C-007

抽象仕様の再構成に基づくオブジェクト指向システムレベル設計
Object-Oriented System Level Design with Reformation of Abstract Specification

山崎 亮介† 吉田 紀彦† 楢崎 修二‡
Ryosuke Yamasaki Norihiko Yoshida Shuji Narazaki

１．はじめに

システムレベル設計とは，SOC(System-On-a-Chip)設計のための設計手法である．SOC はシステムを構成するソフトウェア(以下 SW)とハードウェア(以下 HW)を１つのチップ上に実現しているものである．この設計手法では，設計仕様から動作レベル，通信レベルへと SW や HW の区別なく段階的に詳細化しながら設計を進め，SW 実装と HW 実装を選択し，それぞれをさらに詳細化する．最後に SW と HW を統合しシステムを実現する．記述にはシステムレベル設計言語が用いられており，これら設計手法や設計言語の登場により設計生産性は格段に向上している．最近では更なる設計生産性の向上を図るためにオブジェクト指向を導入する動きもあり，最上位である設計仕様には UML(Unified Modeling Language)が用いられている．ところが，この UML とシステムレベル設計言語との間には大きな抽象度の差があるため，スムーズな段階的詳細化が困難である．

この抽象度の差は，設計制約の多さによるものであるため，段階的に制約を課しながら詳細化する設計手法が適当だと考える．そこで，本研究では次のような手法を提案する．まず設計仕様を基に実行可能仕様を作成する．その後，実行可能仕様を構成する各オブジェクトからメソッドを分離・独立させて単機能オブジェクトとし，このオブジェクトに対して接続や通信の詳細化をおこなう．さらに，変数とメソッドに注目することでオブジェクト間の通信形態，接続形態といったアーキテクチャ選択への指針を示す．例題としてインターネットルータを設計した．本手法を適用することで UML からシステムレベル設計言語の抽象度まで，段階的に制約を課しながら詳細化する．他に，オブジェクト指向を適用する際の問題点について考察する．

２．システムレベル設計とオブジェクト指向

システム設計の設計生産性を高めるためには，設計手法や設計言語の抽象度を上げること，過去の設計資産を再利用することが適当である[1]．オブジェクト指向はこの２点を満たすことができると考えている．オブジェクト指向ではインターフェース接続やポリモーフィックなどによりオブジェクト間の接続を抽象化して記述できるため，オブジェクトの接続は動的に変更可能である．また，インターフェースが同じオブジェクトは交換可能であり，設計仕様が変更された場合も適当なオブジェクトへ交換するだけで対応できる．

ところが HW の場合，接続が動的に変更されることはない．また，ポリモーフィックのように使われるかどうか判らない機能を持っていると，消費電力の増大を招く恐れがあると考えられる．さらに，HW は動的に生成されることがない．このように，HW を記述するためにはオブジェクト指向の抽象度を HW 記述が可能な抽象度まで下げなければならない．現在提案されているシステムレベル設計言語は HW 記述を意識しているため，UML や UML を基に作成した実行可能仕様との抽象度の差が大きい[2].

表１ SW オブジェクトと HW オブジェクトの比較

	粒度・機能	通信	通信プロトコル	入出力・接続	オブジェクトの動的生成
SW オブジェクト	大・多機能	物理的実体なし	メッセージの送受信	非固定・動的	可
HW オブジェクト	小・単機能	物理的実体あり	タイミングなど	固定・静的	不可

SW オブジェクトは実行可能仕様を構成しており，オブジェクト指向のオブジェクトに相当する．この SW オブジェクトは設計仕様を満たすための必須機能の他にも多くの機能を備えており，そのまま HW 化すると設計面積や消費電力などの設計制約を満たすことが困難になる．これに対し，本論文で定義する HW オブジェクトは設計仕様を満たすための必須機能のみを備えているものとする．

SW オブジェクトと HW オブジェクトの比較を表１に示す．入出力・接続に注目すると，SW オブジェクトは入出力が非固定であり接続を動的に変更できる．一方，HW オブジェクトは入出力が固定であり接続は静的に決められていなければならない．HW オブジェクトを設計するためにはこれらの制約を常に考慮していなければならず，設計制約の少ない設計仕様から HW オブジェクトを記述することは非常に困難である．

アーキテクチャ探索はオブジェクトの構成やオブジェクト間の接続形態，通信形態などをもとに行われるが，SW オブジェクトはその構成や接続形態，通信形態が曖昧であるため，アーキテクチャ探索が困難である．したがって，SW オブジェクトの抽象度を HW オブジェクトの抽象度まで下げる方法が必要である．そこで本研究では，抽象度を下げる方法を提案する(図１).

３．提案手法

SW オブジェクトと HW オブジェクトの抽象度の違いについていくつかの項目を挙げたが，各項目ついて，段階的に詳細化し設計制約を満たすよう SW オブジェクトの構成や接続などを変更しなければならない．SW オブジェクトは多くのメソッドを備えており，またそれらのメソッド間の入出力や接続には多くの種類があるため，SW オブジェクトをそのまま HW オブジェクトとして扱うのは難しい．そこでまず，SW オブジェクトを構成しているメソッドを SW オブジェクトから分離・独立させ，メソッドをオブジェクトとすることを考える．この操作により，単機能オブジェクトとすることができるので接続や通信プロトコルなどの詳細化が容易になると期待できる．メソッドの分離・独立の具体的な方法を以下に説明する．

SW オブジェクトを構成するメソッドは，値を返す場合，メソッド自身が何に対して出力しているのかが明確でない．そこで出力先をメソッドの引数として受け取るようにする．この操作により各メソッドの接続形態の曖昧さを除くこと

†埼玉大学 Saitama University
‡長崎大学 Nagasaki University

ができる．メソッドの接続形態が明確になったことで，メソッドをオブジェクトとして SW オブジェクトから独立させ，これを HW オブジェクトとする．

次に，HW オブジェクト間の接続に使われる変数を，書き込み，読み出しメソッドを備える変数オブジェクトとする．HW オブジェクトは，自身に接続されている変数オブジェクトへの書き込み，読み出しメソッドを使うことで変数にアクセスするため，HW オブジェクトは変数オブジェクトを介することでのみ通信することができる．

図２に示すように，以上の操作を繰り返し行うことで，設計仕様を満たすために使用されるメソッドと変数のみをオブジェクトとして取り出すことができる．図２のオブジェクト関係図を図３に示す．

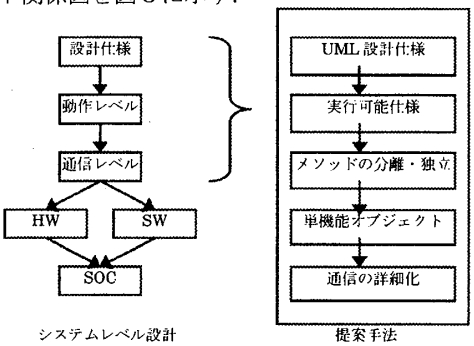

図１　システムレベル設計における提案手法の位置付け

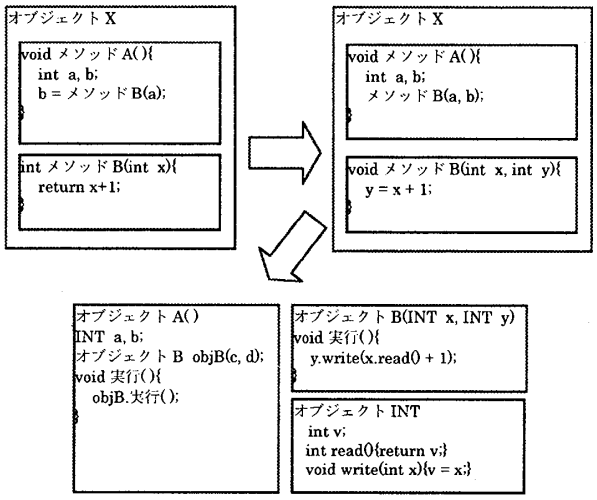

図２　メソッドの分離・独立

HW オブジェクトは並列に動作可能であるため，データの送受信には特に注意しなければならない場合がある．共有変数に対しての書き込み，読み出しの順序が正しくないと，干渉が起こりシステムの一貫性が崩れたり，デッドロックを起こしたりすることもある．これらの問題はセマフォやモニタなどの並行プログラミングの技術を適用することで対応する．システムレベル設計言語ではチャネルと呼ばれる機構を導入している．チャネルには変数に対する通信プロトコルを記述できる．並列動作する HW オブジェクトの通信同期や，待ち合わせ同期はチャネルを用いて実現する．設計仕様の変更などで変数との接続やアクセスに影響を及ぼす場合は，変数と通信プロトコルを１つにまとめチャネルとして実装する．

HW オブジェクトは変数オブジェクトを介して通信を行なうので，変数オブジェクトに注目することで HW オブジェクト間の接続やデータの流れる向きを知ることができる．これらを基にして，アーキテクチャ探索の指針を示すことができると考える．

まず，通信するデータに注目する．通信するデータには２つの種類がある．１つは基本データ型と呼ばれる int や double，char などである．もう１つは，いくつかの基本データ型をまとめて１つのオブジェクトとする抽象データ型である．抽象データ型をやり取りする場合，仲介する変数は１つであるため，シリアル通信とする(図４左上)．また，基本データ型を複数やり取りする場合，その数だけ仲介する変数があるのでパラレル通信とする(図４右上)．処理速度や設計面積などの設計仕様を満たすために，設計者は接続形態の変更が可能である．処理速度を速くするためには抽象データ型を展開することでパラレル接続とする．設計面積に余裕がなければ基本データ型を抽象データ型に集約することでシリアル接続やバス接続とする．設計仕様の変更により HW オブジェクト間の接続や通信プロトコルの交換・修正が必要な場合はチャネルの交換のみで対応する．

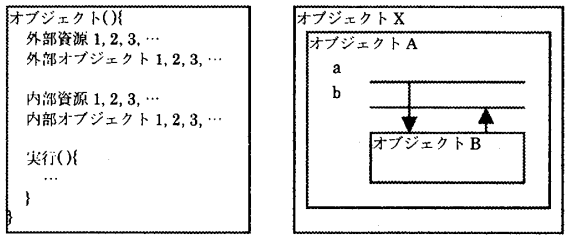

図３　ハードウェアオブジェクトの一般形（左）と図２のオブジェクト関係図（右）

HW オブジェクト間の接続に注目する．ある変数に対して読み出しと書き込みが１つずつある場合，変数の両端に接続されている HW オブジェクトの接続と通信はパイプのアーキテクチャが適当である．読み出しと書き込みが複数ある場合，その変数オブジェクトはレジスタファイル，共有メモリとしての実装が適当であり，アクセスはバスを介することになる(図４右下)．バスは複数のオブジェクトと接続されているため，同時にアクセスされる可能性がある．したがって，このような変数やバス，共有資源へのアクセスにはチャネルを用いる．

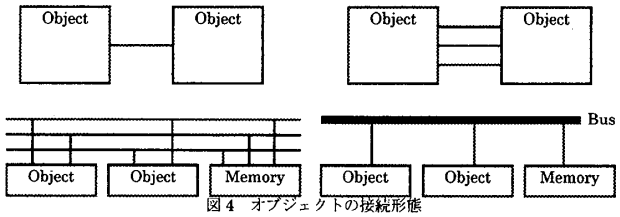

図４　オブジェクトの接続形態

本研究が提案する手法のように，システム全体の一貫性を保ったまま，正しく動作しているオブジェクトの内部構造やオブジェクト間の接続を変更することをリファクタリングという[3]．リファクタリングには目的に応じて様々なテクニックがある．例えば，複雑に絡み合ったオブジェクトの関係を簡潔にするためにメソッドをオブジェクトとして独立させる方法，オブジェクトの再利用性を上げ保守管理を簡単にするためにデザインパターン[4]を導入する方法などがある．すでにリファクタリングは多くの場面で実践

されており，大きな成果をあげている．これらのリファクタリングはSWオブジェクトからSWオブジェクトへ向けてのものであるため，段階的に設計制約を課しながら抽象度を下げ，詳細化するといったものではない．それに対し，本手法はHWオブジェクトへ向けてのリファクタリングである．オブジェクトを単機能とすることで，通信や接続の再構成を容易にすることで，アーキテクチャ探索に柔軟性をもたらすと期待できる．

４．例題：ルータの設計

本手法の適用例として，インターネットルータを設計した．ルータは計算機上の仮想的なものである．作成したルータは距離ベクトル型，リンク状態型，パスベクトル型の3つである．各ルータは経路表，経路探索，通信の3つのオブジェクトから構成される(図５)．経路探索オブジェクトではそれぞれのルータに必要なアルゴリズムを実装した．その他に，例えばLSDB(Link State Data Base)オブジェクトなど，各ルータの経路探索に必要な経路情報データベースオブジェクトも作成した．設計の流れは図１に従う．

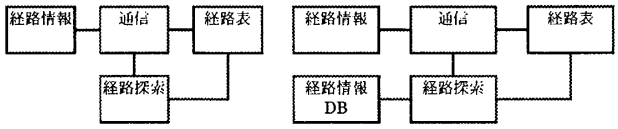

図５　設計仕様のクラス図
(左:距離ベクトル型　右:リンク状態型，パスベクトル型)

まず始めに，HWを意識することなく実行可能仕様を作成した．今回はJava言語を用いて実装を進めた．Java言語を用いた理由は，並行プログラミングが可能であることと，オブジェクト指向言語であることの２点である．特にJava言語でなければならないといったことはない．Java言語による実装の後，設計仕様を満足しているかどうかシミュレーションを行い検証した．

次に，本手法を用いて単機能オブジェクトの粒度まで細かくした．この段階では特に，抽象データ型でないオブジェクトを動的に生成していないかどうかに注意する必要がある．また，抽象データ型を動的に生成している場合，その内部にある属性に対して直接データを書き込むようにした．その後，設計仕様を満足しているかどうかシミュレーションを行い検証した．

最後に，変数オブジェクトへの接続やアクセスに注目して，複数のアクセスが同時に起こりうる変数オブジェクトをチャネル化した．このチャネルの同期機構には並行プログラミングのためのデザインパターンであるReadWriteLockパターンを用いた[5]．この同期機構は変数だけでなく，バスやレジスタファイル，共有メモリなどとして実装する変数には必ず必要となる．

本手法を適用した後のオブジェクトについて表２にまとめる．本手法を適用した後のHWオブジェクトについては，ほとんど再利用ができなかった．これはオブジェクトの入出力や接続，通信形態が固定化されているためと考えている．分離・独立したオブジェクトの数は，設計仕様や実行可能仕様の記述に強く依存するため，設計者が異なるとHWオブジェクトの数も違うと思われる．設計仕様から作成した実行可能仕様の抽象度では多態性やインターフェース接続などにより他のルータ設計で作成したSWオブジェクトの再利用や交換は容易であった．

表２　経路探索オブジェクトと変数のチャネル化

	経路探索オブジェクト		変数のチャネル化
	適用後の数	再利用できた数	
距離ベクトル型	16	2	24
リンク状態型	10	1	12
パスベクトル型	7	0	17

現在は，システムレベル設計言語への書き換えを進めている．対象としているシステムレベル設計言語は，Java言語ベースのPtolemy II [6]とC言語ベースのSpecC[7]である．これら言語への書き換えについて，一部の機能は非常に容易であった．単機能オブジェクトの粒度であるHWオブジェクトの形は，SpecCのbehaviorやPtolemy IIのアクターとよく似ているため，書き換え方次第でさらに容易になると思われる．また，書き換えが終わったHWオブジェクトごとの検証もスムーズに進んでいる．しかし，抽象データ型ではないオブジェクト，例えばIteratorのようなデータ構造とそれに対する操作を持つSWオブジェクトの受け渡しができないため，上記のシステムレベル設計言語への書き換えには特別な方法が必要であるが，現段階では具体的な方法は考えていない．

５．おわりに

距離ベクトル型，リンク状態型，パスベクトル型の順に設計を進めたが，経路表オブジェクトなど一部のオブジェクトは一切の修正・変更なしに再利用可能であった．また，本手法を適用した後の変数オブジェクトとHWオブジェクトの接続形態，通信プロトコルも容易に変更することができ，本手法により設計仕様を段階的に詳細化することができた．ただし，HWオブジェクトの交換や再利用については接続が固定化されているため困難であった．

今後の課題として，メソッドの分離・独立の自動化やプロファイラを使った通信の負荷分散，関連するHWオブジェクトの集約などを考えなければならない．

参考文献

[1]Daniel D Gajski, Jianwen Zhu, Rainer Dömer, Andreas Gerstlauer, Shuqing Zhao/木下常雄,冨山宏之訳.『SpecC仕様記述言語と方法論』CQ出版社,2000

[2]山崎亮介.“オブジェクト指向システムレベル設計におけるインターフェース設計と部品化”.長崎大学大学院修士論文,2004

[3]Martin Fowler/児玉公信,友野昌夫,平澤章,梅澤真史訳.『リファクタリング　プログラミングの体質改善テクニック』ピアソンエデュケーション,2000

[4]結城浩.『Java言語で学ぶデザインパターン入門』ソフトバンクパブリッシング,2001

[5]Mark Grand/原潔,宮本道夫,瀬尾明志訳.『UMLを使ったJavaデザインパターン-再利用可能なプログラミング設計集』カットシステム,2000

[6]Ptolemy Project:http://ptolemy.eecs.berkeley.edu/

[7]SpecC:http://www.ics.uci.edu/~ specc/