5J－12

射影演算に基づくオブジェクト合成における仕様の変更可能範囲*

兼 英樹† 二木 厚吉†

北陸先端科学技術大学院大学

1 はじめに

ソフトウェアの仕様の変更には制限があるため、現状のオブジェクト指向プログラミング [5] などではプログラマが予めある程度の適応範囲を与えることで変更要求に対応している。しかし、これでは予期しない仕様の変更要求には十分に対応できないため、大幅な変更を余儀なくされるケースが多い。

一般に、ソフトウェアの要求が変化した場合にはその仕様を変更し、変更された仕様において要求が満たされていることを証明しなおす必要があるが、もし要求の変化がどのコンポーネントに影響し、そのコンポーネントを他のコンポーネントと置き換えた際にどんな性質を検証すれば全体として全ての要求を満たすことができるのかが分かれば大幅なコスト削減が期待できる。要求分析をどのように行うのか、それに基づいてどのように仕様化するのか、さらに要求の変化にどのように対応するのか、といった問題を形式的に行おうという研究は、ほとんど行なわれていない。

合成されるオブジェクトの仕様の変更可能範囲を形式的に表現できれば、最小限のコストで仕様を変更できることが予想される。

そこで本論文では合成対象のオブジェクトの仕様コードと振る舞い等価 [2] の証明が再利用できる射影演算 [3] を用いたオブジェクト合成において、合成されたオブジェクトが変更できる範囲を分析するための方法を示す。

*Evolution analysis of projection-based specification composition

†Hideki Kane,Kokichi Futatsugi
Japan Advanced Institute of Science and Technology
Graduate School of Information Science

2 CafeOBJ

CafeOBJ[1] は実行可能な代数仕様言語 [4] である。特徴は、多ソート代数、順序ソート代数、書き換え論理、隠蔽代数の任意の組合せを意味として持つことができることである。項書換えシステムをその操作的意味論として持つことで等式論理と書き換え論理の推論規則を使って仕様の検証を行なうことができる。

3 オブジェクト合成

合成対象の各オブジェクトの状態を得るために射影演算 (projection) という演算を定義する。この射影演算は合成対象となっている各オブジェクトに対して定義され、合成されたオブジェクトに対する (操作、属性) 演算が合成対象のオブジェクトに対する (操作、属性) 演算になるように置き換える演算である。射影演算を用いたオブジェクト合成では合成対象のオブジェクトの記述と振る舞い等価の証明が再利用できるため、検証における手間が大きく省かれる。

4 仕様の変更

要求の変更には大きく分けて以下の２つがある。

1，既存の機能を変更する（例えば、機能の詳細化）

2，新しい機能を追加する（例えば、インタフェースの追加）

２の場合、さらに新しい機能が既存の機能の組合せで表現できる場合、そうでない場合に分類できる。

要求の変更は通常曖昧性を含む自然言語で行われるが、ここでは代数仕様言語 CafeOBJ で表現することにより、曖昧性をなくす。

本稿で扱う状況では、CafeOBJ での要求の変更は

- method の追加
- equation の変更
- projection の張り直し

により表現するとする。

さらに、適応範囲の拡大方法としてはモジュールのパラメータ化が挙げられる。

5　仕様の変更例：ATM

以下に示したATMはuser accountの追加、削除、user accountに対する預入、払戻、残高照会が行なえる。

```
mod* ATM {
  protecting (COUNTER* *{ hsort Counter -> Account,
            op init-counter -> init-account,
            op counter-not-exist -> no-account })
  *[ AccountSys ]*

  op init-account-sys : -> AccountSys
  bop add : UId Nat AccountSys -> AccountSys
  bop del : UId AccountSys -> AccountSys
  bop deposit : UId Nat AccountSys -> AccountSys
  bop withdraw : UId Nat AccountSys -> AccountSys
  bop balance : UId AccountSys -> Nat
  bop account : UId AccountSys -> Account

  vars U U' : UId
  var A : AccountSys
  var N : Nat

  eq account(U, init-account-sys) = no-account .
  ceq account(U, add(U', N, A)) =
       add(N, init-account(U))     if U == U' .
  ceq account(U, add(U', N, A)) =
       account(U, A)               if U =/= U' .
  ceq account(U, del(U', A)) =
       no-account                  if U == U' .
  ceq account(U, del(U', A)) =
       account(U, A)               if U =/= U' .
  ceq account(U, deposit(U', N, A)) =
       add(N, account(U, A))       if U == U' .
  ceq account(U, deposit(U', N, A)) =
       account(U, A)               if U =/= U' .
  ceq account(U, withdraw(U', N, A)) =
       add(-(N), account(U, A))    if U == U' .
  ceq account(U, withdraw(U', N, A)) =
       account(U, A)               if U =/= U' .
  eq balance(U, A) = read(account(U, A)) .
}
```

このATMに他のuser accountへ送金するという新しい機能を追加する。

この場合、出金と入金という既存の機能の組合せで表現できるため、methodとequationを追加する方法では、以下の記述を追加すれば良い。

```
bop remit : UId UId Nat AccountSys -> AccountSys
 eq remit(U, U', N, A) =
     deposit(U',N, withdraw(U, N, A)) .
```

ただし、一般にオブジェクトにmethodを追加すると、そのオブジェクトを合成して作成されるオブジェクトは全て変更する必要がある。

例えば、このATMのオブジェクトを合成対象にしているオブジェクトがあれば、当然変更の影響を受けるが、追加したmethodやequationに対する影響を

1: 直接、methodやequationを変更することで対応させる

2: パラメータ化して輸入することで対応させる

3: 射影演算をうまく定義することで対応させる

方法が考えられる。

1の方法だとスタイルに気をつけないと検証が困難になるが、機械的に変更できる可能性がある。2,3の方法は(1に比べて)変更箇所が少ないが、機械的に変更するのは難しいことが予想される。

6　まとめ

要求の変化を分類し、その変化がオブジェクトへ与える影響をATMの例を用いて説明した。この分析を進めれば要求の変化に対して、最小限の変更が行なえることが予想される。

また、形式仕様を使う利点の一つに検証可能であることが挙げられるが、システムが大きくなればなるほど検証は困難になる。本論文の方法では、合成対象のオブジェクトにおいて証明されたことが合成後のオブジェクトにおいて再利用可能であるので、検証における手間が大きく省かれていることになる。

参考文献

[1] R. Diaconescu, K. Futatsugi. CafeOBJ Report, World Scientific, (1998).

[2] J. Goguen and G. Malcolm. A Hidden Agenda, Report CS97-538, April 1997.

[3] S. Iida, M. Matsumoto, R. Diaconescu, K. Futatsugi, and D. Lucanu. Concurrent Object Composition in CafeOBJ, Reprot IS-RR-98-0009S

[4] 二木厚吉, 代数モデルの基礎. コンピュータソフトウェア pp4-22 Vol.13, No1 (1996). ソフトウェア科学会 (1996).

[5] P.Coad,E.Yourdon 著, 羽生田訳, オブジェクト指向分析(OOA)第2版, プレンティスホール/トッパン,1993.