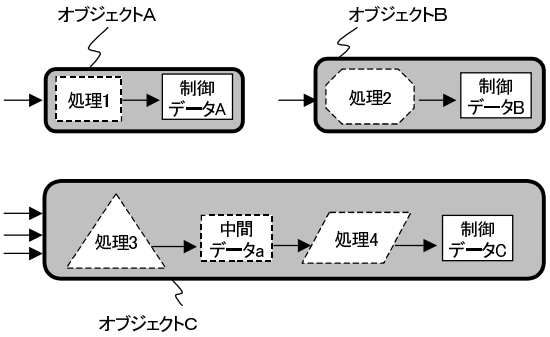

図 6 提案するソフトウェア部品
Fig. 6 Proposed software component.

る原因となっていた．これがスパゲッティプログラム化を助長し，再利用性の低いソフトとなる原因の 1 つになっていた．

これに対し本論文では，データに着目してオブジェクトを抽出する．前述のように，制御モデルの設計においては，制御として意味のあるデータに着目してブロックダイアグラムを作成しているので，そのデータをオブジェクトの候補とする．ただし，制御設計者とソフトウェア開発者では視点が異なるため，必要によりこの段階で制御モデルのブロック構成を見直しながら，オブジェクトを抽出する．

この方法では，**図 6** に示すように，制御的に意味のあるデータと，それを計算する処理をまとめたものがオブジェクトになる．それらデータの計算に用いる中間データは制御ロジックの変更による追加・削除が発生しやすいため，オブジェクトの内部に隠蔽する．

これにより，安定したオブジェクト構成を提供でき，制御ロジックの変更に対してフレームワーク自体は修正せず，オブジェクトの置き換えで対応できる．また，制御的に意味のあるデータがオブジェクトの粒度となるため，オブジェクトが巨大化することはなく，オブジェクト交換の影響範囲も明確になるうえ，個々のオブジェクトの単体テストも容易になる．

3. ソフトウェアの構成と実装

3.1 全体ソフトウェア構成

以下，具体的なソフトウェアの構成および実装方法を，自動車制御システムを例に説明する．**図 7** は，本方式を適用したエンジン制御システムの全体ソフトウェア構成である．ソフトウェアは，アプリケーションソフトウェアと基本ソフトウェア（Platform Software）に分割する．

基本ソフトウェアは，リアルタイム OS と入出力ドライバとで構成する．リアルタイム OS には，欧米で

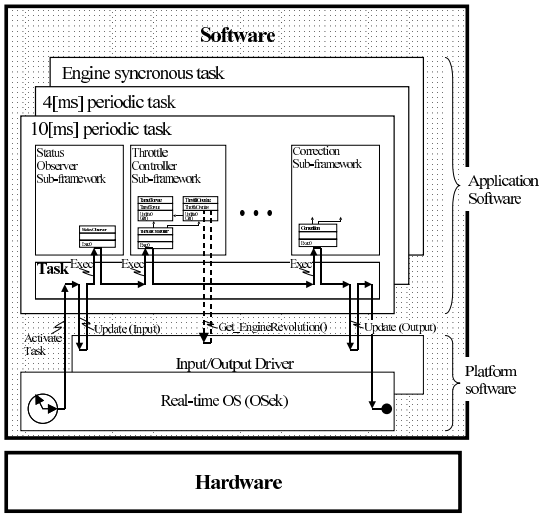

図 7 組み込み制御システムの構成例
Fig. 7 An example of proposed embedded control systems.

採用例の多い OSEK-OS [12] を用いる．また，アプリケーションの移植性を高めるため，入出力ドライバのインタフェースはハードウェア非依存のインタフェースとする．

アプリケーションソフトは，時間周期タスク（ex. 10 [ms]，4 [ms]）およびイベントタスク（ex. エンジン回転同期）とを有している．制御機能を実現するサブフレームワークは，タスクから呼び出す形でシステムに実装される．

図 7 を例に，システムの動作の概要を解説する．まず，タイマ割込みを受けてリアルタイム OS が周期タスク（10 [ms] タスク）を起動する．タスクの先頭で，入力ドライバに対して入力信号の更新を要求する．次に，タスクに登録されているサブフレームワーク “Status Observer”，“Throttle Controller”，…，“Correction” を，あらかじめ規定された順序で呼び出す．サブフレームワークは，それを構成するオブジェクトの処理を実行して，制御に必要なデータの算出・更新を行う．そして，タスクの最後で出力ドライバに出力を要求し，処理を終了する．

3.2 オブジェクトの実装

オブジェクトは実装効率を重視し，いわゆるオブジェクト指向言語ではなく，C 言語を用いて実装する[13]．以下の説明では，**図 8** に示す TargetTorque オブジェクトを例に取り上げる．このオブジェクトは，図 8 中のクラス図に記載したように，算出したデータの値を記憶するための属性 “TargetTorque” と，そのデータを算出し更新するメソッド “Update()”，データの値

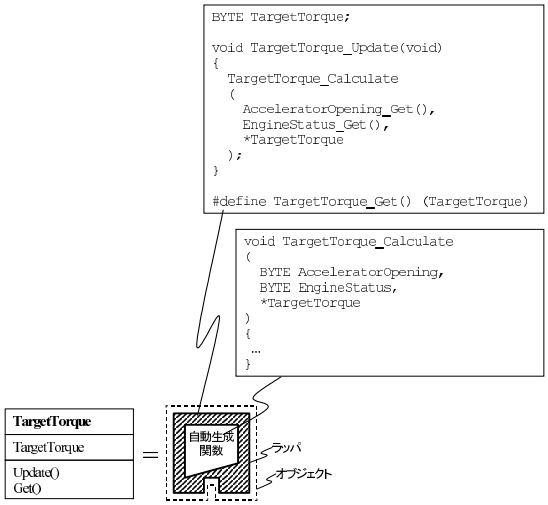

図 8 オブジェクトの例
Fig. 8 An example of object.

を読み出すアクセスメソッド “Get()” を持つ．

オブジェクトは，制御プログラム自動生成ツールが制御モデルのブロック図から生成した演算処理関数にラッパをかぶせることで実現する．演算処理関数は，“データ名_Calculate()” という名称として生成する．入力値は当該関数に対して引数として値渡しし，出力値は引数としてポインタ渡しした出力変数に格納する．図 8 の例では，“TargetTorque_Calculate()” という演算処理関数を生成する．

演算処理関数をオブジェクト化するため，ラッパでは，オブジェクトの属性（データ）とデータ更新メソッドおよびアクセスメソッドを定義する．以下，その詳細について述べる．

(a) 属性の宣言
オブジェクトの計算結果を記憶するための属性（データ）の宣言を行う．実装上はデータ名そのものを名称に持つ変数である．図 8 の例では，属性 “TargetTorque” を宣言している．

(b) データ更新用メソッド
オブジェクトの属性（データ）を算出し，上記属性に記憶する処理を実行する．データ更新用メソッドは実装上 “データ名_Update()” という C の関数である．本メソッドは自動生成関数の入力引数の値を，その値を計算するオブジェクトのアクセス用メソッドを呼び出して得て，自動生成関数 “データ名_Calculate()” に渡す．さらに，自動生成関数が計算した値を属性に記憶する．図 8 の例では，“TargetTorque_Update()” という名称のメソッドで，オブジェクト “AcceleratorOpening” と “EngineStatus” が算出したデータ値を，それらオブジェクトのアクセス用メソッド “AcceleratorOpening_Get()” と “EngineStatus_Get()” を呼び出して得て，演算処理関数 “TargetTorque_Calculate()” の入力引数に渡すとともに，演算処理関数が算出した値の記憶先として，属性 “TargetTorque” を指定している．

(c) アクセス用メソッド
他のオブジェクトが計算結果を参照するための，“Get” というアクセス用メソッドを定義する．実装上は，“データ名_Get()” という名称とするが，処理の効率化のためマクロとする．図 8 の例では，“TargetTorque_Get()” というアクセス用メソッドが，マクロとして宣言されている．マクロ化することで，3.3 節で述べるデータ同時性確保処理の実現も可能となる．

本方式の特徴の 1 つは，データ更新用メソッドが引数を持たないことである．組み込み制御システムでは制御ロジックの変更にともなって，各オブジェクトが参照する他オブジェクトのデータを変更・追加することが多い．本方式では，自動生成したラッパのデータ更新用メソッド内部に，変更・追加の影響を隠蔽できる．これにより，各オブジェクトのデータ更新用メソッドを呼び出すサブフレームワークの処理を変更する必要がなくなり，そのまま再利用可能できる．

3.3 サブフレームワークの実装

図 9 にサブフレームワークの概念図を示す．サブフレームワークは，3.2 節で抽出したそれぞれ固有の計算を行う複数のオブジェクトと，それらのオブジェクトを呼び出して処理を行うサブフレームワークオブジェクトからなる．サブフレームワークオブジェクトの主な役割は，制御設計者が意図した順序でオブジェクトを呼び出すことで，目的とする制御機能を実現することである．

制御データに対応するオブジェクトとサブフレームワークオブジェクトの組合せの例として，エンジン制御における例を**図 10** に示す．この例では，“ThrottleController” というサブフレームワークオブジェクトが，オブジェクト “TargetTorque” と “ThrottleOpening” のデータ更新用メソッドを呼び出している．

サブフレームワークオブジェクトの構成および処理の例を**図 11** に示す．この例では，実行順序を管理するサブフレームワークオブジェクト（ThrottleController）のほか，同時性保証のためのローカルオブジェクト（EngineStatus (Local)）が存在する．

サブフレームワークオブジェクトは，オブジェク

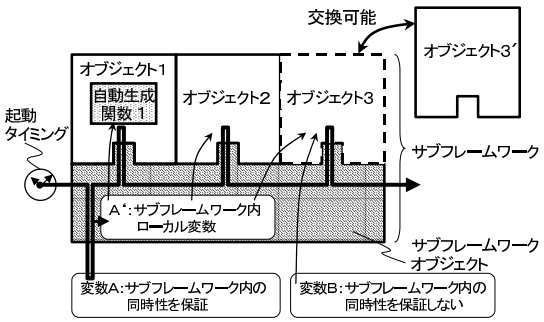

図 9 サブフレームワーク
Fig. 9 Sub framework.

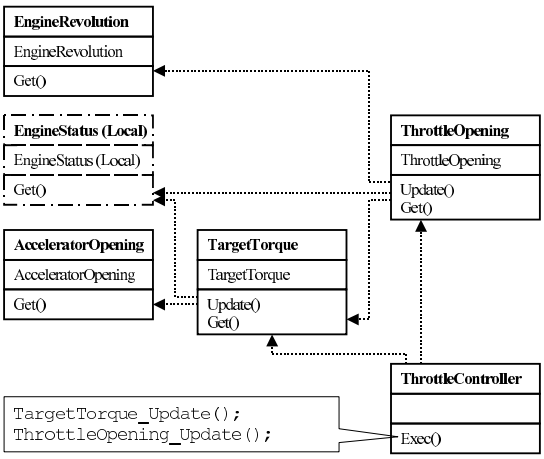

図 10 サブフレームワークの組合せ例
Fig. 10 An example pattern of sub framework.

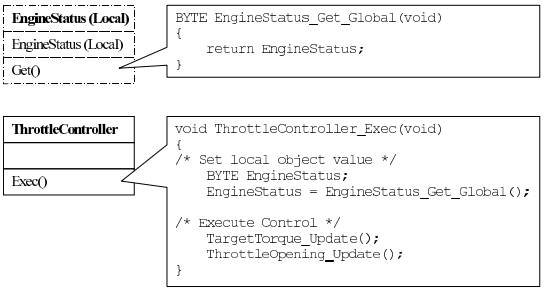

図 11 サブフレームワークオブジェクトの例
Fig. 11 An example of sub framework object.

トのデータ更新用メソッドを呼び出す処理を行う．これはメソッド “Exec” により行われる．実装上は，“制御機能名_Exec()”（図 11 の例では “ThrottleController_Exec()”）という C の関数として記述する．周期内でのオブジェクトの実行順序は，制御モデルブロック図（図 3）でのデータの流れに矛盾しないように起動する必要がある．すなわち，制御モデルブロック図の上流から下流に向かってオブジェクトのデータ更新用メソッドを呼び出すように記述する．並行実行可能なメソッドについてはどちらを先に呼び出してもよいが，設計者の意図を考慮し，左から右へ，上から下へという順に呼び出すこととする．呼び出し順序の違いによりデータの計算順序やタイミングが変化することがあるが，これについては参照側の同時性保障処理で対応する．

一般に，組み込み制御システムでは異なる周期・優先度の制御処理からなるマルチタスク構成となる．このため，同一制御周期で起動される複数のオブジェクトが，異なる周期で起動される他の同一オブジェクトのデータを参照する場合に，異なったタイミングで更新された値を参照する可能性がある．しかし，正しい制御を行うには，同一タイミングで更新された値を参照する必要がある．我々は，同時性の保証が必要なデータを，サブフレームワークオブジェクトのローカルオブジェクトに記憶することでこの問題を解決する．

たとえば，図 3 の例では，変数 “EngineStatus” は他のサブフレームワークにおいて更新されている．“EngineStatus” を更新するサブフレームワークの実行優先度が “ThrottleController” サブフレームワークより上位の場合，“ThrottleController” サブフレームワークの実行中に変数 “EngineStatus” が更新されることがある．そこでサブフレームワーク内での変数の同時性を保証するため，“EngineStatus” ローカルオブジェクトを生成する．

実装上は，“EngineStatus” というローカル変数を宣言し，“ThrottleController_Exec()” メソッドの先頭において “EngineStatus” オブジェクトのデータをローカル変数にコピーする．アクセス用メソッド “EngineStatus_Get()” はマクロ定義のため，サブフレームワーク内でのスコープはローカル変数へと移され，変数の同時性が保証される．

以上により，プリエンプション処理の影響を受けずに各制御処理を実行可能になる．さらに，ユーザが同時性の必要な最小限の変数を指定できるため，オーバヘッドを軽減できる．

4. ソフトウェア開発環境

4.1 制御モデルの作成と演算処理関数の生成

制御モデルの作成は，制御系 CAE/CAD ツールを使用する[1)]．作成された制御モデルは，制御系 CAE/CAD ツールのシミュレーション機能を用いて，ロジック検証が可能である．

最近の制御系 CAE/CAD ツールでは，浮動小数点

を用いた場合のみでなく，多くの組み込み制御システムで採用している固定小数点を用いた場合の検証も可能である．固定小数点化するには，データ値の単位や範囲を決定するスケーリング作業を行った後，固定小数点化の影響（誤差など）についても，シミュレーション機能により検証する．

次に，制御プログラム自動生成ツールを用いて，作成した制御モデルから演算処理関数の C コードを生成する[2]．前述のように，演算処理関数にラッパをかぶせてオブジェクト化するため，コード生成の単位は，オブジェクトに対応するデータを算出するブロック単位で行う．図 3 では，目標エンジントルク算出（“Target Torque Calculation”），あるいは目標スロットル開度（“Throttle Opening Calculation”）という単位でコードを生成する．

演算処理関数は，“算出データ名_Calculate()” という関数とする．目標エンジントルク算出の場合は，“TargetTorque_Calculate()” という関数になる．この演算処理関数は “AcceleratorOpening” と “EngineStatus” を入力引数とし，出力値はポインタ渡しされた “TargetTorque” に格納する．同様にして，オブジェクトに対応するすべての制御ブロックについて，演算処理関数を自動生成する．

4.2 ラッパの生成

我々は，オブジェクト化の作業を自動化するため，ラッパ生成ツールを開発した．ラッパ生成ツールは，制御プログラム自動生成ツールが生成した演算処理関数を解析して，3.2 節で述べた形式のラッパを生成する．

図 8 の TargetTorque オブジェクトを例に，ラッパの生成法について簡単に説明する．ラッパ生成ツールは，演算処理関数 “TargetTorque_Calculate()” を解析してラッパを生成する．オブジェクトの属性名は演算処理関数の出力引数名により決定する．この例では “TargetTorque” である．これからデータ更新用メソッド名も決定でき，この例では “TargetTorque_Update()” という名称のメソッドとする．また，演算処理関数の入力引数名から，データ更新用メソッドが必要とする他のオブジェクトの名称を決定し，そのオブジェクトのアクセスメソッドを呼び出す．この例では，入力引数 “AcceleratorOpening” と “EngineStatus” から，アクセスメソッド “AcceleratorOpening_Get()” と “EngineStatus_Get()” を呼び出すことを決定する．

このようにラッパは定型的な構成をしているため，演算処理関数やその引数の命名法をルール化しておけば，演算処理関数のソースコードを解析し，機械的な処理でラッパを生成することができる．

4.3 サブフレームワークオブジェクトの生成

我々は，サブフレームワークオブジェクトの作成の自動化についても検討した．サブフレームワークオブジェクトは，基本的に，そのサブフレームワーク内のオブジェクトのデータ更新用メソッドを，ブロックダイアグラムの上流から下流に向かって呼び出す処理を行えばよい．したがって，その限りにおいては制御モデルからの自動生成も難しくはない．しかし，タスク構成に依存する同時性保証の処理を制御モデルのみから生成するのは容易ではない．機械的に，すべての入力データについて同時性保証処理を挿入するのであれば容易だが，資源の限られた組み込み制御システムにおいては必要最小限にしたい．

そこで我々は，同時性保証処理の追加は必要によりソフト設計者が行うものとし，同時性保証の処理を除いたサブフレームワークのプロトタイプを制御モデルから生成するツールを開発した．たとえば，図 11 の例では，“EngineStatus” というローカル変数の宣言，および，“EngineStatus” オブジェクトのデータをローカル変数にコピーする処理を除く部分が，ツールにより生成される．実際の制御アプリケーションでは，同時性保証処理の挿入が必要な場合は限られ，ほとんどの場合は生成したプロトタイプをそのまま使用できるため，実用上はこの方法で問題ないと考えている．

4.4 再利用ベースの開発

前節まで，新規に組み込み制御システムを開発するものとして説明を行ってきた．ところが，現実の組み込み制御システム開発の多くは，過去に作成したシステムの修正あるいは機能追加である．そこでここでは，再利用ベースの開発手順について簡単に述べる．

前述のように，我々の手法では制御ロジックの一部修正については，フレームワークの修正は不要で，オブジェクトの交換で対応可能である．すなわち，サブフレームワークオブジェクト，および制御データに対応するオブジェクトのうち制御の変更がないものはそのまま再利用し，変更があるオブジェクトのみ置き換える．したがって，制御系 CAE/CAD ツールによる制御モデル修正の後，変更部分のみに対して，既存コード生成ツールによる演算処理関数の生成と，ラッパ自動生成ツールを用いたオブジェクト化のみを行えばよい．

一方，新規制御の新規追加や制御ロジックを根本的に変更する場合には，サブフレームワークオブジェクト自体の変更が必要になる．すなわち，ツールを用い

て制御モデルからサブフレームワークオブジェクトのテンプレートを生成し，必要によりデータの同時性保証処理を追加する．また，新規作成した制御の演算処理関数およびラッパを生成する．ただしこの場合も，変更がないオブジェクトについてはそのまま再利用できる．

5. 評　　価

5.1 適用評価

自動車用エンジン制御システムに対して本方式を適用し，コード自動生成率および再利用性に関して評価を行った．評価結果を**表 1** に示す．対象とした制御プログラムは，ソースコード実行数比でソフトウェア全体（OS，I/O など含む）の 42%である．

まず制御モデルをもとに実装用コードを開発し，コード自動生成率について評価を行った．評価の結果，対象とした制御プログラムのうち既存ツールで生成した部分は 69%，ラッパ生成ツールで生成した部分は 27%，ハンドコーディングが必要な部分は 4%であった．ハンドコーディングが必要な部位は，サブフレームワーク上での同時性保持処理であり，サブフレームワーク生成ツールによるプロトタイプに対して見直しを行った．モデルベース開発において従来方式では 69%だった実装用コード自動生成率を，本提案方式を用いて 96%に向上させることができた．

さらに，いったん開発が完了した制御機能（提案方式適用部位の 12%）に対して設計変更を行い，再利用性についての評価を行った．本方式では，追加・変更の少ない制御的に意味のある変数の単位でオブジェクト化しているため，変更が多い部分はオブジェクトの内部に隠蔽できる．そのため，変更対象となるオブジェクトのみ変更すればよく，他のオブジェクトおよびサブフレームワークオブジェクトは再利用できる．制御ロジックを変更した結果，一部のデータ値オブジェクトの変更のみで対応できた．ロジック変更のないデータ値オブジェクトはすべて再利用可能となった．またオブジェクト構成の変更がなかったため，サブフレームワークオブジェクトはそのまま再利用可能となり，ハンドコーディングは不要であった．この結果，設計変更における工数を 70%削減することができた．

以上の結果より，本方式によって組み込み制御システムの生産性を向上できる見込みを得られた．また，本論文で提案した新規ツールにより生成したラッパとサブフレームワークのプログラムは定型的構造のため，ハンドコーディングと比較して品質および性能上の問題は見られなかった．今後の課題として，次のものがあげられる．まず，現状の開発環境は複数のツールが独立に存在し，必ずしも使いやすいものではない．それらツールを有機的に結合した統合開発環境の実現が必要である．また変数同時性確保処理に関し，4%のハンドコーディングが必要となっている．このため，複数の時間周期およびイベントを横断して実現される制御機能における変数間の依存関係を統一的に扱う手法の開発を進めていく．

表 1 本提案方式の適用評価
Table 1 An evaluation of the proposed method.

本手法の適用範囲	42% （コード行数比）
内， オートコーディング （既存ツール利用部） （提案ツール利用部） ハンドコーディング	 96% （69%） （27%） 4%

5.2 関連する研究との比較

組み込み制御以外の分野でも，モデルベース開発が進んでいる．それらの分野の多くは，UML によるモデル記述が主流である[14]．モデルを UML で記述することで，モデル検証後に，自然にオブジェクト指向設計・プログラミングに移行できる．

ところが組み込み制御分野では，制御系 CAE/CAD ツールによるブロックダイアグラムを用いた制御モデルがすでに定着しており，ブロックダイグラムによる制御開発から，オブジェクト指向ソフト開発に自然に移行できる設計法が要求される．しかし，そのような狙いの研究は多くない．

そのような中，ブロックダイアグラムを拡張してオブジェクト指向的なソフトウェア設計機能を導入するとともに，コード生成をも可能なツール[15] を用いて，オブジェクト指向ソフトウェア開発を行う手法が発表されている[16]．この手法は基本的に，リアルタイムシステム向けに提案されたオブジェクト指向開発法[17] を，ブロックダイアグラム・ベースのモデル記述法・ツールを使用して行う方法の提案である．このため，制御開発の段階でもオブジェクト指向に関する知識が必要である．

これに対し我々の手法は，制御開発の段階ではオブジェクト指向を意識せずに，ブロックダイアグラムにより純粋に制御モデルを作成し，その後のソフトウェア開発の段階でオブジェクト指向にスムーズに移行するのが特徴である．このため，自動車エンジン制御のように，制御開発がシステム開発の大きな部分を占める組み込み制御システムにおいて特に有効である．

また，システム全体を UML を用いたオブジェクト

指向モデルで記述し，その一部をブロックダイアグラムと対応させることで，ブロックダイアグラムから自動生成したコードを組み込み可能とする手法が発表されている[18]．この手法では，ブロックダイアグラム（データフロー）による制御モデルを UML 表現に変換するとともに，ブロックダイアグラムから生成したプログラムをカプセル化するためのクラスを生成する．ただし，ブロックダイアグラムから UML 表現への変換では，オブジェクト指向を意識せずに作成した構造がそのままオブジェクト構成に反映されるため，制御処理の部分は，必ずしも再利用性の高いオブジェクトに部品化されるわけではない．特に，ほとんどをブロックダイアグラムによる制御モデルで記述した場合，制御仕様の変更によりシステム全体の構造（フレームワーク）が大きく変わる恐れがある．本手法は，大部分は UML で記述し，一部のみブロックダイアグラムで記述するような，制御処理が占める割合が小さいシステム向きであると考える．

これに対して我々の手法は，ブロックダイアグラムから制御的に意味のあるデータを抽出してオブジェクトとすることで，制御処理の部分も部品化再利用に適したオブジェクト構造とすることができる．したがって，自動車エンジン制御システムのように，制御処理が大部分を占めるシステムに適用した場合でも，再利用性の高いフレームワークを実現できるという特徴がある．

6. おわりに

組み込み制御システムのモデルベース開発法として，オブジェクト指向とプログラム自動生成という 2 つの技術を統合化した開発法を提案した．本報告で提案した手法は，ブロックダイアグラムによる制御モデルの開発に引き続き，オブジェクト指向に基づく組み込み制御ソフトウェアを定型的な作業により開発できる．また，オブジェクトのメソッド内の処理，それをオブジェクト化するためのラッパ，さらにフレームワークの大部分のプログラムを自動生成可能とした．これにより，組み込み制御システムのソフトウェア生産性および再利用性を向上させられる見通しを得た．

今後は，実製品への適用時の効果を，既存ツールによる生成コードを含めたシステム全体の品質・コストの面から検証するとともに，より使いやすい統合開発環境を実現する予定である．

謝辞 本研究を進めるにあたりご支援をいただいた（株）日立製作所日立研究所の川端敦部長（前），野木利治部長，箕輪利通主任研究員，オートモティブシステムグループの根本守主任技師に感謝する．

参考文献

1) Moscinski, J.: *Advanced Control with MATLAB and Simulink*, Ellis Horwood, Ltd. (1995).
2) Thomsen, T.: Integration of International Standards for Production Code Generation, *SAE technical paper series*, No.2003-01-0855, Detroit (2003).
3) 渡辺政彦：組込みソフトウェア向け開発環境，情報処理，Vol.45, No.1, pp.10–15 (2004).
4) Hermsen, W. and Neumann, K.J.: Application of the Object-Oriented Modeling Concept OMOS for Signal Conditioning of Vehicle Control Units, *SAE technical paper series*, No.2000-01-0717 (2000).
5) 日本工業規格：自動制御用語（一般）Z8116，日本規格協会．
6) 美多 勉：ディジタル制御理論，昭晃堂 (1984).
7) Johnson, R.E.: Frameworks = (Components + Patterns), *Comm. ACM*, Vol.40, No.10, pp.39–42 (1997).
8) Naya, H., Narisawa, F., Yokoyama, T., Ohkawa, K. and Amano, M.: Object-Oriented Development Based on Polymorphism Patterns and Optimization to Reduce Executable Code Size, *Technology of Object-Oriented Languages and Systems — Tools-25* (1997).
9) 横山孝典，納谷英光，成沢文雄，倉垣 智，永浦渉，今井崇明，鈴木昭二：組込み制御システムのための時間駆動オブジェクト指向ソフトウェア開発法，電子情報通信学会論文誌，Vol.J84-D-I, No.4, pp.338–349 (2001).
10) 吉村健太郎，宮崎泰三，横山孝典，入江 徹，藤本慎哉：組込み制御向けアプリケーションフレームワーク構成法と自動車制御システムへの適用，第 5 回組込みシステム技術に関するサマーワークショップ予稿集，pp.3–5 (2003).
11) 吉村健太郎，宮崎泰三，横山孝典：オブジェクト指向組込み制御システムのためのコード自動生成援用によるモデルベース開発法，組込みソフトウェアシンポジウム 2003，pp.64–71 (2003).
12) OSEK-VDX: *OSEK-VDX Operating System Ver 2.2.1* (2002).
13) Narisawa, F., Naya, H. and Yokoyama, T.: A code generator with application-oriented size optimization for object-oriented embedded control software, *Object-Oriented Technology: ECOOP'98 Workshop Reader*, LNCS-1543, pp.507–510, Springer (1998).
14) 山田正樹：モデル駆動開発とその周辺，情報処理，Vol.45, No.1, pp.3–9 (2004).
15) Gebhardt, M., Stier, G., Beaumont, A. and Noble, A.: Automation of ECU Software De-

velopment: From Concept to Production Level Code, *1999 SAE International Congress & Exposition*, No.1999-01-1174 (1999).
16) Fuchs, M., Nazareth, D., Daniel, D. and Rumpe, B.: BMW-ROOM — An Object-Oriented Method for ASCET, *1998 SAE International Congress & Exposition*, No.981014 (1998).
17) Selic, B., Gullekson, G. and Ward, P.T.: *Real-Time Object-Oriented Modeling*, John Wieley & Sons (1994).
18) Kühl, M., Reichmann, C., Prötel, I. and Müller-Glaser, K.D.: From Object-Oriented Modeling to Code Generation for Rapid Prototyping of Embedded Electronic Systems, *Proc. 13th IEEE International Workshop on Rapid System Prototyping*, pp.108–114 (2002).

(平成 16 年 10 月 29 日受付)
(平成 17 年 4 月 1 日採録)

吉村健太郎（正会員）

1976 年生．1999 年早稲田大学理工学部機械工学科卒業．2001 年同大学大学院理工学研究科機械工学専攻修士課程修了．同年（株）日立製作所日立研究所入社．組み込み制御システムの研究開発に従事．IEEE，ACM，SAE，日本機械学会各会員．

宮崎　泰三

1963 年生．1987 年京都大学工学部機械系学科卒業．1989 年東京大学大学院工学研究科産業機械工学専攻修士課程修了．同年（株）日立製作所日立研究所入社．モータ制御，ハイブリッド車制御の研究開発を経て，システム開発環境の研究開発に従事．電気学会，自動車技術会，SAE 各会員．

横山　孝典（正会員）

1959 年生．1981 年東北大学工学部通信工学科卒業．1983 年同大学大学院工学研究科電気及通信工学専攻修士課程修了．同年（株）日立製作所日立研究所入社．1987〜1990 年まで（財）新世代コンピュータ技術開発機構出向．2001 年同社オートモティブシステムグループ．2004 年より武蔵工業大学．分散システム，組み込みシステム，ソフトウェア工学等の研究に従事．博士（情報科学）．電子情報通信学会，IEEE，ACM 各会員．