RC-007

実行可能UMLのリファクタリングに基づく段階的詳細化設計
Stepwise Refinement Based on Refactoring of Executable-UML

木村 正裕[†] Nurul Azma Zakaria[‡] 照屋 朗[‡] 松本 倫子[‡] 吉田 紀彦[‡]
Masahiro Kimura Nurul Azma Zakaria Akira Teruya Noriko Matsumoto Norihiko Yoshida

1. はじめに

組込みシステム設計やSoC（System-on-a-Chip）設計において，設計生産性や品質の向上を目的として，システムレベル設計手法 [1] が考案されて実用化されつつある．システムレベル設計における仕様モデルから実装モデルまでの段階的詳細化の変換手順について，我々は研究プロジェクトを進めており，これまでにリファクタリング技術 [2] を応用して言語レベルでの規則化，定式化に成功して実証実験も行い，変換手順のカタログ化・柔軟なツール化について見通しを得た [3, 4, 5].

一方で近年，UML（Unified Modeling Language）を用いた組込みシステム設計やSoC設計が注目を集めており，そのための研究 [6, 7, 8] や標準化 [9, 10, 11, 12] が活発に進められている．しかし，UMLの問題点として，作成したモデルを抽象レベルで検証できないこと，また，段階的詳細化設計を設計言語に依存した形で行わなければならないことなどがある．

そこで本研究では，これまでの成果である [4] で定式化した言語レベルの変換（リファクタリング）規則を，近年ソフトウェア工学の分野で注目されているモデル駆動アーキテクチャ（MDA：Model Driven Architecture）[13] に基づく実行可能UML（xUML，Executable UML）[14] に適用し，それに基づいてシステムレベル設計の段階的詳細化を構成することを目指す．このように段階的詳細化の手順までをUMLないし実行可能UMLで記述，定式化した研究事例は，今のところ他に見受けられない．

以上のアプローチにより，システムの仕様から実装までを一貫してUMLベースで記述できるようになり，システムの再利用性や生産性，品質の向上が期待できる．

そこで本論文ではまず，実行可能UMLで先の言語レベルと同様のリファクタリング規則を定式化できるかについて，モデル記述や規則の定式化の実現可能性を検証すること，および，その実現方法を確立することを目的としている．

具体的には，実行可能UMLでシステムレベル設計を記述するためのモデル化の指針を提案する．加えて，設計言語レベルでのリファクタリング規則の内でも重要なものとして，機能の詳細化部分における規則を特に取り上げ，単純な動作モデルを例に，実行可能UMLでの規則の定式化を行う．ここで述べる機能の詳細化とは，段階的詳細化の仕様モデルからアーキテクチャ探索におけるスケジューリングまでを指す．さらに，対象としたリファクタリング規則を，実際のシステム設計事例であるGSM（Global System for Mobile Communications）ボコーダの符号器の段階的詳細化設計 [5, 15] に適用し，規則の実用性を確認する [16]．なお，本研究の最終目標は，段階的詳細化設計のすべての段階において，実行可能UMLでリファクタリング規則を定式化し，カタログ化，支援ツール化することにある．

[†]東芝ソリューション(株) Toshiba Solutions Corp.
[‡]埼玉大学 Saitama University

2. システムレベル設計

システムレベル設計とは，システムの設計仕様を段階的に詳細化する過程でハードウェアとソフトウェアに分割・実装し，最終的にそれらを一つに統合するシステム設計手法である．代表的なシステムレベル設計言語の一つであるSpecC [1] では，段階的詳細化の過程を大きく4つの段階に分けている（図1）．しかし，その過程を構成する個々のモデル変換は，これまでツールに埋め込まれて，追加拡張や検証などの難しい形で提供されてきた．したがって，今後の発展のためには，変換過程の定式化とカタログ化が一つの課題となる．

3. リファクタリング

リファクタリングとは，プログラムの挙動を保存しつつ，その内部構造を変換，再構成することである．「メソッドの移動」や「クラスの抽出」など，コードの再利用・保守を容易にするための様々な変換規則が定義されている [2]．ただし，いわゆるプログラム変換とは異なり，挙動の保存，セマンティクスの保存を理論的に保証するのではなく，実証的に確認しつつ再構成を進める．

4. 段階的詳細化のリファクタリング規則

システムレベル設計におけるモデル変換過程では，設計仕様を表した仕様モデルから，その機能的な挙動を保ちつつ，段階的に抽象レベルを下げていき，最終的な結果である実装モデルへとモデル（コード）変換していく．これは，3. で述べたリファクタリングの「ソフトウェアの挙動を保ちつつ，コード変換を行い，ソフトウェアの内部構造を再構成する」ことと同様である．したがって，リファクタリングを用いてシステムレベル設計，特に段階的詳細化におけるモデル変換を定式化，体系化することができる [3, 4, 5]．しかし，これまでは言語レベル（具体的にはSpecC）での定式化にとどまっており，

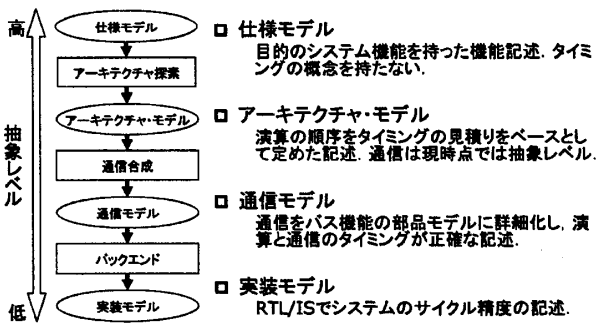


図1: SpecC方法論における設計フロー

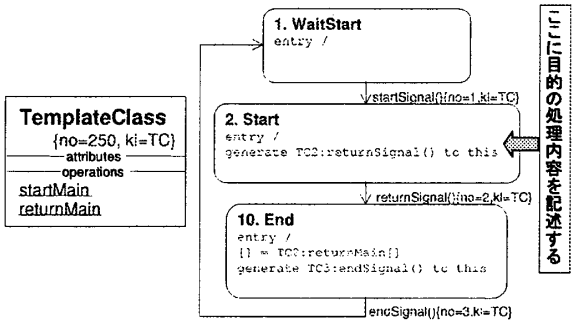


図 2: 実行可能 UML のビヘイビア・クラス

例えば UML や MDA に基づくシステムレベル設計には適用できなかった.

5. 実行可能 UML

実行可能 UML とは，アクション言語とモデルコンパイラを追加した UML である．クラス図，ステートチャート図，プロシージャ（アクション群）を用いてシステムの仕様を記述する．したがって，作成したモデルを実行し，挙動の検証が行える．さらに，実行可能 UML モデルから，C や Java などのプログラムコードを自動生成できる．

本研究では，実行可能 UML ツールとして，iUML [17] を採用した.

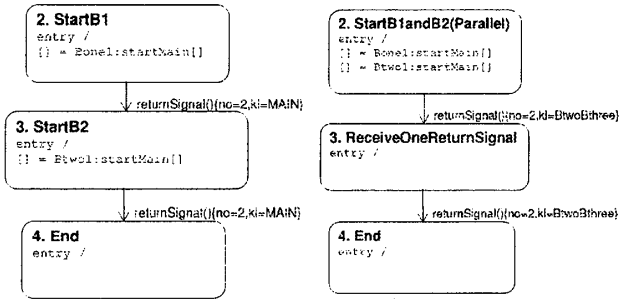
6. 実行可能 UML でのモデル化

実行可能 UML を用いてシステムレベル設計を実現するために，本研究で考案したモデル化の指針について，まず述べる．

システムレベル設計におけるビヘイビア（機能動作）は，実行可能 UML では次のように記述するものとする（図 2）．すなわち，インスタンス生成時の初期状態は，ステート WaitStart にある．他のクラスが，このクラスの startMain 操作を呼び出すと，シグナル startSignal がこのクラスに送られ，次の状態（この例ではステート Start）に遷移する．実際の処理内容は，ステート WaitStart とステート End の間に記述する．この処理が終了すると，状態はステート End に遷移する．最後に，returnMain 操作を呼び出し，このクラスの呼び出し元のクラスに処理の終了を通知する．また，自身にシグナル endSignal を送信し，初期状態のステート WaitStart に遷移を戻し終了となる.

本研究ではビヘイビア・クラスの他に，チャネルなどもクラスとしてテンプレート化した．これらのテンプレートを組み合わせることで設計を行う.

また，ビヘイビアにおける逐次実行と並列実行を表現するために，ステートマシンを図 3 に示すように構成した．(a) は B1 クラスと B2 クラスの逐次実行，(b) は B1 クラスと B2 クラスの並列実行である.



(a) 逐次実行　　(b) 並列実行

図 3: 逐次実行と並列実行の記述

7. 実行可能 UML におけるリファクタリング規則

本論文でまず取り上げた段階的詳細化のリファクタリング規則（計 14 規則）を，以下に一覧で示す.

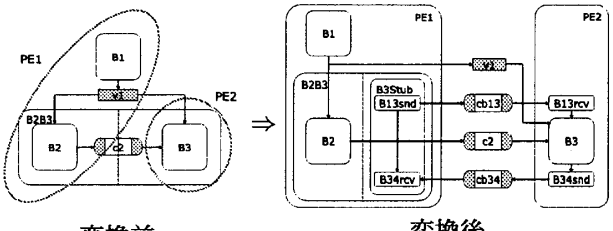
- 通信と演算の分離
  1. チャネルクラスの定義
  2. チャネルクラスの追加
  3. 動作クラスの通信の更新
- PE（プロセス・エレメント：処理要素）の割り付け
  1. 新たなクラス階層の導入
  2. 動作クラスの割り付け・グループ化
  3. 同期クラスの追加
  4. 通信の移動
- 変数分割
  1. PE クラス内への属性の移動
  2. 属性用のチャネルクラスの追加
  3. 属性へのアクセスの更新
  4. 属性用チャネルクラスと同期通信用チャネルクラスの併合
- スケジューリング
  1. クラス階層の平坦化
  2. クラス階層の逐次化
  3. クラス構造の最適化

7.1 規則の適用

ここでは例として，「PE の割り付け」規則の適用を示す．「PE の割り付け」は，ハードウェアとソフトウェアの切り分けなども含む，抽象仕様から実装アーキテクチャを決定する作業である.

「PE の割り付け」のリファクタリング規則の適用順序

1. 新たなクラス階層の導入
   1. 目的のアーキテクチャごとにクラス階層（PE）を追加する.
   2. ハードウェアを示すステレオタイプ（役割などを表す識別子．ここでは verb—¡HardWare¿— など）を必要な箇所に追加する.
   3. 追加したクラスを並列実行するように上位クラスのステートマシンを変更する.
2. 動作クラスの割り付け・グループ化
   1. 各 PE クラスに動作クラスを割り付ける.
   2. PE クラス間にまたがる割り付けをした場合，必要に応じて元々存在していた一方の動作クラスの箇所にスタブ用の動作クラスを追加する.



変換前　　変換後

図 4: PE の割り付け（SpecC での表現）

3. PE クラスの下位クラスに PE クラスと同じステレオタイプ（HardWare など）を追加する．
4. クラスの実行処理（実行先や終了の戻り先，属性の参照先クラス）が適切になるように，各クラスのステートマシンや操作の内容を更新する（以下，「クラスの実行処理を更新する」と呼ぶことにする）．

3. 同期クラスの追加
   1. 並列動作する各 PE クラス間でモデルの元々の実行順序を維持するために，PE クラス間に同期用の動作クラスの対とチャネルクラスを追加する．
   2. クラスの実行処理を更新する．
4. 通信の移動
   1. PE 間で必要となる通信（チャネルクラス・属性）をトップレベルのクラスに追加する．
   2. クラスの実行順序を更新する．

この規則を単純なモデル例に適用した結果を示す．図 4 はこのモデルを SpecC 方法論に従って図示したものである [18]．変換前のモデルでは動作 B1 を実行後，動作 B2 と B3 がチャネル c2 で同期を取りながら並列動作する．変換後のモデルは，動作 B3 を PE2（ハードウェア），それ以外を PE1（ソフトウェア）に割り当てた結果である．

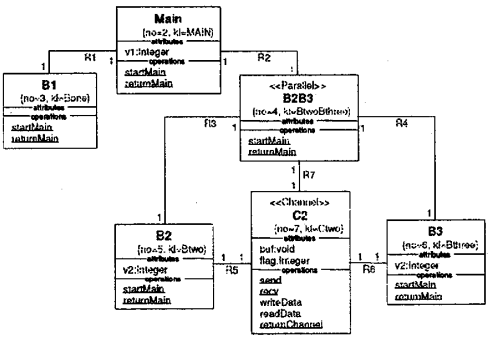


図 5: PE の割り付け（変換前）

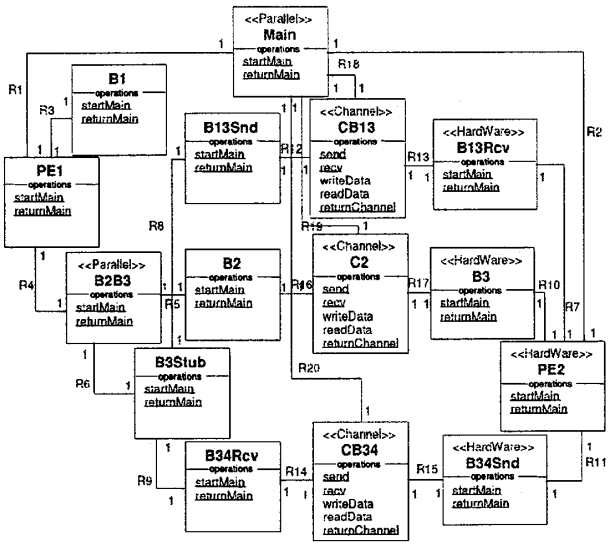
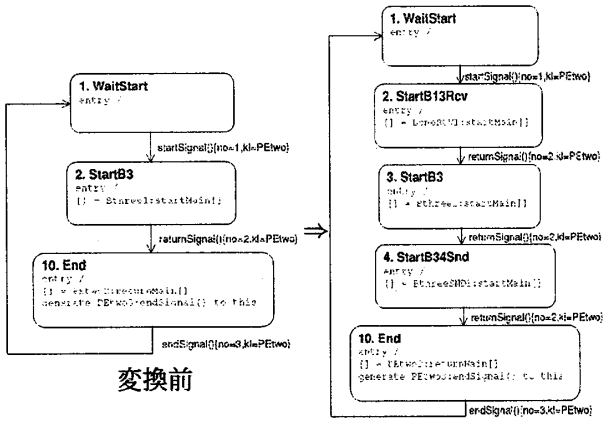


図 6: PE の割り付け（変換後）



変換前　　変換後

図 7: 同期クラスの追加

構造の変換

実行可能 UML での変換前後のクラス図を，図 5 および，6 に示す．各クラスは，トップレベルの Main クラスを起点に，階層的な配置となっている．変換後のクラス図では，スタブ用に追加した PE1 クラス側の動作 B3Stub クラスと PE2 クラスの間に同期動作クラスの対（B13Snd - B13Rcv，B34Snd - B34Rcv）とチャネルクラス（CB13，CB34）を追加している．

状態遷移の変換

状態遷移の変換として，特に「同期クラスの追加」規則の適用を示す．図 7 に示す PE2 クラスのステートチャート図では，変換前には B3 クラスのみ実行していた記述を，変換後には同期クラスも含めた B13Rcv，B3，B34Snd クラスを逐次実行するように変更している．

8. GSM ボコーダへの適用

定式化したリファクタリング規則を実際のシステム設計事例である GSM ボコーダの符号器 [5, 15] に適用し，これらの適用性を確認した．

GSM ボコーダは，ヨーロッパの携帯電話ネットワーク・システム GSM における音声符号化と圧縮の規格に基づいた携帯アプリケーション用の音声符号器/復号器である．そのコーデック（符号化/復号化）方式である拡張フルレート（EFR：Enhanced Full Rate）音声トランスコーディングは現在 European Telecommunication Standards Institute（ETSI）により GSM06.60 として標準化されている．音声合成ボコーダは，人間の音声を単純なデジタル信号に変換する機能を持つ．

実行可能 UML で記述した仕様モデル，すなわち「PE の割り付け」適用前のクラス図を図 8 に，「PE の割り付け」適用後のクラス図を図 9 にそれぞれ示す．ただし，主要部分のみを示している．変換後のモデルでは，Codebook クラスを HW で，それ以外のクラスを DSP で実装するものとし，また，DSP クラス側の Codebook_Stub クラスと HW クラスの間に，同期動作クラスの対（Codebook_Start_Send - Codebook_Start_Recv，Codebook_Done_Send - Codebook_Done_Recv）とチャネルクラス（Ch_Codebook_Start，Ch_Codebook_Done）を

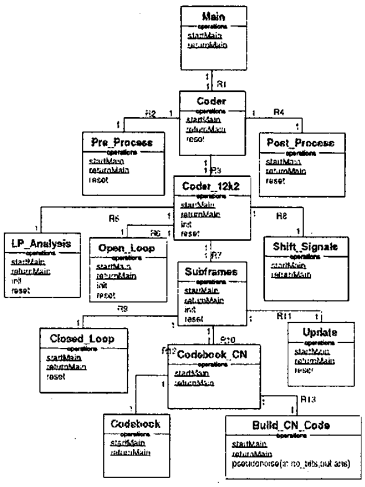


図 8: ボコーダの構造（変換前）

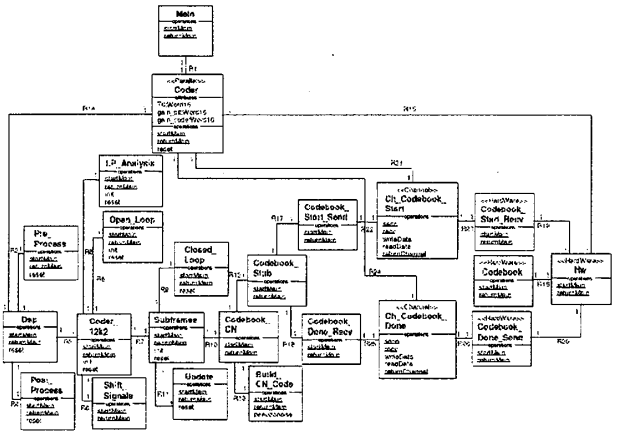


図 9: ボコーダの構造（変換後）

挿入している．なお，ステートチャート図の変換例については，紙面の都合から省略する．

本研究では，以上についてシミュレーションによる検証を行い，変換前後のモデルで挙動（実行順序）が保たれていることを確認した．

9. まとめと今後の課題

本論文では，システムレベル設計における段階的詳細化について，言語レベルで定式化してきたリファクタリング規則を，実行可能 UML に適用する方策を示した．まず，実行可能 UML でシステムレベル設計を実現するためのモデル化の指針を考案した．そして，システムレベル設計で最も重要である機能の詳細化設計を対象として，リファクタリング規則の定式化を行った．その結果，14 の規則を実行可能 UML で定式化できた．機能の詳細化に続く残り 15 規則も，同様に定式化が可能であると考えている．さらに，これら実行可能 UML 向けのリファクタリング規則を，実際のシステム設計事例である GSM ボコーダの符号器の段階的詳細化設計に適用し，規則の実用的な可能性を確認した．

今後の課題としては，実行可能 UML におけるリファクタリング規則の定式化の継続，変換規則の厳密な記述とカタログ化，段階的詳細化の自動変換ツールの作成などが挙げられる．厳密な記述と自動化に向けては，アスペクト指向に基づく形式を検討している．

参考文献

[1] D. D. Gajski, J. Zhu, R. Dömer et al. (木下, 冨山訳), “SpecC 仕様記述言語と方法論”, CQ 出版社, 2000.

[2] M. Fowler (児玉他訳), “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, 2003.

[3] R. Yamasaki, K. Kobayashi, N. A. Zakaria, S. Narazaki, N. Yoshida, “Refactoring-Based Stepwise Refinement in Abstract System-Level Design”, Proc. IFIP 2006 Int. Conf. on Embedded and Ubiquitous Comp. (LNCS, No.4096, Springer), pp.712–721, 2006.

[4] 木村, 小林, 山崎, 吉田, “システムレベル設計におけるリファクタリング規則の定式化とカタログ化”, 情報科学技術フォーラム 2006 論文集, Vol.1, pp.169–172, 2006.

[5] 木村, 小林, 山崎, 吉田, “リファクタリングに基づく段階的詳細化設計の GSM ボコーダへの適用”, 組込みシステムシンポジウム 2006 論文集, pp.83–87, 2006.

[6] K. D. Nguyen, Z. Sun, P. S. Thiagarajan, W. F. Wong, “Model-Driven SoC Design via Executable UML to SystemC”, Proc. 25th IEEE Int. Real-Time Systems Symp., pp.459–468 2004.

[7] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, “An HW/SW Co-Design Environment Based on UML and SystemC”, Proc. Forum on Specification and Design Languages, 2005.

[8] W. Mueller, A. Rosti, S. Bocchio, E. Riccobene, P. Scandurra, W. Dehaene, Y. Vanderperren, “UML for ESL Design: Basic Principles, Tools, and Applications”, Proc. 2006 IEEE/ACM Int. Conf. on Computer-Aided Design, pp. 73–80, 2006.

[9] “UML Profile for System on a Chip (SoC), Version 1.0.1”, http://www.omg.org/technology/documents/formal/profile_soc.htm

[10] E. Riccobene, P. Scandurra, A. Rosti, S. Bocchio, “A UML 2.0 Profile for SystemC”, ST Microelectronics Technical Report, 2005.

[11] SysML, http://www.omgsysml.org/

[12] MARTE (Modeling and Analysis of Real Time and Embedded Systems), http://www.omgmarte.org/

[13] D. Frankel (日本 IBM TEC-J MDA 分科会 監訳), “MDA モデル駆動アーキテクチャ”, エスアイビーアクセス, 2003.

[14] S. Mellor, M. Balcer (二上他監訳), “Executable UML – MDA モデル駆動型アーキテクチャの基礎”, 翔泳社, 2003.

[15] SpecC WebSite, http://www.ics.uci.edu/ specc/

[16] 木村, “実行可能 UML のリファクタリングに基づく段階的具体化設計”, 埼玉大学修士論文, 2008

[17] Kennedy Carter Ltd., http://www.kc.com/

[18] A. Gerstlauer, R. Dömer, J. Peng, D. D. Gajski (木下訳), “システム設計 – SpecC による実現”, SpecC Technology Open Consortium, 2002.