LC-006

抽象的システムレベル設計へのリファクタリング技術の適用
Application of Refactoring Techniques to Abstract System Level Design

山崎 亮介* 小林 憲貴* 楢崎 修二† 吉田 紀彦*
Ryosuke YAMASAKI Kazutaka KOBAYASHI Shuji NARAZAKI Norihiko YOSHIDA

1 まえがき

システムレベル設計において，更なる設計生産性の向上を目指し，オブジェクト指向技術をシステムレベル設計へ応用する研究が進められている．それらは主に，UMLを用いてシステム仕様を記述し，SystemC，SpecCといったシステムレベル設計言語を用いて実装を進める[1][2]．システムレベル設計言語を用いることで，モジュール（ビヘイビア）の階層や接続を明確にすることができる．これは，最上位の抽象度において，すでにハードウェア／ソフトウェア（以降，HW／SWと略記する）のモデルや分割を考慮しているといえる．つまり，UMLによるシステム仕様を記述する際には，モジュール（ビヘイビア）の階層や接続を考慮することになる．

そこで，本稿は，HW／SWのモデルや分割を考慮しない，より高い抽象度における設計手法を提案する．提案手法では，システム仕様記述の次に，オブジェクト指向プログラミング言語による実行可能仕様を記述する．その後，この実行可能仕様を再構成することでシステムレベル設計言語による時間の概念がない抽象度の実行可能仕様を得る．このようにすることで，HW／SWのモデルや分割を考慮するための記述の制約なしに，継承や多様型の機構から得られる設計生産性の向上が期待できる．

実行可能仕様の再構成はソフトウェア工学の分野で発展してきたリファクタリング技術[3]に基づく．リファクタリング技術を用いることにより，再構成の前後でシステムの挙動を保存することができるので，実行可能仕様を安全に再構成することが可能である．

本稿の構成は以下のとおりである．2で，関連研究について述べる．3で，提案手法について述べる．4で，提案手法による例題設計について述べ，結果を考察する．5で，本稿をまとめ，今後の課題および展望を述べる．

2 システム再構成技術

正しく動作するシステムの挙動を変えないシステム再構成の方法は大きく2つある．1つは理論的な方法であるプログラム変換技術[4]，もう1つは実践的な方法であるリファクタリング技術である．

2.1 プログラム変換技術

プログラム変換とは，あるプログラムの意味（挙動）を保存しつつ，一定の規則に則って別のプログラムに書き換える手法である．プログラムの意味（挙動）と同時に，表現形式も保存されるので，コンパイルをプログラム変換としてとらえることはほとんどない．理論的であるので，変換の前後でシステムの挙動が保存されていることを完全に保証する．

この技術はプログラムの導出，効率化，特殊化などに向けて研究が進んでいる．システムレベル設計に関連する応用として，ハードウェアアルゴリズムの導出がある[5]．これは理論的アプローチであるがゆえに実用規模の応用は難しい．

2.2 リファクタリング技術

リファクタリングとは，「システムの外部振舞いを保存しつつ，部品化，再利用が容易になるようシステム内部を変化させること」である．部品化や再利用を妨げる様々な状況ごとに，それらを改善するための技法がある．これら技法はある程度体系的にまとめられており，目的に応じて技法を単独で，あるいは組み合わせて適用する．

それぞれの技法は局所的なプログラム変換とその手順からなる．これらのほとんどは経験則に基づいているので，リファクタリングの前後でシステムの挙動が保存されていることを完全には保証していない．

リファクタリングを適用する際に遵守すべき2つの原則がある．1つは，細かく決められた手順に必ず従わなければならないということである．もう1つは，リファクタリング後にコンパイルとテストを必ず行い，システムの挙動が保存されていることを確かめなければならないということである．これら2つの原則を遵守することで，完全ではないがシステムの挙動を保存しつつシステム内部を安全に再構成できる．ただし，リファクタリングする前には，少なくともシステムがほぼ正しく動作する水準に達している必要がある．

リファクタリング技術は実践的アプローチであるので，実用規模のシステム再構成に応用しやすい．また，ソフトウェアの再利用性や保守性，部品化などを目的としており，技法の多くは局所的かつ汎用的である．それらを組み合わせた，「手続き的な設計からオブジェクト指向設計への変換」や，「プレゼンテーションとドメインの分離」のような，大局的なリファクタリングもある[3]．

これまでシステムレベル設計へのリファクタリング技術の適用可能性について研究を進めてきた[6][7]．適用するリファクタリング技法，適用順序などを体系的にまとめることで，システムレベル設計への適用が有効であると考えている．

3 提案手法

図1，図2に提案手法の設計フローと概念図を示す．なお，図1中の（A），（B），（C），（D）は図2中の（a），（b），（c），（d）に対応している．図1に沿って説明する．

はじめに，システム仕様記述言語を用いてシステム仕様

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

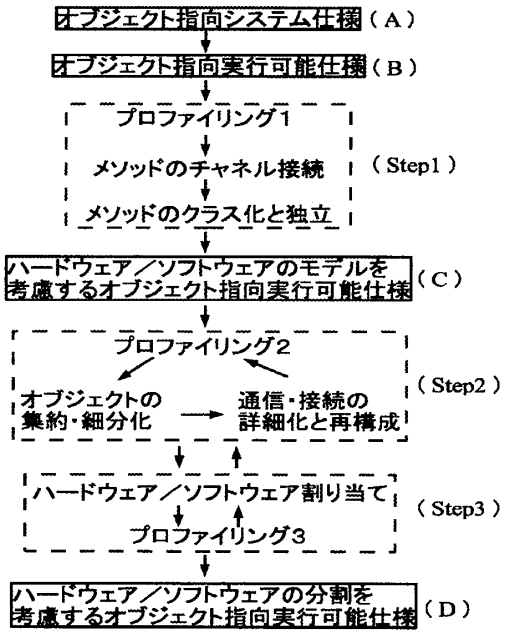


図 1: 設計フロー

(A) を記述する．提案手法では UML を用いる．次に，システム仕様をもとにオブジェクト指向プログラミング言語を用いて実行可能仕様 (B) を記述する．この実行可能仕様 (B) をテストし，要求された機能を満たしていることを確かめる．続いて，(Step1) で HW ／ SW のモデルを考慮するように実行可能仕様 (B) を再構成する．この再構成の結果得られる実行可能仕様 (C) はシステムに必要なオブジェクトのみで構成され，全てのオブジェクトはチャネルを介して通信する．その後，(Step2) で HW ／ SW の分割を考慮するために実行可能仕様 (C) を再構成する．最後に，(Step2) で得られた実行可能仕様を (Step3) で HW 実装部分と SW 実装部分へ分割する．HW ／ SW 分割が困難な場合は (Step2) へ戻る．(Step1)，(Step2) および (Step3) の再構成により，HW ／ SW の分割を考慮した実行可能仕様 (D) を得る．

図 1 に示すとおり，提案手法のシステム再構成は大きく 3 つのステップからなる．それぞれのステップについて，以下に述べる．

3.1 再構成 (Step1)

HW ／ SW のモデルを考慮するための最も簡単な方法は，チャネルを導入することである．チャネルを導入することでメソッド間あるいはオブジェクト間の接続と通信を明確にする．また，この再構成によりシステムに必要なオブジェクトのみで構成される実行可能仕様を得る．図 2(b) の場合，object1 の m12 と m13，object3 の m32 と m33，object4 の m43 はシステムに必要ない．これらメソッドを除外することで HW リソース制約などに対処する．

そこでまず，プロファイリングを行いシステムに必要なメソッドとフィールドを洗い出す．次に，プロファイリングの結果得られたフィールド，メソッドの引数，メソッドの返り値の型に合わせてチャネルを定義し．メソッドの引数や返り値代入用のフィールドをチャネル化する．メソッドの引数を参照している部分と返り値を代入している部分をチャネルへアクセスするように書き換え，メソッドをチャネルで接続する．最後に，チャネルで接続されたメソッドをクラス化し，もとのクラスから独立させる．

ここで使用するリファクタリング技法は，「オブジェクトによるデータ値の置き換え」，「メソッドの引き下げ」，「階層の平坦化」，「フィールドの移動」，「メソッドの移動」，「クラスの抽出」，「局所的拡張の導入」である．

3.2 再構成 (Step2)

この再構成では，オブジェクトの集約・細分化・共有，および通信・接続の詳細化と再構成を行う．

長いメソッドや，容量が大きいフィールドを持つオブジェクトは HW としての実装が困難となることがある．このような場合はオブジェクトを細分化する．細分化により機能分担が可能となり，並列性を向上させることで処理時間を短縮できる可能性が出てくる．また，HW として実装していた機能を分割し，一部を SW 実装することで，設計面積を削減できる可能性が出てくる．例えば，図 2(c) 中の object-m11 のコード数が他のオブジェクトと比べて多いとする．この場合，図 2(d) のように object-m11 から object-m111 を抽出することで object-m11 を細分化する．

短いメソッドや，容量が小さいフィールドを持つオブジェクトは他のオブジェクトへ集約する．集約により，オブジェクト間の接続・通信に使用していたチャネルを削減し，設計面積や処理時間を少なくすることで設計制約を満たしやすくする．例えば，図 2(c) 中の object-m42 のコード数が他のオブジェクトより少ないとする．この場合，図 2(d) のように親オブジェクトである object-m22 へ集約する．

必要があれば同一機能のオブジェクトを共有する．図 2(b) 中の object4 の m41 メソッドは object2 の m21 メソッドと m22 メソッドから呼び出されている．このような記述の場合，図 2(c) 中の object-m41 のように，同一機能のオブジェクトがシステム内に複数存在してしまう．よって，設計面積制約が厳しく，処理時間制約に余裕がある場合は object-m41 を共有するほうがよい．

データ抽象化はオブジェクト指向技術の特長の 1 つである．これにより抽象データ型を定義できる．通信に抽象データ型を使用している場合，ビットベクタ型やシグナル型などの HW 指向のデータ型へ書き換える必要がある．

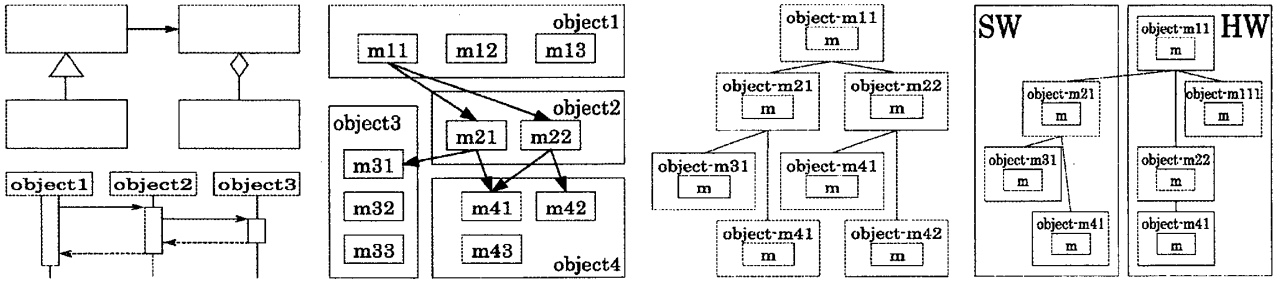
図 1 中の実行可能仕様 (C) を構成するオブジェクトは全てチャネルで接続されている．接続形態は 1 対 1 接続，多対多接続の 2 つに大別できる．

1 対 1 接続の場合，オブジェクト間の接続は直結とし，オブジェクト間はハンドシェイク通信とする．

多対多接続の場合，チャネルはレジスタ，バス，メモリとし，オブジェクト間の接続と通信はそれらを介する形態とする．複数のオブジェクトがチャネルに接続していることから，チャネルへの同時アクセスが起こり得る．その場合はアービターを用いてチャネルへのアクセスを排他制御しなければならない．

そこでまず，以下に挙げる項目についてプロファイリングする．

- メソッドのコード数
- フィールドの数と容量
- 同一機能のオブジェクトの有無
- チャネルで接続されているオブジェクトの数



(a) System Spec. with UML. (b) Executable Spec. (c) HW/SW modeled Spec. (d) Partitioned Spec.

図 2: 概念図

次に，プロファイリングの結果をもとに，オブジェクトの集約・細分化を行う．また，必要があれば同一機能のオブジェクトを共有する．その後，チャネルの実装を決める．

ここで使用するリファクタリング技法は，「メソッドの抽出」，「クラスの抽出」，「フィールドの移動」，「メソッドのインライン化」，「フィールドのインライン化」，「フィールドの移動」，「メソッドの移動」である．

3.3 再構成（Step3）

HW ／ SW 分割の目的は設計制約を満たすことであり，様々な分割基準と分割方法がある．ここでは，HW ／ SW の割り当ての後，プロファイリングすることで，設計制約に違反していないかどうかを確かめる．

提案手法は，現段階では，HW ／ SW の割り当て方法やプロファイリングの項目などを詳細に決めておらず，それらは外から与えられると仮定している．

4 例題設計

提案手法の有効性を確かめるために，インターネットルータを例題として設計した．我々の研究室では，計算機ネットワークに関するテーマも扱っているからである．ルータは単一計算機上に仮想的に作成し，経路表を作成する基本機能のみを実装することとした．設計したルータは，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．ルータ間で授受する情報は，始点ノード，終点ノード，リンクのコストなど，それぞれのルータに必要な情報だけである．

以下に，例題設計の前提条件，例題設計の流れと結果，評価と考察を述べる．

4.1 前提条件

提案手法は実行可能仕様の再構成に主眼を置いているので，設計の際の前提条件を以下のようにした．

設計仕様の記述には一般に広く用いられている UML を，実行可能仕様の記述には並行処理の記述が容易なオブジェクト指向プログラミング言語である Java を用いた．java.lang.Runnable インタフェースを全てのオブジェクトで実装し，オブジェクトを並行動作させることとした．

実行可能仕様（B）を記述する際，特に以下の制限事項を設けた．

- コンストラクタ外での動的インスタンス生成の禁止
- コレクションクラスおよび可変長配列の使用禁止
- インスタンス変数の授受の禁止
- 再帰呼び出しの使用禁止

（Step2）の「通信・接続の詳細化と再構成」におけるチャネル実装は，バス結合・バス通信のみとした．（Step3）の HW ／ SW 分割基準は，最も簡単なオブジェクトの動作頻度とし，動作頻度が高いオブジェクトを HW へ，動作頻度が低いオブジェクトを SW へ割り当てることとした．

システム LSI 設計の主要な設計制約である設計面積制約，処理時間制約，消費電力制約については考慮しないこととした．提案手法は現在のところ，時間の概念および具体的な見積もり手法を取り入れていないからである．

4.2 例題設計の流れと結果

はじめに，ユースケース分析をおこない，「経路を計算する」，「経路情報を送受信する」，「次ホップを検索する」，「経路表を更新する」の 4 つのユースケースを得た．これらユースケースを実現するために，「経路探索」，「通信」，「経路表」の 3 つのクラスを定義した．経路を計算するために必要な情報は，「送信元」，「宛先」，「宛先までのコスト」，「転送先」の 4 つであった．これらの情報をまとめる「経路情報」クラスを定義した．仮想ルータは，「経路情報」クラスのインスタンスを授受することで経路を計算することとした．これらをもとにシステム仕様に必要となる図を作成した（図 3(a)）．なお，紙面の都合上，システム仕様の一部のみ図示している．

続いて，図 3(a) をもとに実行可能仕様（図 3(b)）を作成した．

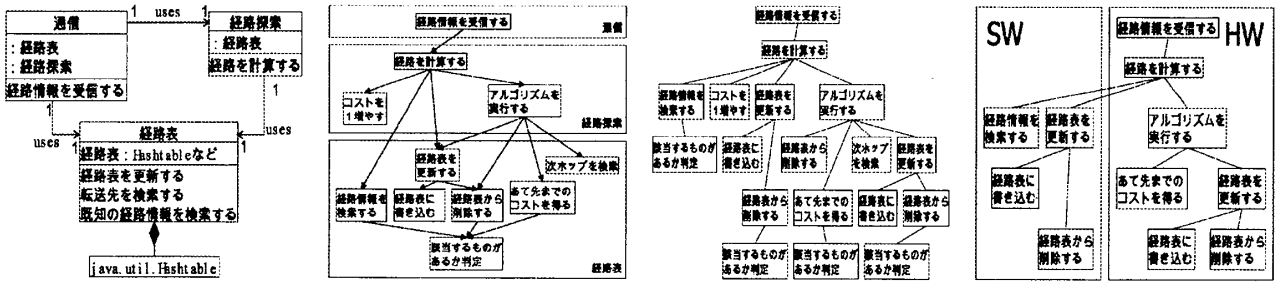
その後，（Step1）に従い実行可能仕様を再構成し，HW ／ SW のモデルを考慮した実行可能仕様（図 3(c)）を得た．この実行可能仕様に対してオブジェクトの集約・細分化のためのプロファイリングを行った．プロファイリングの項目は最も簡単なオブジェクトのコード数とした．

最後に，オブジェクトの動作頻度についてプロファイリングし，その結果をもとに HW ／ SW 分割を行った（図 3(d)）．

システム再構成による距離ベクトル型ルータのクラス数，オブジェクト数，コードの行数の変化を表 1 に示す．なお，コードの行数はコメント行，空白行を除いた行数である．

4.3 評価と考察

提案手法は，UML によるシステム仕様をもとに，記述の抽象度をシステムレベル設計言語の抽象度まで段階的に下げる設計手法である．そこで，システムレベル設計言語による時間の概念がない実行可能仕様と同等の抽象度であることを確かめるために，全てのオブジェクトを HW へ割り当てると仮定し，それらを SpecC を用いて書き換えた．結果を表 1 に示す．SpecC への書き換え手順は以下の通りで



(a) System Spec. with UML. (b) Executable Spec. (c) HW/SW modeled Spec. (d) Partitioned Spec.

図 3: 実験結果

表 1: 距離ベクトル型ルータのクラス数・オブジェクト数・コード数

実行可能仕様	(B)	(C)	SpecC
クラス数	4	14	14
オブジェクト数	4	14	14
コードの行数	84	333	253

ある.

1. behavior にクラスと同じ名前をつける.
2. クラスのコンストラクタ引数を behavior の引数とする.
3. クラス内で定義したチャネルとオブジェクトを，behavior 内に同様に定義する.
4. クラスで定義しているメソッドを behavior にコピーする.

SpecC への書き換えは 1 回だけで済み，バグの混入はなかった．シミュレーションの結果，外部振る舞いが保存されていることが確認できた.

提案手法は実行可能仕様を再構成することで詳細化するので，図 1 の実行可能仕様（D）は設計初期のシステム仕様や実行可能仕様に強く依存する．設計初期においてシステムに適切でない設計および実行可能仕様であった場合，不適切な HW ／ SW 分割となるおそれがある．しかし，(Step1)，(Step2)，(Step3) により，システムに適切な方向へ再構成することで，不適切な HW ／ SW 分割となる可能性を低くする.

図 3(d) に示すとおり，図中に 2 つある「経路表を更新する」以下のオブジェクトを HW 部と SW 部に同時に実装できることがわかる．メソッド単位や関数単位で HW ／ SW 分割を行うと，複数のメソッドから呼ばれるメソッドは HW ／ SW のどちらか一方にのみ実装されてしまう．提案手法では，オブジェクト単位で HW/SW へ割り当てることでこの問題に対処する.

5 むすび

本稿は，HW ／ SW のモデルや分割を考慮しない，より高い抽象度における設計手法を提案した．この設計手法では，システム仕様をもとに作成した実行可能仕様を再構成する．再構成はソフトウェア工学の成果であるリファクタリング技術を応用した．リファクタリング技術を応用することにより，再構成の前後でシステムの挙動を保存することができた．また，実行可能仕様を再構成することにより，システムレベル設計言語による時間の概念がない実行可能仕様と同等の抽象度とすることができた．このことから，システムレベル設計の詳細化にリファクタリング技術が有効であることがわかった.

今後の課題を述べる．4.1 の前提条件を緩めて，より実用規模の設計に対応することである．再構成の各ステップにおけるプロファイリングの項目，評価基準，評価方法を詳細化する必要がある．プロファイリングの結果を利用し，性能や消費電力を考慮するシステムレベル設計向けのリファクタリングの体系化などを検討する必要がある.

現段階では，手動でシステム再構成を行っている．簡単なリファクタリングは，いくつかの統合開発環境において自動化されており，自動化に関する研究も活発に行われている．これらのツールや研究成果を提案手法に取り入れることで，設計生産性を向上させることができ，実用規模のシステムレベル設計が容易になると期待できる.

参考文献

[1] W.H. Tan, P.S. Thiagarajan, W.F. Wong and Y.Zhu, "Synthesizable SystemC Code from UML Models", 41st Design Automation Conference (UML for SoC Design), 2004.

[2] Elvinia Riccobene, Alberto Rosti and Patrizia Scandurra, "Improving SoC Design Flow by means of MDA and UML Profiles", 3rd Workshop in Software Modeling Engineering (WiSME2004), 2004.

[3] マーチン・ファウラー, "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, 東京, 2000.

[4] 吉田紀彦, "プログラム変換手法による自動プログラミング", 自動プログラミングハンドブック (大野, 原田編, オーム社), 109-120, December, 1989.

[5] 吉田紀彦, "プログラム変換に基づくシストリック・アレイの導出", 情報処理学会論文誌, 30:12, 1530-1537 (December, 1989).

[6] 山崎亮介, 吉田紀彦, 楢崎修二, "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", 電子情報通信学会/情報処理学会 情報科学技術フォーラム (FIT) 2004, Vol.1, 233-235, September, 2004.

[7] 山崎亮介, 吉田紀彦, 楢崎修二, "リファクタリング技術を応用した HW/SW 分割", 電子情報通信学会技術研究報告, 104:477, 43-48, December, 2004.