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

2009－SE－163 (22)
2009／3／19

再利用による自動コード生成を目的とした B Method におけるソフトウェアの部品化

中村 丈洋　　織田 健

電気通信大学大学院 情報通信工学専攻

本稿ではB Methodの段階的詳細化はソフトウェア部品の信頼性向上と部品検索に有効であると考え, 経験則から部品化と再利用の手順を提案する. 提案手法を用いて銀行口座システムを部品化し, その部品を用いて図書館システムを実装する実験では6操作中3操作を実装することが出来た. この実験結果から段階的仕様記述言語に対して部品再利用を用いた自動コード生成を適用することで高信頼ソフトウェア開発を迅速に行うことが出来るのではないかと予想し, これを実現するためには本手法をB Methodの証明責務から再構築する必要があると考察した.

Reuse and auto code generation in B Method

Takehiro Nakamura　　Takeshi Oda

The University of Electro-Communications
Graduate School of Infomation and Communication Engineering

In this paper, we propose a method of reuse and code generation in B Method. We think the B Method is effective for code generation with software conmponent reuse. Our method implement 3 operations of library system, which has 6 operations, reusing bank system. We expect applying reuse and code generation in B Method will allow us to develop highly trusted software rapidly. For this expection, we shuld reconstruct our method based on proof obligations of B Method.

1 はじめに

ソフトウェアの大規模複雑化に伴い, ソフトウェアの再利用や高信頼化が求められている. 同時に近年の情報技術の興亡は激しく, それらへの迅速な対応も重要な課題である. 本手法では高信頼性ソフトウェア開発に実績のあるB Methodに部品再利用を適用することで, 高信頼な自動コード生成を目標とする. ソフトウェア部品再利用を自動化するためには, その部品が再利用可能なものか否かを判断するための情報を部品に与える必要がある. 細粒度リポジトリに仕様などの情報を付与する研究[4]は行われているが, 再利用を自動化, 高信頼化するためには情報が形式化され, さらにその情報が正しいことを保証しなければならない. これに対してB Methodは仕様の段階的詳細化と各段階間の検証によりその高信頼性を得ている. そこで, 各段階間の整合性を保持した仕様の自動細分化を行うことで高信頼な部品を整備し, 入力された仕様を満たす部品群を仕様の比較により検索・結合することで自動コード生成が出きると考えた. 本稿では, 経験則から部品化と再利用の手順をつくり, これによって入力仕様を満たすコード生成が可能である例を挙げることで, この研究目標が達成可能であることを示唆し, さらに研究目標達成の為の課題を挙げる.

2 B Method

B Method[1]は集合論に基づく形式的仕様記述手法の一つである. B Methodの特徴的な点は仕様を段階的に詳細化し, 各段階間に矛盾が無いかを数学的に検証できる点である. これにより, 実装が仕様を満たす事を保証する事ができる.

B MethodではAMN (Abstract Machine Notation)を用いて抽象機械を記述, 詳細化する. 抽象機械はMODEL, REFINEMENT, IMPLEMENTATIONの順に詳細化され, IMPLEMENTATIONまで記述することで実行可能コードを得る.

以下の節ではB Methodの性質と, 本研究における着眼点を述べる.

2.1 集合と状態

B Methodでは集合をデータ型として扱い, 変数はその部分集合や要素として表現される. 集合や部分集合がそれぞれどの様な関係を持つかは不変条件によって与えられる. この不変条件を読み取ることにより, 集合や変数をベン図に書き下す事が出来る. 操作による変数の変化はベン図上では集合間の要素移動と見る事が出来る. すなわち, それぞれの集合について操作をイベントとした状態遷移図に書き下す事が出来る.

以上の事からB Methodで表されるシステムは状

態間の関係を表す不変条件と状態遷移を表す操作中の式の組み合わせによって表現されていると捉える事が出来る．ここから，1つの状態遷移を表すのに必要な不変条件と操作中の式をまとめて部品とし，これを組み合わせることでシステムを記述出来ると考えた．

2.2 リンク不変条件

リンク不変条件はMODEL, REFINEMENT, IMPLEMENTATIONという各段階間で変数がどの様な関係を持つかを表す式である．これにより，IMPLEMENTATIONがMODELを満たすかを検証する事が出来る．

3 研究目標と本手法の位置づけ

本研究の目標は段階的仕様記述言語に対し，部品再利用を用いた自動コード生成を適用することで高信頼ソフトウェア開発を迅速に行うことである．高信頼ソフトウェアを自動生成することは一般に難しい．本研究では自動化された再利用で可能な限りコードを生成し，生成出来なかった部分に対してはユーザがコードを追記することでソフトウェアを迅速に開発する．再利用で可能な限りコードを生成するためにソフトウェアを細分化し，これを部品とする．また，B Methodの枠組みの中で細分化，再利用を行うことで高信頼性を保証する．

本稿では再利用手法を構築し，入力仕様に対して正しい実装を得られる例を示す．しかし，この再利用手法は経験則から構築されているため，これを他の入力仕様に適用する為には6.1節に示すように細分化と再利用手順を改善する必要がある．

4 提案手法

提案する手法はB MethodにおけるMODELを入力，REFINEMENT, IMPLEMENTATIONを出力とした高信頼なコード自動生成である．B Methodの記述を蓄積，再利用するためにそれらを分割してリポジトリに格納する．また，仕様から実装を得るために仕様を分割し，それに合致する部品をリポジトリから取得，結合する．合致する部品が見つからない場合には不完全な実装が得られるため，足りない実装を追記し，得られたソフトウェアを再び部品化することでリポジトリを強化する．

4.1 コード生成と部品登録の流れ

本手法は図1に示すように記述分割器，部品検索器，記述統合器からなる．以下にそれぞれの役割を説明する．以降，コード生成の入力されるMODELを入力仕様，出力されるIMPLEMENTATIONを出力実装と呼ぶ．

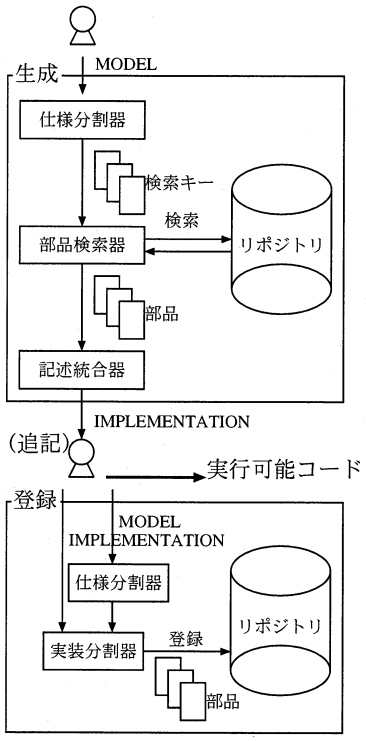

図 1：提案手法概要

仕様分割器 与えられたMODELを部品の単位に分割する．仕様分割器の出力を細分化仕様と呼ぶ．コード生成時には細分化仕様は部品検索の為の検索キーとなる．また，部品登録時には細分化仕様は実装分割器の入力となる．

実装分割器 実装分割器は与えられたIMPLEMENTATIONから細分化仕様に対する実装を抽出し，部品を生成する．

部品検索器 細分化仕様を検索キーとしてリポジトリから対応する部品を検索する．これに対するアプローチは4.6節で説明する．

記述統合器 検索で得られた部品を統合し，IMPLEMENTATIONを生成する．MODELの変数や型と部品の変数や型の対応付けは部品検索の時点で行われるため，記述統合器は部品の結合と重複した記述の削除を行う．

本手法におけるユーザの役割はMODELを記述する**仕様記述者**と不完全なIMPLEMENTATIONを受け取り，足りない実装を追記する**実装記述者**の2者である．

4.2 本手法が満たすべきこと

高信頼ソフトウェアを再利用を用いたコード自動生成で行うために，本手法には以下のような要求が

生じる.

1. 再利用可能な部品が見つかった部分のみをコード生成する. 生成されたコードは入力仕様に矛盾しない.
2. 再利用できる部品が見つからなかった部分はユーザが実装を記述する. この際, 他の部分で再利用された変数等を用いるが, 再利用元の部品の実装に関する知識は要求しない.
3. 部品登録を自動化する.

1 と 2 は実装記述者の負担を軽減し, '自動コード生成でありながら, 再利用に失敗した際には実装記述者を必要とする' という欠点を補う.

3 は作業軽減と部品の信頼性を確保する為である. 部品登録の際には MODEL や IMPLEMENTATION を部品の単位に細分化する必要があるが, この工程をユーザが行うことは部品の信頼性を損ね, 結果的に生成されるコードの信頼性を保証する事ができなくなる.

以上の要求を満たすために次の様に手法を構築する.

1. 部品検索機は検索キーと部品の細分化仕様の完全一致によって部品再利用の可否を決定する. (要求 1)
2. 部品が細分化仕様を満たすよう実装分割器を定めることによって部品の信頼性を高める. (要求 1)
3. 4.3 節の様に変数名の規則を定めることで, 自動生成された実装中の変数名を入力仕様の変数名に合わせる. (要求 2)
4. 4.4 節の様に分割手順を定め, 仕様分割器と実装分割器を自動化する. (要求 3)

4.3 変数の命名規則

提案では異なる記述者が記述した複数の仕様と実装を組み合わせることによって入力仕様に対して実装を与える. ここで以下のような問題が生じる.

1. 生成された実装における変数名の衝突
   1 つの部品が複数箇所で再利用された場合, 名前の衝突が起き, 実装を得ることが出来ない.
2. 入力仕様と出力実装における変数名の不統一
   自動生成された実装は不完全な場合がある. 記述者は足りない実装を追記するが, 部品の変数名をそのまま用いると入力仕様の変数名と齟齬が生じるため作業に支障をきたす恐れがある.

以上の問題を解決するため, 本研究では変数名に命名規則を定め, 部品を再利用する際に変数名を入力仕様に合わせる事とした.

1. MODEL における変数名に '_' は用いない. 以降ではこの変数名が 'base' であると仮定する.

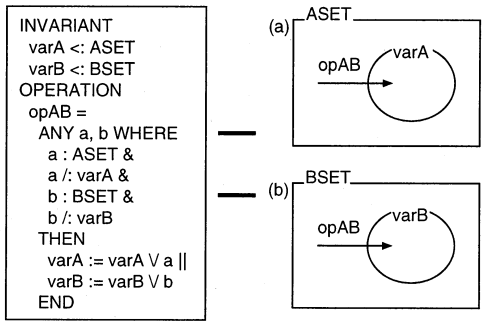

図 2: 操作と状態遷移

2. REFINEMENT では base に対応する変数に base_R という名前をつける
3. IMPLEMENTATION では base に対応する変数に base_I という名前をつける.

4.4 仕様分割

記述の細分化には MODEL を分割して細分化仕様を得る仕様分割と, IMPLEMENTATION を分割して部品を得る実装分割がある. 記述の細分化は部品の再利用性向上と部品再利用の際の検索コスト軽減の為に行われる. 仕様分割器では与えられた MODEL を分割して複数の細分化仕様を出力する. 細分化仕様には部品が満たすべき仕様が記述されるため, どの様に仕様分割を行うかによって部品の再利用性が左右される.

本手法では B Method が状態遷移図に書き下せることから, 仕様細分化を操作による状態遷移に着目して行った.

以下に仕様細分化の手順を示す.

1. MODEL 中の操作 O において, ある型 S に注目し, その型に属する変数が変化する式を操作 O から抽出する.
2. 1 で抽出した式が依存する式を抽出する.
3. 1,2 で抽出した式を細分化仕様とする. 以上の操作を全ての操作, 全ての型について行う.

4.4.1 仕様の分割単位

仕様の分割単位は 1 操作における集合の状態遷移とする. すなわち, 図 2 の様に操作 opAB によって生じる状態遷移 (a), (b), それぞれを表す式を仕様から抽出し, 細分化仕様とする.

4.4.2 依存する式の抽出

'依存する式の抽出' では 1 で抽出した集合が初期化時にどのような状態にあり, また集合間でどのような関係を維持するかを表す式を抽出する. すなわち, 図 2 の (a) の様な状態遷移を表す為には varA := {} の様な初期状態と ASET == NAT & varA : ASET の

様に集合間の関係を表す式を抽出する.

4.5 実装分割

実装分割器は細分化仕様とIMPLEMENTATIONを受け取り, 部品を生成する. 部品には細分化仕様とそれに対する実装が記述される. 部品の持つ実装が細分化仕様を満たすようIMPLEMENTATIONを分割することによって, 細分化仕様を検索キーとした部品検索においてこれにより, 細分化仕様を検索キーとした部品検索において検索キーと部品の細分化仕様の一致判定を行えばよく, 部品が検索キーを満たすかの数学的判定が不要になる.

以下に仕様細分化の手順を示す.

1. 細分化仕様にあらわれる変数のリンク不変条件をIMPLEMENTATIONから抽出する.
2. 抽出されたリンク不変条件が依存する式を抽出する.
3. 細分化仕様が表す状態遷移に対応する式を操作から抽出する. IMPLEMENTATIONから抽出した式と細分化仕様を記述したものを部品とする.

4.5.1 リンク不変条件の抽出

リンク不変条件とは段階的詳細化を行う際に, 詳細化前の抽象機械の変数と詳細化後の抽象機械の変数間の関係を表す式である. リンク不変条件の抽出では, 細分化仕様に現れる全ての変数について, IMPLEMENTATIONからリンク不変条件を抽出する.

4.5.2 IMPLEMENTATIONにおける依存関係の抽出

抽出したリンク不変条件に現れる変数群についてその初期状態と変数間の関係を4.4.2節と同様に抽出する.

4.5.3 IMPLEMENTATIONからの状態遷移の抽出

4.4.1節で注目した操作の状態遷移を表す式をIMPLEMENTATIONから抽出する. すなわち, 細分化仕様の操作部分に現れる変数とリンク不変条件を持つIMPLEMENTATIONの変数についての式を操作から抽出する.

4.6 部品の検索と統合

ここでは, 入力されたMODELに対してIMPLEMENTATIONを得る手順を説明する.

1. 入力仕様を細分化仕様に分割する (4.4節). 得られた細分化仕様を検索キーとする.
2. 検索キーと部品の細分化仕様を比較し, 再利用可能な部品を取得する.
3. 取得された部品の変数名を命名規則に従い置換する.
4. 取得された部品を制約式に矛盾が生じないよう組み合わせることでIMPLEMENTATIONを生成する.
5. 取得されたIMPLEMENTATIONから重複する式を削除する.

4.6.1 細分化仕様の比較

部品が再利用可能か否かの判定は制約式の完全一致によって行う. すなわち, 部品の細分化仕様を検索キーが満たしたとしても, 部品の細分化仕様が検索キーの仕様を満たさない限りは再利用されない. 例えば, 入力仕様記述者が`x : NAT`の様な制約をしたのに対して, `x : NAT & x \= 100`の様な部品を再利用すると'xが100にならない'という仕様記述者の意図しない動作を招く. そのため, この様な部品を再利用することは出来ない.

B Methodでは集合の詳細をMODELで記述せず, IMPLEMENTATIONで詳細を与えるdeffered setという記法がある. 実装を得るためには集合の詳細が必要であるため, deferred setについては例外的に部品の細分化仕様の詳細を用いて詳細化することとする.

4.6.2 部品中の変数名置換

部品中の変数名を一意に変換することによって, 部品を組み合わせた時に, 異なる部品の間で変数名が衝突することを回避する. 4.3節の命名規則によって部品の変数'`base_I`'と細分化仕様の変数'`base`'が対応する. 検索キーと細分化仕様の比較の段階で細分化仕様のどの変数に検索キーのどの変数が対応するかは分かっているため, この'`base`'部分を対応する変数名で置換することにより, 部品を組み合わせた際に異なる変数が同じ変数名を持つことを回避できる.

4.6.3 部品間の矛盾検出

部品を組み合わせた際, 制約式が矛盾する場合がある. 部品の組み合わせによって, 同じ変数に対して異なるリンク不変条件を与えた場合, その組み合わせでは実装を得ることが出来ない. この様な組み合わせを回避するためにリンク不変条件が合致しない組み合わせを排除してIMPLEMENTATIONを生成する.

5 手法適用例

この節では銀行口座システムの仕様と実装を再利用することで図書館システムの仕様に対して実装を与える.

この例で用いる銀行口座システムと図書館システムは参考文献 [2, 3] のコードを元に修正したものを用いた. コードの一部を図 3, 4, 5 に示す.

銀行口座システムは顧客情報と口座情報を持ち, 顧客登録, 口座登録, 残高照会, 預金, 引落しなどの操作を持つ. 図書館システムは利用者情報と図書情報を持ち, 利用者登録, 利用者削除, 図書登録, 図書削除, 貸出, 返却という操作を持つ.

5.1 銀行口座システムの部品化

図 6 に操作 NewCustomer (顧客登録) における CUSTOMER の状態遷移に着目した部品を示す. この部品は提案手法に基づき以下の手順で抽出した.

1. 操作中で CUSTOMER の状態遷移を表す式として図 3:15 行目が抽出される. この際, 要素 'newCustomer' を規定している ANY-WHERE ブロック (12-14 行目) も抽出する.
2. 状態遷移が依存する式として図 3:5 行目などが抽出される.
3. 銀行口座システムの IMPLEMENTATION より関連する変数のリンク不変条件として図 4:11 行目が抽出される.
4. リンク不変条件に現れる変数についての制約として図 4:2 行目が抽出される.
5. 次に CUSTOMER の状態遷移を表す式を IMPLEMENTATION の操作から抽出するが, この例では Object というモジュールを用いているため, リンク不変条件から抽出対象となる customers_I.object に対する演算が操作として隠蔽されている. これについては操作 NewCustomer において参照される customers_I.Object の操作群から customers_I.object の状態を変更する操作を特定することで抽出できる. 具体的にこれに合致する操作は Object.CreateObject のみであるため, 図 4:18 行目が抽出される.

上記の手順を他の状態遷移にも適用することで銀行口座システムは 32 個の部品に分割される. 表 1 に分割された部品の一部を挙げる. 本手法では部品は状態遷移を表すものであり, 各部品に名前は与えられないが, ここでは仮に名前を振ることとする. 表において名前に 'nc' と付く部品は操作 NewCustomer から, 'na' と付く部品は操作 NewAccount から抽出された.

5.2 図書館システムの実装生成

本節では図書館システムの MODEL に対してを前節で得た部品群を再利用することで IMPLEMENTATION を生成する. 特に操作 registUser におけ

表 1: 銀行口座システムから得た部品 (抜粋)

部品名	説明
nc	customers に要素追加
ncName	customerName に要素追加
ncYob	customerYob に要素追加
na	accounts に要素追加
naNum	accountNumber に要素追加
naOwner	accountOwner に要素追加

```
01 SETS
02   PERSONS
03 VARIABLES
04   users
05 INVARIANT
06   users <: PERSONS
07 INITIALISATION
08   users := {}
09 OPERATION
10     ANY user WHERE
11       user : PERSONS & user /: users
12     THEN
13       users := users \/ {user}
14     END
```

図 7: 図書館システムの細分化仕様 (抜粋)

表 2: 変数名の置換

置換前	置換後
CUSTOMER	PERSONS
customers	users
customers_I	users_I

る PERSONS の状態遷移に着目してその手順を説明する.

1. 前節と同様の手順で操作 registUser において PERSONS の状態遷移に注目した入力細分化仕様を作る. この細分化仕様を図 7 に示す.
2. 図 7 に合致する部品を前節の銀行口座システムから作った部品群から検索する. 図 6 において, CUSTOMER を PERSONS, customers を users に読み替えると図 6:5 行目と図 7:6 行目が対応する. また,
   user : PERSONS & user /: users という式は user : PERSONS - users と等価であるため, 図 6:11-15 行目と図 7:10-14 行目が対応する. 図 7 において PERSONS は型の詳細を持たない deferred set として記述されている. deferred set は型の詳細を MODEL 中で行わず, REFIMENET や IMPLEMENTATION に先送りする記法である. 実装の為には型の詳細を定義する必要があるため, ここでは図 6:7 行目によって NAT 型を与える.
   以上の様に, この細分化仕様は NewCustomer の CUSTOMER に着目した部品と一致する.
3. 一致した部品における変数名を表 2 の様に置換

```
01 VARIABLES
02   customers, accounts, accountOwner,
03   ...
04 INVARIANT
05   customers <: CUSTOMER &
06   accounts <: ACCOUNT &
07   accountOwner : accounts --> customers &
08   ...
09 OPERATIONS
10   NewCustomer(name, yob) =
11   ...
12     ANY newCustomer WHERE
13       newCustomer : CUSTOMER - customers
14     THEN
15       customers := customers \/ {newCustomer} ||
16       customerName(newCustomer) := name ||
17       customerYob(newCustomer) := yob
18     END
19   END;
```

図 3:銀行口座システムの MODEL(抜粋)

```
01 IMPORTS
02   customers_I.Object(10000, 2, NAT, 0),
03   accounts_I.Object(100000, 4, NAT, 0),
04   ...
05 DEFINITIONS
06   customerName_I == 0;
07   ...
08   accountOwner_I == 3
09 INVARIANT
10   ((fileOpen = TRUE) =>
11     customers = customers_I.object &
12     accounts = accounts_I.object &
13     accountOwner = accounts_I.field(accountOwner_I) &
14     ...
15 OPERATIONS
16   NewCustomer(name, yob) =
17   VAR cid, ii IN
18     cid <-- customers_I.CreateObject(0);
19     ii <-- BS.AddString(name);
20     customers_I.SetField(cid, customerName_I, ii);
21     customers_I.SetField(cid, customerYob_I, ii)
22   END;
```

図 4:銀行口座システムの IMPLEMENTATION(抜粋)

```
VARIABLES
  users, copies, issued
INVARIANT
  users <: PERSONS & copies <: COPIES &
  issued : copies --> users
INITIALISATION
  copies := {} || users := {} || issued := {}
OPERATIONS
  nu <--registUser =
    ANY user WHERE
      user : PERSONS & user /: users
    THEN
      users := users \/ {user} ||
      nu := user
    END;
  ...
```

図 5:図書館システムの MODEL(抜粋)

```
01 PART MODEL
02 VARIABLES
03   customers
04 INVARIANT
05   customers <: CUSTOMER
06 DEFINITIONS
07   CUSTOMER == NAT
08 INVARIANT
09     customers := {}
10 OPERATION
11     ANY newCustomer WHERE
12       newCustomer : CUSTOMER - customers
13     THEN
14       customers := customers \/ {newCustomer}
15     END

16 PART IMPLEMENTATION
17 IMPORTS
18   customers_I.Object(10000, 2, NAT, 0)
19 INVARIANT
20   ((fileOpen = TRUE) => customers = customers_I.object) &
21   ((fileOpen = FALSE) => customers = {})
22 OPERATION
23   VAR cid IN
24     cid <-- customers_I.CreateObject(0);
25   END;
```

図 6: 部品化された銀行口座システム (一部)

する.

4. 1 から 5 の操作を他の状態遷移にも適用し, 各状態遷移に対して再利用可能な部品を抽出する. この例では表 3 の様に再利用可能な部品が抽出された.
5. 各状態遷移で再利用可能な部品の組み合わせから, リンク不変条件に矛盾が生じない組み合わせを抽出する. 今回の例ではリンク不変条件に矛盾が生じる組み合わせは存在し無かった. `registUser`, `registCopy` はどちらも部品 nc, na が再利用可能であったが, 今回は `registUser` に nc を,`registCopy` に na を再利用した.
6. 重複する式を取り除く. 今回の例では利用した部品 naOwner に部品 nc と部品 na と同じ制約条件が存在するため, これらを取り除く.

以上によって図 8 の実装が得られた. 図書館システムの 6 操作を細分化した結果, 6 つの細分化仕様が得られた. 表 3 にこれらの細分化仕様に対して銀行口座システムのどの部品を再利用したかを示す. 表では図書館システムが 1 操作 1 部品であったため, 各部品を操作名で表している. 表の様に図書館システムの 6 つの細分化仕様のうち再利用によって 3 つを実装することができた. `removeUser`, `removeCopy`, `returnCopy` に対しては再利用可能な部品が存在し無かった.

```
IMPORTS
  users_I.Object(10000, 2, PERSONS, 0),
  copies_I.Object(100000, 4, COPIES, 0)
DEFINITIONS
  issued_I == 3
INVARIANT
  ((fileOpen = TRUE) =>
    users_R = users_I.object &
    copies_R = copies_I.object &
    issued_R = copies_I.field(issued_I) &
    ...
OPERATIONS
  nu <-- registUser =
    nu <-- users_I.CreateObject(0);
  ...
```

図 8: 生成された図書館システムの実装 (抜粋)

表 3: 図書館システムで再利用可能な部品

細分化仕様	再利用した部品
registUser	nc, na
removeUser	未実装
registCopy	nc, na
removeCopy	未実装
issueCopy	naOwner
returnCopy	未実装

下線は実際に再利用した部品

6 考察

6.1 生成コードの信頼性

5 節では特定の入力に対して提案手法で自動コード生成が可能な事を示した. この例では 6 操作中 3

操作を実装し, 実装できた 3 操作全てが, B Method での検証に合格した. 本手法において高信頼性に特に寄与したのは部品の分割単位を状態遷移とした点であると考えられる. これによって部品化の際に仕様と同じ状態遷移を表す式を実装から抽出することで, 仕様と実装に齟齬が生じることを防ぐことが出来た. しかし, 本稿の段階では提案した手法で仕様を満たす実装が生成できるかという証明は行っていない. B Method では証明責務を満たすか否かで実装の正しさを判定している. このため, 部品分割単位を証明責務から再構築し, 部品化・再利用した際に証明責務を満たす事を保証する必要がある.

6.2 重複部品の扱い

5.1 節で銀行口座システムから抽出した部品のうち, 表 3 の nc と na は変数名が異なるだけで, 同じ細分化仕様を持つ部品であった. 生成された図書館システムの実装 (図 8) において 2 行目は部品 nc から, 3 行目は部品 na から再利用された記述であるが, これを見ると同じ細分化仕様を持つ部品であっても異なる実装が行われていることが分かる. 2 行目では `users_I.object` が保持できる要素数の上限が 10000 であることを, 3 行目では `copies_I.object` が保持できる要素数の上限が 100000 であることを表している. 仮に, 銀行口座システムの部品の細分化仕様に '保持できる要素数の上限が 10000 である' という制約があれば上記の部品は異なる細分化仕様として認識される. このことから, 異なる部品が同じ細分化仕様を持つ場合について, 細分化仕様が不十分な場合と, アルゴリズムが異なるだけで機能としては同じである場合の 2 通りに事例を分類し, 細分化仕様が不十分な場合の検出手法の提案が必要と考えられる.

6.3 モジュールの扱い

B Method では大規模システムの構造化のためにモジュールを構築・利用する事が出来る. 5 節の例でもモジュール Object が登場したが, 本稿ではモジュールに対しては実装分割時にその MODEL を参照するだけに留まった. モジュールによる構造化を保ったまま部品化を行った場合, 再利用性が低下することが予想される. これは, 1 つの変数に対して複数のモジュール化された変数を再利用する事が出来ないためである. しかし, モジュールを用いないで実装を自動生成した場合, 実装の可読性が低下し, 自動生成出来なかった際の追記作業に支障をきたす恐れがある. 以上から, 再利用性を考慮したモジュールの利用には複数モジュールを統合したモジュール再生成が有効であると考えられる.

6.4 不完全な実装に対する実装支援

5 節に再利用による実装生成例をあげた. この例では `removeUser`, `removeCopy`, `returnCopy` の 3 操作を自動実装することが出来なかった. 本稿では割愛しているが, 生成された実装に足りない実装を追記することで, これらの操作を実装することが出来た. 本手法では生成された実装中の変数名を入力仕様の変数名で置換することによって, その可読性を向上させている. これによって上記 3 操作の追記では, 実装のどの変数にどの様な処理を行えば良いかが容易に判断できた. 具体的には `removeUser` の追記では変数 `users` に対する削除処理を実装するのだが, これは `users` に対応する変数 `users_I.object` に対する削除処理を記述すれば良いことが分かる.

また, この作業の過程で不完全な実装に対する追記では '実装出来なかった細分化仕様の操作と制約式に現れる変数' だけを用いて実装できるという予想を得ている. この予想が正しければ実装に必要な変数を特定し, 記述者に示すことが可能となる.

7 おわりに

本稿ではソフトウェア部品の仕様とその実装に B Method を用いることで部品の信頼性を保証し, 部品再利用により入力仕様を満たす実装を得ることが出来ると考え, その実例を示した. この結果,6 つの操作を持つシステムにおいて 3 つの操作を自動生成し, 残り 3 つの操作も追記によって実装することができた. しかし, 提案手法が常に正しい実装を生成できる保証は無いため, B Method の証明責務から手法の再構築を行う必要がある. また, 異なる部品が同じ細分化仕様を持つときに, それらが同じ機能を持っているかどうかを判定する手法の提案が必要である. また, 実装支援についても証明責務に基づいた実装に必要な変数の特定手法の提案が必要である.

参考文献

[1] J.-R. Abrial. *The B-Book.* Cambridge University Press, 1996.

[2] E.Sekerinski and K.Sere. *Program Development by Refinement.* FACIT. Springer, 1999.

[3] 来間啓伸. B メソッドによる形式的仕様記述. トップエスイー実践講座, No. 1. 近代科学社, 2007.

[4] 福安直樹, 山本晋一郎, 阿草清滋. 細粒度リポジトリに基づいた case ツール・プラットフォーム sapid. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1990 – 1998, 1998.