社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－SLDM－124 (13)
2006／3／16

組込みシステムのシステムレベル設計における オブジェクト指向技術の応用

松井 健† 小松 聡†† 藤田 昌宏††

† 東京大学大学院工学系研究科電子工学専攻 〒113-0032 東京都文京区弥生 2-11-16
†† 東京大学大規模集積システム設計教育研究センター 〒113-0032 東京都文京区弥生 2-11-16
E-mail: †{matsui,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

あらまし オブジェクト指向技術は，設計の可読性と再利用性を向上させる技術として，ビジネスアプリケーション系ソフトウェア開発で広く利用されている．本論文では，ハードウェア設計を含む組込みシステムのシステムレベル設計に，ソフトウェア設計向け技術であるオブジェクト指向技術が応用可能かどうかを考察する．そしてその結果として，組込み設計向けに機能的構造と実装構造との対応付けを工夫することによって，オブジェクト指向分析とオブジェクト指向設計の概念が利用できることを示す．

キーワード 組込みシステム，オブジェクト指向技術，統一モデリング言語（UML），システムレベル設計

Application of Object-Oriented techniques to system-level design of embedded systems

Ken MATSUI†, Satoshi KOMATSU††, and Masahiro FUJITA††

† Department of Electronics Engineering, Graduate School of Engineering, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku, Tokyo, 113-0032 Japan
†† VLSI Design and Education Center, University of Tokyo
Yayoi 2-11-16, Bunkyo-ku, Tokyo, 113-0032 Japan
E-mail: †{matsui,komatsu}@cad.t.u-tokyo.ac.jp, ††fujita@ee.t.u-tokyo.ac.jp

Abstract Object-Oriented (OO) techniques have been widely adopted in business application software developments for better readability and reusability. In this paper, we consider whether OO techniques are also applicable for system-level design of embedded systems which involves hardware design. And in conclusion, the concepts of OO analysis and OO design are available if functional structures are mapped into implementation structures by a well-suited rule for embedded systems design.

Key words Embedded systems, Object-Oriented technique, Unified Modeling Language (UML), System-level design

1. はじめに

近年の半導体技術の進展に伴い，いかに効率よく組込みシステムを開発するかは大きな課題となっている．これに対する解決策として，従来は主として設計言語の高位化が進められてきた．ところが，設計の複雑化・大規模化により，外部仕様・内部仕様の策定や設計領域の切り分けなどといった人的問題が顕在化し，設計期間に大きな影響を及ぼすようになってきている [1]．そしてこうした問題について効率化を図る場合，言語の高位化だけでは対処できないことが普通である．

これに対し，オブジェクト指向技術を利用することによって問題の改善が期待できる，という議論がある．これは，オブジェクト指向技術に基づいて作られた設計が高い可読性・再利用性を持ち得るためである．しかしオブジェクト指向技術は，ビジネスアプリケーション系ソフトウェア開発のために作られたものであるから，それがハードウェア・ソフトウェア双方を含む組込みシステムの開発にも利用できるかどうか，評価しなくてはならない．そこで本論文では，特に組込みシステム設計の初期段階，一般にシステムレベル設計と呼ばれる段階に焦点を当てて，オブジェクト指向技術の適用可能性を考察する．そして考察結果に基づいた設計フローを提示し，ケーススタディを通してそれが実践可能であることを示す．

2. オブジェクト指向技術の成り立ちと特徴

2.1 「オブジェクト指向」とは

「オブジェクト指向」という言葉をタイトルに含む書籍は和書だけでも数百冊存在するが，それらの間で「オブジェクト指向」とは何か，ということに対しての定義がひとつに定まっておらず，ときには全く異なる意味づけを行っているのは興味深い事実である．こうした状況を好意的に見れば，「実際問題としてオブジェクト指向技術と呼ばれる技術が対象とする問題領域が極めて広いため一意に定義付けを行うことは難しいという事情がある」と察せられる．しかし，オブジェクト指向技術の利用を実際に考えるのならば，その意味するところを確実に把握しなければならない．そのためにはオブジェクト指向技術の成り立ちを追うのが最も適切である．そこで以下に，オブジェクト指向技術の主たる要素である OOP, OOA, OOD, UML について，それぞれの成立の経緯をまとめる．

2.2 オブジェクト指向プログラミング（OOP）

歴史的に見て，オブジェクト指向技術に関係する技術用語の中で最も早く使われはじめた単語は「クラス」である．「クラス」という言葉は，O.J.Dahl らの論文 [2] において，「サブルーチンと変数はまとめて扱うべき」という彼らの持つソフトウェアプログラミングに対する思想と，それを具現化した Simula67 というプログラミング言語のキーワードとして初めて使われた．

その後，Alan Kay により考案された Smalltalk [3] というプログラミング言語において，継承構造によるクラスライブラリが整備されたが，その最上位クラスの名前が「Object」であった．これは Smalltalk 上のクラスは全て「Object」クラスのサブクラスになることを意味する．ここから，「あらゆるものを『Object』から作る」という意味で，「Object Oriented = オブジェクト指向」という言葉と概念が作られた．つまり，「オブジェクト指向」という言葉はもともとプログラミングパラダイムと対応するものであり，具体的にはカプセル化 (encapsulation)，ポリモーフィズム (多態性，polymorphism)，継承 (inheritance) といった考え方に基づいてソフトウェアのコーディング効率を高めるための手法であったのである．

これを一般に「オブジェクト指向プログラミング（Object Oriented Programming: OOP）」と称しており，たとえば「C++や Java はオブジェクト指向言語だ」と述べる際の「オブジェクト指向」はこの意味で用いられている．近年では「デザインパターン [4]」として，再利用性に富んだコーディングのベストプラクティスが示されるなど，この流れを汲む技術はその後発展を続けている．また，「あらゆるものを『Object』から作る」という概念は，計算機上に仮想現実を作るという行為とよくなじむため，GUI としてウインドウや 3D 空間を持つようなシステム，たとえば Macintosh や Windows，Croquet [5] などの設計思想にも大きな影響を与えている．

2.3 オブジェクト指向分析（OOA）

一方，様々な問題や事柄をグループ分けする際の分割単位にも「クラス（=種類，集合）」という語が使われることからの連想で，「オブジェクト指向」という言葉をソフトウェアプログラミングに限らず，ビジネスモデリングなど問題分析一般に対して用いるという発想が生まれた [6] [7]．これを一般に「オブジェクト指向分析（Object Oriented Analysis: OOA）」と称し，「オブジェクト指向開発プロセス」「オブジェクト指向ビジネスモデリング」という表現がなされているものは，概ねこの意味で用いられている．この考え方は，大規模プロジェクトの問題分析と管理を行うためのプロジェクトマネジメント手法として OOP とは別に発展してきており，特に要求管理や品質管理の効率化に寄与するとされている．

2.4 オブジェクト指向設計（OOD）

さらに近年，主にビジネスアプリケーション系ソフトウェア開発において，OOA で作成した問題分析モデルを直接 OOP 言語における実装構造に対応づける，といったことが行われはじめた [8]．具体的には，ビジネス要件やソフトウェアの機能モデルをクラス構造として表記した上で，それをそのままプログラム上のクラス構造としてしまう手法である．これを「オブジェクト指向設計（Object Oriented Design: OOD）」と称する．一般に，こうした短絡的なモデルの対応付けは，実装構造の策定を省力化する反面，処理速度の面で非効率的な実装になることが多い．しかしハードウェアの性能向上に伴ってこうした性能面でのオーバーヘッドが相対的に無視できることが多くなったため，OOD も徐々に普及してきている．

2.5 統一モデリング言語（UML）

以上のように多様な側面を持つオブジェクト指向技術によって作られた成果物，すなわち分析設計モデルを表記するために，UML（Unified Modeling Language：統一モデリング言語 [9]）が作られた．

UML はその名称が表すように，既存のオブジェクト指向設計方法論で提案されたオブジェクト指向モデリング概念とその表記法（OMT 法 [6]，OOSE 法 [7]，Booch 法 [10] など）を統合・標準化したものである．1997 年に UML1.1 が OMG(Object Management Groupe) という標準化団体により認定され，2004 年には UML2.0 [11] の骨子が策定されて，現在詳細な仕様の検討が行われつつある．

ただし UML の規定時，方法論の統一には至らなかった．その結果として UML はあくまで単なる表記法にとどまり，どのような設計上の背景の中でこれを用いるかは定められなかった．おそらく，各方法論が異なる設計上の問題領域を扱っていたため，それらを一つにまとめようとしたとしても本質的困難が伴い，結果として方法論の統一が技術的に不可能だったのではないか，と考えられる．たとえば，Ivar Jacobson が提唱していた OOSE 法（Objectory 法）といった方法論では要求分析という多分に社会学的・認知学的な分野を扱うのに対し，James R Rumbaugh, Grady Booch らが提唱していた OMT 法，Booch 法といった方法論ではソフトウェア開発という工学的分野を対象にしており，その間に大きな隔たりがある，といったことである．

この結果，現在，UML の使われ方は様々な問題領域毎に無数に存在している．たとえばビジネスアプリケーション系ソフトウェア開発の分野では Rational Unified Process [12] やそれ

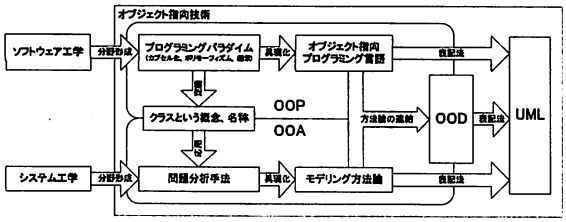

図 1　オブジェクト指向技術の成立関係

から派生した方法論などが，組込みソフトウェア開発の分野では eUML [13] などが提案されている．

以上，こうしたオブジェクト指向技術の成立関係を成立順にまとめると図 1 のようになる．

3. 組込みシステム設計に対するオブジェクト指向技術の適用可能性

3.1 組込みシステム設計の特徴

前述の通り，オブジェクト指向技術はビジネスアプリケーション系ソフトウェア開発のために作られたものであるから，これが組込みシステム一般に適用できる保証はない．そこで，組込みシステム設計特有の問題点を踏まえた上で，オブジェクト指向技術のどの要素がどのように活用できるかを考えなくてはならない．

そこで，まず組込みシステム設計の特徴を考える．これは，技術的側面とビジネス的側面に分類でき，以下のようにまとめられる [14].

技術的側面に関する特徴

- ソフトウェア開発だけでなく，専用ハードウェアの開発を伴うことが多い．
- スクラッチから開発することは稀で，過去の資産に対する修正と追加が多い．また，他者の設計資産（Intellectual Property:IP）を IP ベンダなどから購入して利用することもある．このとき，IP はブラックボックスとなっていて，その実装を知ることができない場合が多い．
- ハードウェアのレジスタを直接アクセスしたり，周辺装置とのインタフェイスを実時間制御したりすることが求められるため，使用するハードウェアを正確に理解・把握することが必要となる．
- 対象顧客が不特定多数であるがために，要求事項が曖昧で不確定要素が多い，要求事項が膨大になる，非機能的な側面の要求事項が曖昧，といった要求獲得上の困難がある．
- ほとんどの組込みシステムでは，ユーザがプログラムを入れ替えたり更新したりすることは想定されない．そのため汎用コンピュータよりも自由にシステム構成を選択できる．複数のプロセッサ要素（Processing Element : PE）を組み合わせてシステムを実現するアーキテクチャが一般的であり，この場合，共有メモリアーキテクチャ（メモリ階層の下，大容量のメインメモリが一つだけ存在する，というハードウェアアーキテクチャ）は前提とされない．
- ハードウェアが後から追加されたり，変更されたりすることは基本的に無い．

ビジネス的側面に関する特徴

- 市場競争の結果，開発期間の短期化が求められる．そのため，ソフトウェアの開発とハードウェア開発とが同時並行的に進められることが一般的である．
- 大量生産される製品の場合にはコストが非常に重要となるので，少ない容量のメモリと，安価なハードウェアでの実装が求められる．特に組込みソフトウェアのプログラムサイズはメモリのチップサイズに直結し，コストにそのまま反映されるため，ビジネスアプリケーションに比べ大きな制約がある．
- 多くの場合，ソフトウェアは ROM に書き込まれた状態で出荷されるため，出荷後にバグが発見されると，製品回収・ROM 交換作業などが必要になり，多大な費用がかかる．近年，ソフトウェアは ROM ではなくフラッシュメモリに格納され書き換え可能になったが，出荷後の修正が困難なことは変わっていない．したがって，非常に高度な品質が要求される．また当然ながら，ハードウェアについても同様のことが言える．

3.2 OOP，OOA と組込みシステム設計

第 2 節で示したオブジェクト指向技術の成立過程を踏まえた上で，まず OOP の基礎概念である「カプセル化，ポリモーフィズム，継承」が組込み開発にも適用できるかを考えると，これは問題なく可能である．なぜなら，これらの概念が組込みシステム設計の技術的側面に関する特徴によく合致するからである．すなわちカプセル化という概念は「プログラムからグローバル変数を廃し各クラスに対するアクセスを制限すること」を意味し，ポリモーフィズムという概念は「外部とのインタフェイスに対する内部実装のあり方をオブジェクト側で定義すること」を意味する．これらは共にハードウェア設計におけるブラックボックス化の概念に対応する．そして継承は「複数のクラスから変数とメソッドをまとめた共通クラスを作り，その定義を一括して行うこと」を意味し，ハードウェア設計におけるインタフェイスの仕様定義に対応する．

次に，OOP の概念を具現化したものであるオブジェクト指向プログラミング言語を組込み開発に用いることができるかを考えると，これは困難であると思われる．その理由として，オブジェクト指向プログラミング言語の動作の仕組みが共有メモリアーキテクチャを前提に作られていることが挙げられる．オブジェクト指向プログラミング言語の動作の特徴はメモリの使われ方にあり，その具体的な実装は各言語ごとに少しずつ異なる．しかし，いずれの実装においても

- クラス情報は共有メモリ中の静的領域に配置され，そこから生成されたインスタンスはヒープ領域に配置される（なお，このときのメモリの確保・解放により，性能面でのオーバーヘッドが伴う）．
- インスタンス変数にはインスタンスそのものではなく，インスタンスのヒープ領域中におけるポインタが格納される（C++では実値格納かポインタ格納かを選択できるが，メモリリークの原因となりやすいため，Java ではポインタ格納に統一されている）．

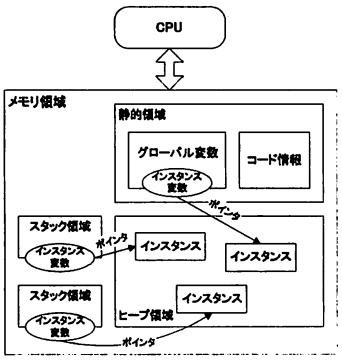

図 2　オブジェクト指向プログラミング言語におけるメモリの使われ方

という二つの点は共通しており（図 2），その実現のため大容量の共有メモリと，共有メモリを持つアーキテクチャに対応した中間マシンやコンパイラといったものを前提としているのである．

しかし組込みシステム設計では，前述のビジネス的側面に関する特徴から，十分なメモリ容量を確保することはコスト上難しい．また技術的側面においても，共有メモリを前提とするアーキテクチャは仮定できない．もちろんバスを介する形で共有メモリを導入することもできるが，各 PE 毎に個別のワーキングメモリを付与することもあり，どちらを選択するかは実装によって異なるため，システムレベル設計の段階ではどちらとも決められない．したがって，ビジネスアプリケーション系ソフトウェア開発用 OOP 言語とその仕組みをそのまま組込み開発に利用することは，基本的にできないと考えられる．

さて一方，クラスの概念に基づく問題分析手法やモデリング方法論（OOA）について見てみると，これらは問題なく利用可能である．ひとつには，これらの手法はシステムが外部に提供する振る舞いのみに注目し，内部の実装については詳細を考えない立場に立っているため，もともとハードウェアやソフトウェアに特化した方法論になっていないことが理由として挙げられる．また「クラス分け（＝分類）」という概念は，ソフトウェア開発だけでなく一般に広く適用できる概念であることから，ハードウェア設計への応用も十分可能であると考えられ，その点からも OOA がハードウェア/ソフトウェア双方を含む組込みシステム全体の設計に対して有効であると言うことができる．

3.3　OOD と組込みシステム設計

結論から述べると，OOD は組込みシステム設計用途には単純には利用しにくいと考えられる．

その理由は，先に述べた「組込みシステム設計の上位設計においては共有メモリの存在を仮定できない」ことに関連する．ビジネスアプリケーション系ソフトウェア開発では共有メモリの存在が仮定できるので，たとえば図 3 に示すように，大きなサイズのデータが複数の機能的オブジェクトを渡っていくというシーケンスを実装する際，オブジェクト構造はそのまま直接対応付け，オブジェクト間の通信形式だけをポインタ渡しにするようにすれば，比較的簡単に実際のオブジェクト間通信コストを抑えた実装を導くことができる．また実際に，そうしたオブジェクト構造と効率的な実装の対応付けが，コンパイラや中間マシンの形で自動化されてもいる．

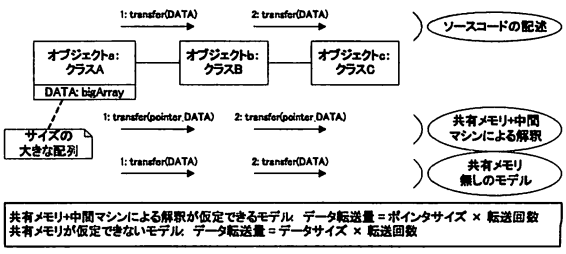

図 3　実装の違いによるデータ転送量の違い

しかし組込みシステム設計に同様の手法を適用しようとしても，共有メモリの存在を仮定できず，ポインタが使えないので，オブジェクト間のデータは値渡しにせざるを得ず，それをそのまま実装しても結果として通信コストが極めて大きくなってしまう．この問題は，通信コストに厳しい制約のあるハードウェア設計を伴う組込みシステム設計では致命的と言える．そしてまた，アーキテクチャ選択の自由度が高いために，汎用的なコンパイラや中間マシンのようなものを製作しようとしても，多大な工数が必要となってしまう．

したがって組込みシステム設計に OOD の概念を応用するには OOA のモデルをそのまま単純に実装構造に置き換えるということをせず，適切な実装構造と対応づけるような工夫が必要と考えられる．これに対するひとつの解となる設計方法論を第 4 節に示す．

3.4　組込みシステム設計に応用可能なオブジェクト指向技術

以上の議論をまとめると，オブジェクト指向技術と呼ばれる技術の中でも，組込みシステム設計に応用可能であるものとそうでないものがあることが分かる．すなわち，OOP の概念と OOA の概念・方法論は応用可能，一方既存の OOP 言語やその動作の仕組みは応用不可能である．また OOD の概念は，単純にビジネスアプリケーション系ソフトウェア開発の場合と同じ手法をとることはできず，機能モデルと実装構造モデルの対応付けに工夫をしてはじめて応用可能となる．

4.　OOA と OOD を取り入れたシステムレベル設計フローとケーススタディ

4.1　システムレベル設計フローの提案

組込みシステムが対象とする製品種別は多岐に渡るため，それら全てに対し汎用的に用い得る設計方法論を定めることは困難である．そこで具体的方法論を考察する際にはその分類を行う必要がある．本論文では，組込みシステムと称される製品を以下の三種類に分類する．

- 一般消費者向け製品（例：デジタルカメラ）
- 一般消費者向け製品の実装のために使われる部品（例：画像処理 SoC）
- さらにそうした部品の実装に使われる粒度の細かい部品

(例：jpeg 符号化回路)

このうち，二番目の項目に属する製品のシステムレベル設計について OOA と OOD を応用し，また設計表記に UML を利用する方法として，図 4 のような設計フローを提案する．この設計フローではスパイラル型開発モデルを前提としており，各設計段階で UML として表現された設計上の意思決定過程とその結果は再利用のためにデザインライブラリに登録される．機能面に関する分析設計結果はクラスライブラリに，実装面に関する分析設計結果は PE ライブラリに登録される．

この設計フローの入力としては製品コンセプトとその外部仕様を自然言語で記述したものを，出力としては SpecC [15] コードと設計詳細化のために必要な情報を UML として表記したものを考える．SpecC はシステムレベル設計言語の一つであり，ハードウェアとソフトウェア双方を含んだシステムを一つの言語として表せ，またチャネルによる機能と通信の分離を行う設計モデルを持っている．この設計モデルは OOA, OOD の概念とよく馴染むため，本設計フローにより作られるオブジェクト指向設計モデルのさらなる詳細化に適していると考え，設計フローの出力として設定した．

提案する設計フローは大きく二つの段階に分けられる．ひとつは以後「分析段階」と呼ぶ段階で，

- 与えられたシステムの外部仕様に関する要求仕様から機能的構造を作成し，
- またその構造の中でのシステムの振る舞い方を決定する

段階である．この段階においては特に OOA を中心としたオブジェクト指向技術を活用する．前節で指摘したように，OOP 言語を利用したビジネスアプリケーション系のソフトウェア開発であれば，この分析段階を終えた後，OOD の概念に沿って得られた機能的構造を元に直接コーディングをはじめることができる（「実装段階」）．しかし，組込み機器開発の場合はそれができないため，提案手法では分析段階と実装段階を結ぶために，「設計段階」を導入している．設計段階では，

- 分析段階で定めたシステムの機能的構造に対して物理的アーキテクチャのテンプレートを割り付け，
- テンプレートに対して PE ライブラリ中にある物理的部品を割り当て，
- その上でシステムの振る舞いを実現するために必要とされる各物理的部品間の通信量や処理速度を見積もって作成したアーキテクチャの評価を行う

といったことを行う．このとき，ビジネスアプリケーション系ソフトウェア開発のように，その実装手段として OOP 言語を使うということは前提としない点に注意する．

4.2 ケーススタディ

図 4 に示した設計フローが実践可能であることを示すため，ケーススタディとしてデジタルカメラなどに多用されるコンパクトフラッシュメモリインタフェイス（CF-IF）の分析設計を行った．なお，この分析設計にあたっては，実際の製品設計者からレビューを通した助言を受けることができた．

その結果を表 1 にまとめて示す．この表は，提案手法における各工程について，それぞれ SpecC を用いた最終的な機能記

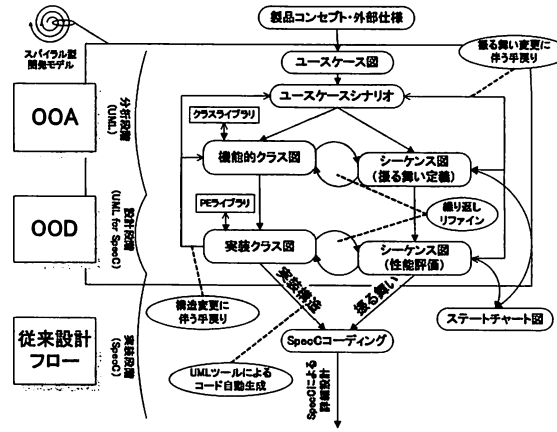

図 4　提案する設計方法論の設計フロー

述に結びついた UML 図の数と，設計案として検討されたものの設計上の決定に伴い最終的な機能記述には反映されなかった，いわば設計の代替案となった UML 図の数を，それらの作成にかかった時間とあわせて示したものである．

この結果の中で，OOA と OOD の概念を実現する際特に重要となる機能構造と実装構造の対応付けについて注目すると，関連する図として図 5，図 6，図 7 が挙げられる．

機能クラス図（図 5）は OOA による機能分析の結果を反映したものである．この構造を直接実装構造に対応づけられないのは前述の通りである．そこで，CF-IF のようなインタフェイス回路の実装構造テンプレートとして，「外部との IF+バッファ+コントローラ」という構造を用意し，これに機能クラス図中のクラスを割り付けて図 6 を導いた．その上で各機能クラスを実現するための，粒度の細かい部品（PE）をライブラリから選択して実装クラス図（図 7）を導いた．この図ではハードウェアコンポーネントの切り分けが完了し，またその中の機能構造もオブジェクト指向分析に基づき適切に分割・定義されているため，これをもとにして SpecC の設計フローに沿った設計を開始することができる．

一方，本論文で示したフローに従った CF-IF のオブジェクト指向分析設計には 3 人月（ただし，1 人で作業）を要した．過去同様の製品を開発した技術者によれば，従来の一般的な設計フローに沿って自然言語で与えられた仕様書から合成可能な RTL 記述を完成させるまで，12 人月（ただし複数人で作業）であったとのことである．このうちどれだけの工数が本論文での分析設計経過と対応するか，残念ながら正確な記録はない．しかし，「仕様策定など計画段階にかかる工数は，それ以後のコーディングからテストにかかる工数の 1/2」という経験則 [1] によれば，4 人月程度と推測できるものと推測される．従来手法での工数は複数人が開発に携わった場合の工数であり，そこには開発関係者間でのヒューマンコミュニケーションコストが加わっていることを考慮すると，提案手法の工数は従来手法のそれとほぼ同じ程度である．このことから，工数の面でオブジェクト指向技術を導入する不利益はさほど大きくないと考えられる．

表 1　作成した UML の数と要した工数

	設計に利用	代替設計として保存	計	作業日数割合
ユースケース図	5	2	7	9%
機能的クラス図	3	20	23	23%
振舞い定義用シーケンス図	27	26	53	21%
振舞い確認用ステートチャート図	6	9	15	11%
実装クラス図	11	52	63	22%
性能評価用シーケンス図	11	14	25	14%
計	63	123	186	3 人月 (100%)
導出された SpecC コード（ビヘイビア構造+メッセージ送受信順が定義済み）: 552 行				

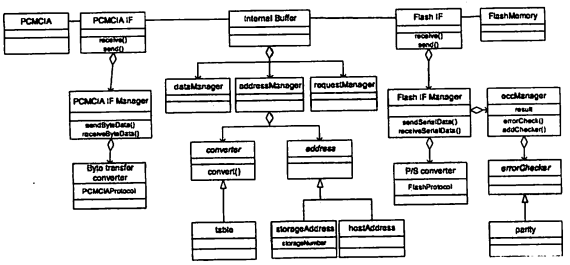

図 5　CF-IF の機能的クラス図

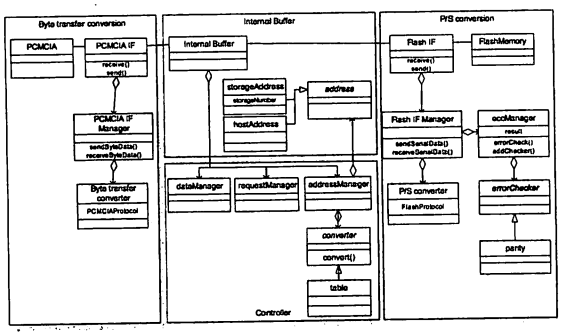

図 6　実装テンプレートに沿った機能構造と実装構造の対応づけ

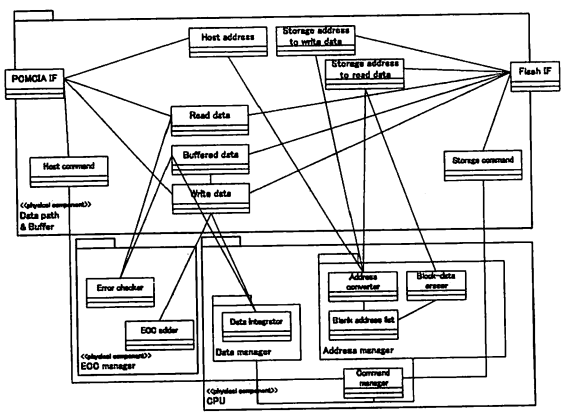

図 7　CF-IF の実装クラス図

5. おわりに

本論文では，ハードウェア設計を含む組込みシステムのシステムレベル設計に，ソフトウェア設計向け技術であるオブジェクト指向技術が応用可能かどうかを考察した．その結果として OOA と OOD の概念が利用できることがわかったが，ビジネスアプリケーション系ソフトウェア開発の場合と同様に OOA による機能的モデルを直接実装構造に置き換えることはできず，組込みシステム設計の特徴に沿って機能的モデルと実装構造との対応付けを工夫しなければならないことも明らかになった．そこで，ケーススタディを通してそのための方法を探り，実装テンプレートを導入することで従来の OOD のように機能構造と実装構造を直接対応づける場合と比べ，比較的効率的で無理のない実装を導けることを示した．

文　献

[1] Frederick P. Brooks, Jr., " The Mythical Man-Month: Essays on Software Engineering, 20th Anniversary Edition, " Addison-Wesley Professional, 1995.

[2] Dahl, O., et al., "Simula 67: Common Base Language," Norwegian Computing Center, 1967.

[3] A. Goldberg, A. Kay, "Smalltalk 72 Instruction Manual," Xerox PARC, 1976.

[4] E. Gamma, et al., "Design Patterns," Addison-Wesley, Reading, MA, 1995.

[5] DA Smith, et al., "Croquet - A Collaboration System Architecture, " Proceedings of the First Conference on Creating, Connecting, 2003.

[6] James R Rumbaugh, et al., "Object-Oriented Modeling and Design," Printice Hall.

[7] Ivar Jacobson, "Object-Oriented Software Engineering," Addison-Wesley Publishing Company.

[8] John D. Poole, "Model-Driven Architecture: Vision, Standards And Emerging Technologies," JWorkshop on Metamodeling and Adaptive Object Models, ECOOP, 2001.

[9] Grady Booch, "The Unified Modeling Language User Guide, " Addison-Wesley Publishing, 1998.

[10] Grady Booch, "Object-Oriented Analysis and Design with Applications (2nd Edition)," Benjamin Cummings Publishing Company, 1993.

[11] "The current UML specification, Version 2.0," http://www.omg.org/uml/

[12] Ivar Jacobson, et al., "UML による統一ソフトウェア開発プロセス—オブジェクト指向開発方法論 ," 翔泳社, 2000.

[13] 渡辺 博之 他, "組込み UML," 翔泳社, 2002.

[14] 経済産業省, "組込みソフトウェアの開発力向上に向けた施策と提言," 2004 年 6 月.

[15] D. Gajski, et al., "SpecC: Specification Language and Methodology," Kluwer Academic Publishers, 2000.