組み込み制御システムのためのオブジェクト指向コード生成ツール

成沢文雄[†] 納谷英光[†] 横山孝典[†,☆]

本論文では，組み込み制御システム向けの，メモリ消費量の少ないオブジェクト指向プログラムを生成する手法およびツールを提案する．一般にオブジェクト指向言語を用いると，クラス・インスタンス，継承，動的束縛，データ抽象等のオブジェクト指向プログラミングに特有の機能を実装するため，プログラムのコードサイズが増大する．組み込み制御システムの多くはメモリ容量の制約が厳しく，これがオブジェクト指向導入の障壁の 1 つとなっている．ところが現実の組み込み制御システムでは，必ずしもオブジェクト指向の特徴機能のすべてが必要ではない．そこで我々は，オブジェクト指向設計により作成した仕様書から，対象とするアプリケーションに不可欠な最低限のオブジェクト指向機能のみを実装した C 言語プログラムを生成する，コード生成ツールを開発した．そして，本ツールを現実の組み込み制御システムに適用し，有効性を確認した．

An Object-oriented Code Generator for Embedded Control Systems

Fumio Narisawa,[†] Hidemitsu Naya[†] and Takanori Yokoyama[†,☆]

The paper presents a method and a code generator to generate program codes with less memory consumption for embedded control systems. In general, the code size of an application program written in an object-oriented language is larger than the code size written in a non-object-oriented language, because object-oriented functions, such as class and instance, inheritance, dynamic binding, and data abstraction, require some resources. This makes it difficult to introduce object-oriented technology to embedded control systems, because their memories are limited. But an embedded control system may not require all of the functions. We have developed a code generator for embedded control systems, which inputs an object-oriented specification and generates an application program written in C language with just necessary object-oriented functions. We have evaluated the code generator applying embedded control systems.

1. はじめに

オブジェクト指向技術は，ソフトウェアの部品化・再利用を促進し生産性を向上させるものとして，情報系計算機システムのソフトウェア開発において広く使用されているが，組み込みシステムのソフトウェア開発においてはそれほど普及してはいない．その大きな障壁の 1 つは，オブジェクト指向言語の使用によるコードサイズの増大である．多くの組み込みソフトウェア開発者がオブジェクト指向分析・設計の導入を検討しているものの，コードサイズの制約によりオブジェクト指向言語の適用が難しいため，オブジェクト指向技術の適用を躊躇しているといわれている[1)]．この傾向は，組み込みシステムの中でも制御システムにおいて特に強い．

我々が対象としているのは，家電，自動車制御，FA 等の組み込み制御システムである．組み込みシステムの中でも，情報端末や情報家電のような情報系の組み込みシステムでは比較的大きなメモリが搭載される．これに対し組み込み制御システムの多くは，コスト上の制約から，メモリを内蔵したワンチップ・マイクロコントローラを使用するのが普通である．この場合，すべてのプログラムを内蔵メモリに搭載しなければならないにもかかわらず，そのメモリ容量はきわめて限られる．たとえば，自動車制御分野向けのワンチップ・マイクロコントローラの内蔵メモリ容量は 64〜512 KByte 程度である．我々の対象のうちの典型的なマイクロコントローラの ROM 容量は 62 KByte，RAM 容量は 2 KByte である．このようなマイクロコントローラにオブジェクト指向言語で記述したプログラムを組み込むことは困難であった．

† 株式会社日立製作所日立研究所
Hitachi Research Laboratory, Hitachi, Ltd.
☆ 現在，武蔵工業大学コンピュータ・メディア工学科
Presently with Department of Computer Science and Media Engineering, Musashi Institute of Technology

そこで，分析・設計はオブジェクト指向で行うが，プログラミングは非オブジェクト指向言語を用いることで，コードサイズの増大を避ける方法も用いられている．しかし，非オブジェクト指向言語でオブジェクト指向プログラミングを行うには，そのためのプログラミング技術を習得する必要がある．また，オブジェクト指向言語を使用する場合に比べ，工数が増大したりバグが発生しやすくなったりするという問題もある．これらの問題が発生しない解決法が求められている．

オブジェクトコードのサイズが増大する原因は，オブジェクト指向言語を使用すると，クラス・インスタンス，継承，動的束縛，データ抽象等のオブジェクト指向の特徴機能を実現するためのコードやデータをコンパイラが出力するためである．これらを削減できれば，メモリ消費量を減らすことができる．

オブジェクト指向プログラムのメモリ消費量を削減する方法として，組み込みシステム向けに機能を制限したオブジェクト指向言語仕様を新たに定義する方法がある．Embedded C++[2] はその代表例であり，C++[3] から多重継承，例外処理，テンプレート等の機能を除くことで，メモリ消費量を削減している．しかし，組み込みシステム一般に広く使用できる汎用的なプログラミング言語を目指しているため，削除できる機能には限界がある．個々のアプリケーションごとに不要な機能を削除できれば，さらにメモリ消費量を削減することが可能である．

オブジェクト指向言語の最適化に関する研究[4)~6)]も多くなされている．これらは，コンパイラがソースコードを解析して，無駄な部分や最適化可能な部分を抽出し，コードの削除や最適化を行うもので，ソフトウェア開発者に負担をかけることなく個々のアプリケーションごとに最適化ができるという点で，有用な技術である．しかし，いったん多機能な言語仕様に従って記述したソースコードを解析し，最適化対象部分を抽出することは容易ではなく，ワンチップ・マイクロコントローラを使用した組み込み制御システムが要求するほどの高い効果を得るのは難しい．

そこで本研究の目的は，組み込み制御システム向けのメモリ消費量の少ないオブジェクト指向プログラムの開発法と，それを実現するためのツールを開発することである．

我々は，コードサイズを削減する方法について，プログラミングの段階のみでなく設計段階をも含めて検討を行った．そして，これまでのいくつかのオブジェクト指向開発の経験から，組み込み制御システムの場合には，詳細設計の終了時点で，対象とするアプリケーションの実装に必要でないオブジェクト指向の機能が明らかになることが分かった．

オブジェクト指向設計では，UML[7] のような図式表現により，オブジェクトの仕様を記述する．組み込み制御システムの場合，固定的なオブジェクト構成になることが多いため，この段階で，クラスの仕様はもちろん，具体的なインスタンスの構成まで明確になるのが普通である．たとえば，動的なインスタンスの生成は不要であるとか，1 つのクラスには 1 つのインスタンスしか存在しないとか，オブジェクト構成が固定的なので動的束縛が不要であるといった情報を得ることができる．

そこで我々は，それらの情報を用いることで，オブジェクト指向の不要な機能を排除し，メモリ消費量の少ないプログラムを作成する方法を検討した．そして，設計時に作成した図式表現によるオブジェクト指向仕様書から，実装上不要な機能を除いたコードを自動生成するツールを開発した[8)]．本ツールは，実装上不要な機能をユーザが指定することで，それらの機能を排除した，C 言語記述によるソースコードを出力する．

この方法は，コンパイラによる最適化のようにプログラム作成後にコードを最適化するのではなく，あらかじめ不要なコードをプログラムに盛り込まないようにするため，より高い効果が得られるものと期待できる．またコード自動生成により，ユーザは C 言語（非オブジェクト指向言語）によるオブジェクト指向プログラミングの技術を習得する必要がなく，プログラミングの工程が不要になるうえ，人手コーディングによるバグも発生しない．また最近注目されている，OMG（Object Management Group）の MDA（Model-Driven Architecture）[9)] に代表されるモデル駆動開発（Model-Driven Development）における，プラットフォーム依存モデル（PSM，Platform Specific Model）からコードへの自動変換への適用も期待できる．

本論文の構成は以下のとおりである．まず 2 章で，オブジェクト指向言語の提供するオブジェクト指向の特徴機能を取り上げ，組み込み制御システムの実装時におけるそれらの必要性（削除可能性）について論じる．次に 3 章で，開発したコード生成ツールの機能や，その入力であるオブジェクト指向仕様書について説明する．4 章では，削除対象とするオブジェクト指向の特徴機能ごとに，その内容と効果の詳細を述べる．そして 5 章で，本ツールを実際の組み込み制御ソフトウェアに適用した事例を紹介した後，6 章で関連する研究との比較を行い，7 章で結論を述べる．

2. 組み込み制御システムにおけるオブジェクト指向機能

2.1 オブジェクト指向の特徴機能

組み込み制御システムをオブジェクト指向で実装する場合に削除可能な機能について検討する前に，オブジェクト指向言語が提供する特徴機能とは何かについて考える．オブジェクト指向言語が備えるべき機能については必ずしも統一見解があるとはいえないが，本論文では，Wegner によるオブジェクト・ベース言語（Object-Based Language）の分類法[10] に基づいて，オブジェクト指向の特徴機能を考える．

Wegner は，オブジェクト（Object），クラス（Class），継承（Inheritance），データ抽象（Data Abstraction），強い型付け（Strong Typing）の 5 つの次元（Dimension）により，オブジェクト・ベース言語を分類している．ここで，オブジェクトは操作（Operation）と操作の結果を記憶するための状態（State）の両者を有するものと規定されている．

そして，オブジェクトの概念を持つ言語をオブジェクト・ベース言語，それに加えてクラスの概念をも持つ言語をクラス・ベース言語（Class-Based Language），さらに加えて継承の概念を持つ言語をオブジェクト指向言語（Object-Oriented Language）と分類している．また，オブジェクト指向言語のうち，データ抽象と強い型付けの両者の特徴を持つ言語を，強い型付けのあるオブジェクト指向言語（Strongly Typed Object-Oriented Language）と呼んでいる．ただし，データ抽象と強い型付けは独立した概念で，それぞれ単独でも成立するとしている．ここで，データ抽象とは手続きを介してしかオブジェクトの状態（インスタンス変数）にアクセスできないこと，強い型付けとはオブジェクトの型（クラス）をコンパイル時に決定できることである．

これに従えば，オブジェクト指向（広い意味での「オブジェクト指向」であり，Wegner の分類では「オブジェクト・ベース（Object-Based）」に対応）に必要不可欠な特徴概念は「オブジェクト」であり，さらに，「クラス」,「継承」,「データ抽象」,「強い型付け」の 4 つが付加的な特徴概念であると見なすことができる．

ただし，クラスという概念にはいくつかの機能が含まれ，実装上の観点からは，クラスから動的にインスタンスを生成するかどうかで，必要とする機構が大きく異なる．そこで本論文では,「インスタンスの動的生成」という機能項目を,「クラス」という項目から独立させる．これにより,「クラス」の主な機能は，複数のインスタンスによるメソッドの共用になる．また「強い型付け」に関しては，実装上は，強い型付けがある場合ではなく，それがない場合に追加コードが必要になる．具体的には，強い型付けがない言語では実行時にクラスの解決を行うために，動的束縛（Dynamic Binding）の機能が必要になる．そこで本論文では,「強い型付け」の代わりに,「動的束縛」を特徴機能として扱うことにする．

表 1 オブジェクト指向の付加的特徴機能と実装機構
Table 1 Object-oriented functions and implementation.

付加的特徴機能	実装機構
クラス	メソッドのインスタンス引数化
インスタンスの動的生成	構築子・消滅子
継承	仮想テーブル
動的束縛	仮想関数，メソッド検索
データ抽象	アクセスメソッド

以上により，本論文では，オブジェクト指向において必要不可欠な特徴機能は「オブジェクト」であるとする．そして,「クラス」「インスタンスの動的生成」「継承」「データ抽象」「動的束縛」の 5 つは付加的な特徴機能であると見なす．そして，それら付加的特徴機能について，組み込み制御システムの実装における削除可能性を論じることとする．

上記の付加的特徴機能を実現するには，そのための機構が必要になる．5 つのオブジェクト指向の付加的特徴機能と，C++においてそれらを実装する機構を**表 1** に示す．クラスによるメソッド共用を実現するには，対象とするインスタンスを指定するため，メソッドの実装にインスタンスへのポインタを引数として与える．インスタンスを動的に生成・消去するには，構築子・消滅子を用いる．継承を用いたときには，仮想テーブル等が生成される．動的束縛を実現するには，仮想関数とメソッド検索ルーチンが必要となる．データ抽象は，アクセス権を適切に設定したアクセスメソッドを用いることで実現される．

2.2 組み込み制御システムの実装に必要な機能

我々の手法では，オブジェクト指向分析・設計の段階では特に制限は設けず，いずれの付加的特徴機能も使えるものとする．そして，プログラムの実装時に，付加的特徴機能のうち不要なものがあれば，それを実現するための機構をプログラムから排除することで，メモリ消費量を削減する．

我々がこれまでに扱った 5 つの組み込み制御システムについて，表 1 に示したオブジェクト指向の付加的特徴機能が，実装（プログラミング）の段階でも必要かどうかを調べた結果を**表 2** に示す．○は必要，×は

表 2 組み込み制御システムにおける付加的特徴機能の必要性
Table 2 Necessary functions for embedded control systems.

組み込み制御システム	クラス	動的生成	継承	動的束縛	データ抽象
自動車エンジン制御システム	×	×	×	×	×
自動車車間距離制御システム	×	×	×	×	×
エアコン制御システム	×	×	×	×	×
エレベータ制御システム	○	×	×	×	×
医用分析装置	○	×	○	×	○

不要であったことを示している．

対象としたシステムについて簡単に説明する．自動車エンジン制御システム[11]と自動車車間距離制御システム[12]は連続系の制御が主体のシステムである．エアコン制御システムとエレベータ制御システム[13]はイベント・ドリブンな制御主体のシステムである．これらはいずれもワンチップ・マイクロコントローラを使用している．医用分析装置もイベント・ドリブンな制御を行うが，他のシステムと異なり，制御以外の分析結果に対するデータ処理が大きな比重を占めているという特徴を持つ．また医用分析装置は，組み込み制御システムとしては大型で，外付けメモリを使用している．

表 2 を見ると分かるように，エレベータ制御システムの場合はクラスが必要で，医用分析装置の場合はクラス，継承，データ抽象の機能が必要であった．それ以外のシステムの場合はいずれの機能も不要であった．

エレベータ制御でクラスが必要になったのは，複数のエレベータや各階で使用する同一種類の部品が多く，それに対応して同一クラスに属する複数のインスタンスが必要なためであった．また，医用分析装置でクラス，継承，データ抽象が必要であったのは，制御処理ではなくデータ処理の部分である．クラスと継承は分析結果を記憶するデータ構造の表現に有効であった．またデータ抽象を必要としたのは，アクセスメソッドによりデータの正規化や単位変換を行ったためである．なお，分析結果を記憶するためのインスタンスを動的生成することも考えられるが，従来より動的な記憶領域の確保は行っていないため，必須ではなかった．

これらの結果から，純粋な制御処理については，オブジェクト指向の付加的特徴機能の多くを，実装時には排除できることが分かった．

継承機能については，設計時には有用でも，実装時に必要とは限らない．例として自動車の車種によってスロットルの種類が異なる場合を考えると，既存のスロットルのクラスを継承して新しいスロットルのクラスを定義できる機能は，設計時には有効である．ところが 1 つの車種では 1 種類のスロットルしか使用しないので，実装時には，継承クラスの内容を含めて 1 つのクラスで表現してよい．これにより継承機構を不要にできる．

同様に，オブジェクトおよびそれらの間の関係が固定的であるため，インスタンスの動的生成・消去やメソッドの動的束縛の機能は不要なことが多い．たとえば，制御システムではセンサやアクチュエータの構成が決まっているため，それらを表すインスタンスの動的生成や動的束縛は不要である．また，プログラムは固定的で ROM に焼かれるため，データ抽象の機能を厳密に実装する必要はない．それから，制御処理の場合，インスタンスが 1 つしか存在しないクラスが多く，その場合はクラスの機能は不要になる．

以上のように，我々が対象とする組み込み制御システムの場合，実装時には，オブジェクト指向の付加的特徴機能を必ずしも必要としないのが普通である．したがって，アプリケーションに応じて，不要な付加的特徴機能の実現機構を削除したコードを生成することにより，メモリ消費量を削減することができる．なお，5 つの付加的特徴機能のすべてを必要とするシステムに対しては，通常のオブジェクト指向言語の処理系により実装するものとし，本論文で提案するツールの対象外とする．しかし，そのような組み込み制御システムは少なく，本手法はほとんどの組み込み制御システムに適用可能と考える．

3. コード生成ツール

3.1 ツールの位置付け

設計時に作成した図式表現によるオブジェクト指向仕様書から，実装時に不要なオブジェクト指向機能を除いたコードを生成するツールを開発した．本ツールは，実装上不要な機能をユーザが指定することで，それらの機能の実装機構を排除した，C 言語記述によるソースコードを出力する．**図 1** にプログラム生成ツールの入出力と処理の概要を示す．

我々は入力となるオブジェクト指向仕様書の記述法として，Martin と O'dell によるオブジェクト構成図（Object Structure Diagram）とイベント図（Event Diagram）の記法[14]を採用した．オブジェクト構成図はクラス（オブジェクト）の記述に，イベント図はメソッドの記述に使用する．イベント図は，処理の流れを直感的に分かりやすい形で詳細に記述できるのが

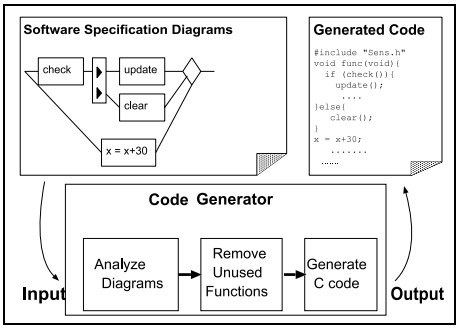

図 1 コード生成ツールとその入出力
Fig. 1 Input and output of the code generator.

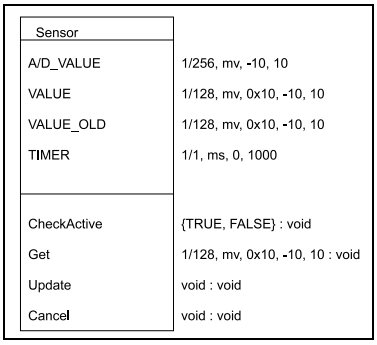

図 2 オブジェクト構成図外観
Fig. 2 An example of an object structure diagram.

特徴である．これら仕様書は，ツールの入力としてのみでなく，設計者，テスト担当者，キャリブレーション担当者等が参照するドキュメントとしても用いられる．オブジェクト指向仕様書の詳細については，3.2 節で述べる．

なお，オブジェクト指向仕様書の記法として UML を使用しなかったのは，UML が普及する前から，Martin と O'dell の記法を我々が使用していたためである．ただし UML が広く使用されるようになったため，我々は現在，UML のクラス図とアクティビティ図を入力できるツールを開発中である．オブジェクト構成図と UML のクラス図はほとんど同じ記法であり，イベント図と UML のアクティビティ図も記法は異なるが，ほぼ同様の内容を記述するものである．

3.2 オブジェクト指向仕様書

3.2.1 オブジェクト構成図

オブジェクト構成図はクラス定義の静的な情報を記述するものである．詳細な仕様を記述するために，オブジェクト構成図には，データメンバやメソッドの型の情報を記載するよう，Martin，O'dell の記法を拡張した．

一般には，設計段階ではデータ型や単位までは特定せず，実装段階で決定することも多い．ところが組み込み制御システムでは，ソフト設計者とは別の制御設計者が制御仕様を決定することが多く，データの精度が制御性能に影響を及ぼすため，制御仕様書でデータの精度や単位が指定されていることが多い．また自動車制御のキャリブレーションと呼ばれる工程では，データに関する詳細情報を参照できる必要がある．このため，データの分解能や単位の情報も記述可能とする必要があった．

オブジェクト構成図には，そのオブジェクトが持つデータとメソッドの一覧と，それぞれのデータの値のとりうる範囲，分解能，物理的な単位を記載する．これらは，コード生成ツールがデータ型を決定する際にも使用される．なお，以下ではこれらの並び

分解能 物理単位 (オフセット) [最小値： 最大値]

をデータの “型情報” と呼ぶ．ただし，オフセットが 0 のときは省略することができる．またメソッドの静的な情報として，引数の型情報と返り値の型情報を記述する．

なお，このメソッドの引数の型情報と返り値の型情報を “返り値の型情報：引数の型情報” と表記したものを以下では，“メソッドの型情報” と呼ぶ．

図 2 の例では，“Sensor” がクラスの名前である．また，“A/D_VALUE”，“VALUE”，“VALUE_OLD”，“TIMER” がデータであり，“CheckActive”，“Get”，“Update”，“Cancel” がメソッドである．

データ名とメソッド名の同一行右欄には，それぞれの型情報を記載する．たとえば，図 2 のデータ “A/D_VALUE” の型情報は，分解能が 1/256，物理単位が mV，オフセットは 0（オプションであり省略した場合は 0），最小値が −10，最大値が 10 である．また，処理 “CheckActive” は引数をとらず，返り値として “TRUE” あるいは “FALSE” のどちらか一方をとる．

3.2.2 イベント図

イベント図（Event Diagram）は個々の処理の詳細を記述したものであり，オブジェクトに対するメッセージ送信と，その連接，並行，条件分岐，ループの制御構造を描くことのできる図である．これにより，メソッドの処理内容を記述する．イベント図の解析を容易にするため，条件分岐は “if...then...else”，ループについては “while” に対応するものに限定した．また，1 つのメソッドは C 言語の 1 つの関数で実装することを原則としている．データの値の計算式はイベント図の箱の中に C 言語の文法に従って記述する．

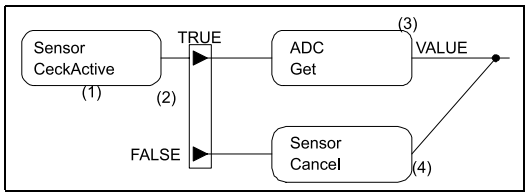

図 3 イベント図の例
Fig. 3 Example of an event diagram.

```
void Sensor_Update(void){
  if (Sensor_CheckActive()){
    VALUE = ADC_Get();
  }else{
    Sensor_Cancel();
  }
}
```

図 4 コード出力例
Fig. 4 Example of generated code.

イベント図の例を**図 3** に示す．図の処理では，センサのオブジェクトの値の更新判定を行っている．このイベント図では，以下のような処理を記述してある．まず,(1) で，センサの更新条件が成立している否かを判定し，その結果により（2）で分岐する．“TRUE” の場合には（3）で AD 変換のオブジェクト（ADC）から AD 変換値を取得してデータ “VALUE” に記憶し，“FALSE” の場合には（4）で値のキャンセルを行う．

これらの仕様記述から生成した C 言語のコード出力の例を**図 4** に示す．

3.3 不要機能の指定方法

コード生成ツールは，不要と判断したオブジェクト指向の付加的特徴機能を排除したコードを生成する．ここで，表 1 に示した 5 つの付加的特徴機能のうちどれを排除するかは，基本的に設計者が設計結果に基づいて指定する．不要機能の判定も自動化できるのが望ましいが，たとえばインスタンス数のように，オブジェクト構成図とイベント図のみからは抽出が困難な情報があることや，前述のように組み込み制御システムの場合は，詳細設計時に不要な機能を明らかにできるのが普通であるため，設計者による指定で問題ないと判断した．ただし，使い勝手の観点から，指定項目をできるだけ少なくする方法を検討した．

その結果，付加的特徴機能の要・不要の指定は，実際の組み込み制御システムの開発上必要な指定のみとした．表 2 に示したように，これまで我々が扱ったシステムの中には，実装時にインスタンスの動的生成および動的束縛を必要とするものはなかった．そこで，組み込み制御システムではそれらの機能を実装する必要はないと判断し，ツールではインスタンスの動的生成および動的束縛のための機構はつねに生成しないものとした．

また，オブジェクトの一部のデータやメソッドのみで必要な機能については，それごとに要・不要の指定をするのは煩雑なため，できるだけツールで自動的に判定することとしたい．今回の場合，データ抽象がそれにあたる．前述のように，アクセスメソッドが必要なのはデータの正規化や単位変換の場合のみであった．そこで，ユーザはそれらが必要な場合のみ明示的にアクセスメソッドを定義し，そうでない場合は定義しないものとし，明示的にアクセスメソッドが定義されている場合はデータ抽象が必要であるが，そうでない場合は不要と見なす．

残ったクラスおよび継承の機能については，設計者が対象アプリケーションごとにそれら機能の要・不要を決定し，コード生成時にツールに指定することとする．

コード生成ツールは，上記により不要と判断された機能について，それらの実装機構を排除したコードを生成する．生成するコードは C 言語で記述したソースコードである．各機能の具体的な排除方法およびその効果については，4 章で詳細に説明する．

4. 不要機能の排除方法と効果

4.1 動的束縛の機構の排除

4.1.1 機構排除の方法

動的束縛は，オブジェクト指向が提供する多相性（Polymorphism）と呼ばれる概念を実現するための機構であり，実行時に各クラスの同じ名前のメソッドのうち，どのメソッドを呼び出すかを動的に決定するものである．C++言語では，仮想関数とその動的束縛によって多相性を実現している．ところが，選ぶメソッドが 1 つしかない場合や，呼び出されるクラスがあらかじめ分かっている場合には，仮想関数の機能を排除できる．

たとえば，あるクラスとそのサブクラスに属するインスタンスが 1 つしかないことがあらかじめ分かっていれば，呼び出すメソッドを探す必要はなく，直接そのメソッドにジャンプできる．また，具象クラスが 1 つのみであれば，仮想関数の機能および上位の継承階層に属する仮想関数を削除することができる．

例として**図 5** に示す，継承関係にあるクラス A, B を考える．詳細設計の結果，実装時には A のインスタンスがなく，B のインスタンスのみ必要と分かったとする．この場合には，A に属する仮想関数

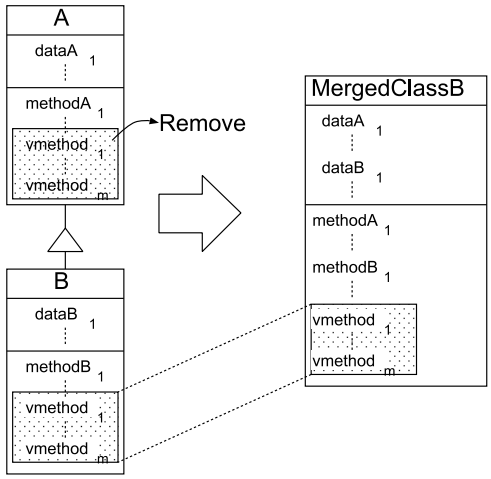

図 5 仮想関数の排除
Fig. 5 Removal of virtual functions.

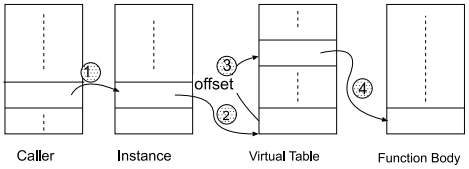

図 6 仮想関数の呼び出し手順
Fig. 6 Call of virtual functions.

A::vmethod$_1$ . . . A::vmethod$_m$ は不要である．また，この実行時の解決を行うルーチンも不要となる．またこれにともない，メソッド検索ルーチンを含む仮想関数呼び出しルーチンを，メソッド検索の不要な非仮想関数の呼び出しに変換することで，メソッド検索ルーチンを排除できる．

4.1.2 効　　果

本論文で評価に用いたコンパイラは，GNU gcc/g++ version 2.7.2（target: Hitachi H8-300 CPU）である．以下の評価結果は同コンパイラを用いた値である．

動的束縛は，このコンパイラでは仮想テーブルを用いることで実装される．その手順を**図 6** に示す．呼び出し側は，インスタンスに対して（①），仮想テーブルのアドレスを探し（②），仮想テーブルの先頭から呼び出しメソッドに該当するオフセットを加えてテーブルを検索し（③），そのテーブルに記載されているアドレスにジャンプする（④）．動的束縛が不要であれば，このような起動ルーチンを排除できる．

継承が存在する場合は，仮想テーブルのサイズは増大する．継承階層が r 階層で，仮想関数の数が v 個である場合に，排除される仮想テーブルのサイズは $6 \times v \times r$ Bytes となる．

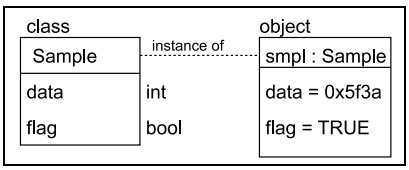

図 7 インスタンス定義の例
Fig. 7 Example of instance declaration.

```
struct Sample{
  int  data;
  bool flag;
}
struct Sample smpl = {0x5f3a, TRUE};
                        /* 静的に定義 */
int foo(void){
    ......
    var = smpl.data;
    ......
}
```

図 8 出力コードの例
Fig. 8 Example of generated code.

さらに，仮想関数 vmethod() の呼び出し部に 25〜30 Bytes 程度のメソッドの検索ルーチンが作られるが，これを非仮想関数の呼び出しに置き換えることにより，4〜8 Bytes になる．したがって，1 つの呼び出しあたり，コードサイズ約 20 Bytes を排除することができる．

4.2 インスタンス動的生成の機構の排除

4.2.1 機構排除の方法

一般にオブジェクトはクラスのインスタンスとして生成されるが，これを生成する際に構築子，消滅子が用いられる．これらは，構築および消滅を行うプログラムとして実装される．ところが，プログラム実行中に構築・消滅を行う必要がないものは，これらのルーチンは不要である．

前述のように今回のツールでは，動的生成は不要として，構築子，消滅子は生成しない．この場合，インスタンスは静的に定義する．入力仕様におけるその記述例を**図 7** に，出力コードの例を**図 8** に示す．構築子，消滅子を作らず，静的に初期化を行うことで，構築子，消滅子を排除できる．

4.2.2 効　　果

C++言語で実装した場合には複雑なデータ型を持たないクラスで，構築子，消滅子中で処理を行わず，値の初期化のみを行う場合，おおよそ，18 + (データメンバ数) × 8 Bytes 程度のサイズになり，これを排除できる．

4.3 データ抽象の機構の排除

4.3.1 機構排除の方法

前述のように，組み込み制御システムでアクセスメソッドが有効なのは，データの値の変換が必要な場合である．すなわち，内部で保持しているデータのサイズ・形式と参照する際に参照元に返すサイズ・形式とが異なる場合に，アクセスメソッド中で変換を行うことで，オブジェクト内部でのデータの実装を外部に対して隠蔽することができる．C++言語ではこれらの機能は，アクセスメソッドとなる関数を介することで実現される．

ところが組み込み制御では，前述のように制御仕様書によりデータのサイズや形式が指定されていることが多く，これらは 1 つの製品モデルを通じて一意に定まっているため，ほとんどの場合，アクセスメソッドによるデータサイズ・形式の変換は不要である．したがってこのような場合には，実装時には変数に対して直接アクセスしても問題はない．

本ツールでは，これをマクロ化することにより，関数としての実装を不要とし，コードサイズを削減する．すなわち，オブジェクトの主要なデータメンバの参照，変更を行うメソッド “Get” “Set” を特に次のように記述することで，静的に解決できる．

```
#define Obj_Get() (Obj_VALUE)
#define Obj_Set(arg) (Obj_VALUE = arg)
```

データ変換を行うアクセスメソッドが必要となる場合には，ユーザが明示的にそのアクセスメソッドを定義する．その場合ツールは，上記マクロを生成しない．

4.3.2 効　　果

アクセスメソッドをそのまま C++でメソッド化した場合は，コード部 14 Bytes，呼び出し部 6 Bytes となるのに対し，マクロ展開した場合には単なる変数へのアクセスに変換されるので，命令 2 Bytes 中のオペランド 1 Byte に出現するのみに抑えることができる．

メソッド本体部の削減分はメソッド 1 つあたり 14 Bytes であり，メソッド呼び出し 1 回あたりの削減分は 4–6 Bytes である．

4.4 継承の機構の排除

4.4.1 機構排除の方法

継承をしているクラス階層のうちで，実装する具象クラスが 1 つだけでよい場合には，実装時には継承階層を作成する必要がなく，これを排除できる．

具体的には，複数のクラスの継承階層を平坦な 1 つのクラスにまとめることで，仮想テーブルの占めるコードサイズを削減できる．**図 9** のようなクラス構成で，対象とするシステムの実装では，クラス A，クラス B のインスタンスがなく，クラス C のインスタンスのみがある場合に，親のクラスのメンバをすべて併合することにより，継承の機能を排除できる．

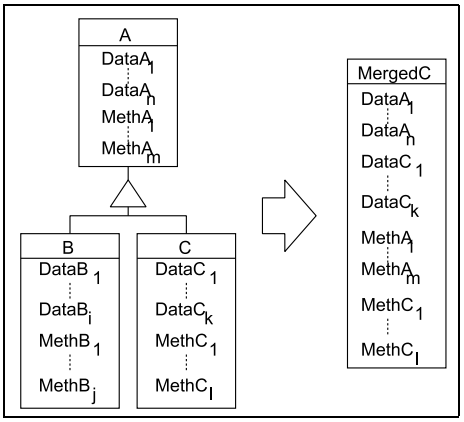

図 9 継承
Fig. 9 Inheritance.

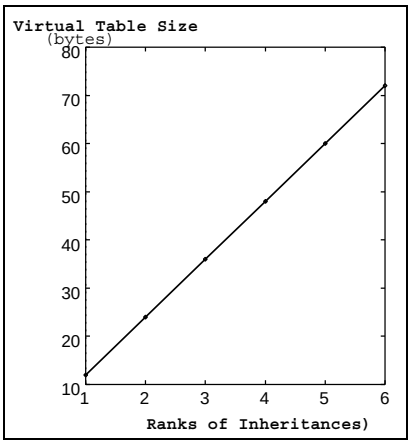

図 10 継承階層の深さと仮想テーブルのサイズ
Fig. 10 Virtual table size and inheritances.

4.4.2 効　　果

厳密には継承それ自体が新たな処理やデータを作ることはないが，継承を有効に使うためには，仮想関数，動的束縛の機能を使うことが多い．また，継承の階層が深くなると，これらの増加分が継承階層の数だけ倍増するため，結果として継承により上記の増加分が助長される．

たとえば，仮想関数と動的束縛を用いた場合に生じるメモリ容量の増加分と継承回数との関係は，**図 10** のとおりである．

継承機能を排除することによりこのメモリ容量の増加分を排除することができる．

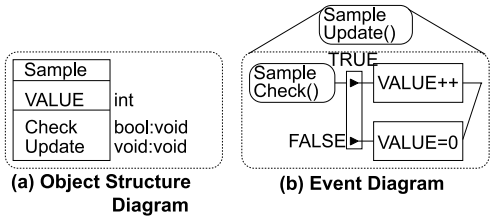

図 11 入力仕様書の例
Fig. 11 Example of input specification.

4.5 クラスの機構の排除

4.5.1 機構排除の方法

クラス機能が不要な場合は，メソッドの引数の実装を効率化できる．すなわち，各クラスのメソッドは実装上は，引数としてそのクラスのインスタンスへのポインタをとる．ところが，インスタンスが 1 つのみの場合は，インスタンスを固定できるので，引数での指定を排除できる．

クラス Cls のインスタンスを $\mathrm{Obj}_1, \mathrm{Obj}_2, \ldots \mathrm{Obj}_n$ とする．このとき Cls のメソッド Meth_i:Type → Type_{ret} の C 言語あるいは C++言語での実現を関数 $\mathtt{Func}_i$ とするとき，$\mathtt{Func}_i$ の実装は，

$\mathtt{Type}_{\mathrm{ret}}$ $\mathtt{Func}_i(\mathtt{Cls} * \mathtt{this},$ $\mathtt{Type\ arg})$

の型を持つ関数として実装される．また，メソッドの実行

$\mathrm{Obj}_j.\mathrm{Meth}_i(arg)$

を実現するコードは，

$\mathtt{Func}_i(\&\mathrm{Obj}_j, arg)$

と実装し，第 1 引数によりインスタンスを指定することで，複数のインスタンスに対して同一のメソッドを実行している．

ところが，クラス Cls のインスタンスが Obj_1 ただ 1 つに固定できると，Func の第 1 引数を省略することができ，その結果 Meth_i:Type → Type_{ret} を実現する C 言語の関数 $\mathtt{Func}_i$ は，

$\mathtt{Type}_{\mathrm{ret}}$ $\mathtt{Func}_i(\mathtt{Type\ arg})$

となり，その本体部の this の出現を $\&\mathrm{Obj}_1$ で置き換えた関数として記述できる．これによりインスタンスを引数で渡すためのコードを排除できる．

入力となるオブジェクト指向仕様書の例と，そのときの出力を示す．**図 11** は入力仕様書の例で，クラス Sample を定義するオブジェクト構成図 (a) とそのメソッド Update を定義するイベント図 (b) である．

図 11 (a)，(b) に対する通常の C++のコードは **図 12** (a) のようになる．これに対して，インスタンスの指定を排除した C のコードは図 12 (b) のようになる．

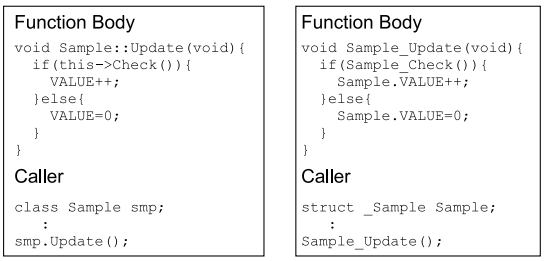

図 12 最適化前後の関数本体
Fig. 12 Code of the function body before and after optimization.

C++	C

`mov.w #smp, r2`
`mov.w r2, r0`
`jsr @A::Update(void)`
	`jsr @A_Update(void)`

図 13 最適化前後の関数呼び出し部
Fig. 13 Machine code of the caller before and after optimization.

4.5.2 効　　果

図 12 (a)，(b) をコンパイルし，コードサイズを比較すると，関数本体部はほぼ同サイズとなるが，呼び出し部は，**図 13** のようになり，1 呼び出しあたり約 6 Bytes ずつ排除できる．

5. 実際のシステムへの適用結果

以上提案した手法およびコード生成ツールを，自動車の組み込み制御ソフトウェアに適用し，C++でプログラムを作成した場合との比較を行った．

対象とした組み込み制御ソフトウェアの特徴は，以下のとおりである．

- 実装は ROM 上になされる．
- メモリ容量は，ROM のサイズが 62 KBytes，RAM のサイズが 2 KBytes である．
- 1 つのモデルでは，センサやアクチュエータ等のデバイスの構成が固定的である．

本アプリケーションでは，オブジェクト指向の 5 つの付加的特徴機能のいずれも必要ではない．まず，クラスは固定的なので，実装時に処理を決める動的束縛は必要はない．また，制御処理は静的に定まった個数のデータの値を更新することで行うので，実行時に新たなインスタンスが生成されることはなく，インスタンスの動的生成の機構は使う必要がない．また，アクセスメソッドによるデータサイズ・形式の変換は不要であったため，データ抽象は実装時には必要ではなかった．継承の概念は，設計の段階では用いるものの，実装時には，継承関係にある親，子のクラスを 1 つのク

表 3 対象アプリケーションのオブジェクト構成
Table 3 Objects of example application.

項目	個数
クラス数（分析時）	190 個
クラス数（実装時）	110 個
インスタンス数	110 個
メソッド数/1 クラス	15 個
データメンバ数/1 クラス	20 個

表 4 削減分の内訳
Table 4 Itemize of reduction.

要素	削減量 (%)
インスタンス渡しのメソッド	25%
仮想関数	25%
構築子・消滅子	17%
仮想関数呼び出し	14%
アクセスメソッド	12%
仮想テーブル	7%

ラスにまとめることができる．また，1 つのクラスに対して 1 つのインスタンスしか存在しないため，クラスの機能も排除できる．

本アプリケーションにおける，クラスおよびインスタンスの数，1 クラスあたりの平均メソッド数およびデータメンバ数を，**表 3** に示す．

本アプリケーションのオブジェクトの仕様を単純に C++言語に変換すると，全体のサイズは 90 Kbytes となった．これに対して，本研究のコード生成ツールを用いてコードを生成した場合には，60 Kbytes となり，コードサイズを 2/3 に削減できることを確認した．削減分の内訳（削減量全体を 100%とした場合の値）を**表 4** に示す．

以上のように，本論文で提案した手法およびコード生成ツールを用いることで，メモリ消費量の少ないプログラムを生成することが可能になる．

ただし現在は，本方式向けのデバッグ環境は特に用意してはおらず，生成した C ソースコード・レベルでデバッグする必要がある．ツールが生成するコードは，通常の C プログラミングにおける構造体および関数と同等のものであり，C 言語を知っていれば，デバッグそれ自体はそれほど難しくはない．しかし，プログラムを修正する場合には，オブジェクト指向仕様書を修正し，あらためてコードを生成する必要がある．オブジェクト指向仕様書レベルでデバッグができる環境を開発することが今後の課題である．

6. 関連研究との比較

これまで，オブジェクト指向プログラムの最適化に関する研究は数多くなされている．特に，クラス階層を分析することにより仮想関数の解決を行う方法や，総称的なアルゴリズムの詳細化等が多く行われている．

オブジェクト指向プログラムを静的に最適化する手法として，Dean らの方法[5] がある．これはクラスの階層を分析することにより最適化を行う方法である．また，Aigner らの方法[4] は，仮想関数の呼び出しを型の解析を行うことにより，最適化するものである．また，Srivastava の方法[6] では，到達不可能な関数をリンク時に削除することで不要なコードを取り除く．

これらいずれの手法も，ソースコードの情報をもとに，ソースコード全体にわたって分析した結果により最適化を行うものである．これに対し我々の手法は，ソースコードから情報を抽出するのではなく，設計時に得られた対象アプリケーションに関する情報を利用するという特徴がある．

また，プログラマが情報を与えることにより最適化を行うものとして，プログラム特化（Specialization）あるいは部分計算（Partial Evaluation）と呼ばれる手法がある．プログラム特化を行うことによって最適化を行うものに，Volanschi らの方法[16] がある．これは，テンプレートを用いた総称的（Generic）なプログラムに対して特定のパターンの場合のコードやデータ型を指定することにより主に実行速度を速めるための手法である．また，リソースの観点から，プログラム特化によるパフォーマンスの低下の指針を与えたものに，Debray の方法[17] がある．

これらの方法は，各メソッド等の実装方法をプログラマが積極的に記述する方法であり，ユーザにオブジェクト指向プログラムの実装に関する知識がある場合には，詳細な最適化ができ，有効な方法である．これに対し我々の手法は，アプリケーションの性質によって，オブジェクト指向の各機能を使用するかしないをユーザが指定することで，最適なコードを生成する．したがって，ユーザはオブジェクト指向プログラムの実装に関する知識を必要とせず，容易に最適化コードを生成できるという特徴がある．

7. むすび

組み込みシステムは年々規模が大きくなってきており，すでにオブジェクト指向化されているシステムもある．ただし，それらの多くは，比較的メモリ容量が大きな情報系の組み込みシステムである．メモリ容量が限られたワンチップ・マイクロコントローラによって実装される組み込み制御システムの場合は，オブジェクト指向言語導入によるコードサイズの増大のため，オブジェクト指向の導入が難しい．

本論文では，多くの組み込み制御システムにおいては，実装時に必要としないオブジェクト指向の付加的特徴的機能が存在することを明らかにし，それら不要な機能を排除することで，プログラムのコードサイズを削減する手法を提案した．そして，図式表現されたオブジェクト指向仕様書を入力し，不要機能を排除したC言語によるプログラムを出力する，コード生成ツールを開発した．これにより，メモリ容量の制約が厳しい組み込み制御システムへのオブジェクト指向の適用が容易になる．

今後の課題としては，オブジェクト指向仕様書のレベルでデバッグができる環境の開発や，モデル駆動開発におけるプラットフォーム依存モデルからのコード生成への適用等があげられる．

謝辞 本研究を支援していただいた，福永泰部長（当時），堀田多加志主任研究員（当時）をはじめとする（株）日立製作所日立研究所の諸氏に深謝する．

参 考 文 献

1) Douglass, B.P.: Designing Real-Time Systems With UML-PartI, *Embedded Systems Programming*, Vol.11, No.3, pp.46–64 (1998).
2) Plauger, P.J.: Embedded C++: An Overview, *Embedded Systems Programming*, Vol.10, No.12, pp.40–45 (1997).
3) Stroustrup, B.: *The C++ Programming Language, Third Edition*, Addison-Wesley, Reading (1997).
4) Aigner, G. and Hölzle, U.: Eliminating Virtual Function Calls in C++ Program, *Proc.ECOOP '96*, LNCS 1098, pp.142–166, Springer-Verlag (1996).
5) Dean, J., Grove, D. and Chambers, C.: Optimization of Object-Oriented Programs Using Static Class Hierarchy Analysis, *Proc. ECOOP '95*, LNCS 952, pp.77–101, Springer-Verlag (1995).
6) Srivastava, A.: Unreachable Procedures in Object-Oriented Programming, *ACM Letters on Programming Languages and Systems*, Vol.1, No.4, pp.355–364 (1992).
7) Ranbaugh, J., Booh G. and Jacobson, I.: *Unified Modeling Language Reference Manual*, Addison Wesley, Reading (1998).
8) 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-Verlag (1998).
9) Frankel, D.: *Model-Driven Architecture: Applying MDA to Enterprise Computing*, John Wiley & Sons, New York (2003).
10) Wegner, P.: Dimensions of Object-Based language Design, *Proc. OOPSLA '87*, pp.168–182 (1987).
11) 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, *Proc. 25th International Conference TOOLS Pacific*, pp.65–74 (1997).
12) 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).
13) 成沢文雄，本山敦久：組み込み制御向けオブジェクト指向設計手法とエレベータ制御への適用，第4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 (2002).
14) Martin, J.: *Principles of Object-Oriented Analysis and Design*, Prentice-Hal, Engelwood Cliffs (1993).
15) Douglass, B.P.: *Real-Time UML: Developing Efficient Objects for Embedded Systems*, Addison Wesley, Reading (1997).
16) Volanschi, E.N., Consel, C. and Muller, G.: Declarative Specialization of Object-Oriented Programs, *Proc. OOPSLA Conference '97*, pp.271–285 (1997).
17) Debray, S.: Resource-Bounded Partial Evaluation, *Proc. ACM '97 Partial Evaluation and Semantics-Based Program Manipulation*, pp.179–192 (1997).

(平成16年11月4日受付)
(平成17年 3 月1日採録)

成沢 文雄（正会員）

1970年生．1994年東北大学工学部情報工学科卒業．1996年同大学大学院情報科学研究科情報基礎科学専攻修士課程修了．同年（株）日立製作所日立研究所入社．ソフトウェア工学，ソフトウェア開発環境の研究開発に従事．ACM会員．

納谷 英光

1965 年生．1988 年北海道大学工学部電気工学科卒業．1990 年同大学大学院工学研究科電気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．画像処理，分散処理，ソフトウェア工学の研究開発に従事．電子情報通信学会会員．

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）日立製作所入社．1987 年から 1990 年まで（財）新世代コンピュータ技術開発機構出向．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．