社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

インタフェースの provide-require 関係の解析に基づいた自動的な構成管理手法の提案

早瀬　康裕† 　神谷　年洋†† 　松下　誠† 　井上　克郎†

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻　〒 560–8531 大阪府豊中市待兼山町 1–3
†† 独立行政法人 科学技術振興機構 さきがけ　〒 560–8531 大阪府豊中市待兼山町 1–3
E-mail: †{y-hayase,kamiya,matusita,inoue}@ist.osaka-u.ac.jp

あらまし　ソフトウェアシステムは，複数のサブシステムから構成するのが一般的である．しかし，それぞれのサブシステムが独立に進化するような状況においては，サブシステムを組み合わせたときにシステムが正しく動作するかどうかを確認するのは難しく，ソフトウェア開発において問題となっている．そこで， Java 言語を対象として，オブジェクトのインタフェース情報と，データフロー解析を利用して，サブシステムの互換性を計算する手法を提案する．
キーワード　互換性，インタフェース，provide-require 関係, データフロー

A Proposal of Automatic Configuration Management with Provide-Require Relation on Interface

Yasuhiro HAYASE†, Toshihiro KAMIYA††, Makoto MATSUSHITA†, and Katsuro INOUE†

† Graduate School of Information Science and Technology, Osaka University
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
†† Japan Science and Technology Agency
1–3, Machikaneyama-cho, Toyonaka-shi, Osaka 560–8531, Japan
E-mail: †{y-hayase,kamiya,matusita,inoue}@ist.osaka-u.ac.jp

Abstract　It is a common practice to develop a software system as a collection of sub-systems. In case of each sub-system being independently evolving, the developers have to confirm it will work well in company with the other sub-systems, which is a difficult and troublesome task in a development process. This paper presents an automatic analysis for Java program to calculate compatibility among sub-systems from interface and data flow of objects.
Key words　compatibility, interface, provide-require relation, dataflow

1. はじめに

ソフトウェアシステムは一般に，関数やクラス，モジュールといった部品（以降,「サブシステム」と呼ぶ）の組み合わせとして作られる．現在のソフトウェア開発においては，これらの部品をすべて新規開発することは現実的ではなく，多くのサブシステムは再利用あるいは外部からの調達でまかなわれる．このようなとき，サブシステムはソースコードが付属せず，バイナリのみで提供されることも多い．このようにして調達したサブシステムを組み合わせて作成されたシステムの動作を検証する際，システムが意図通りに動作するか確認する作業には多くの労力が必要とされる．

さらに，サブシステムは独立に進化するため，サブシステムの新しいバージョンがリリースされる度に，作業を行わなければならない．近年では，セキュリティ問題の修正や，機能追加の要求の高まりなどにより，サブシステムの進化は早まっており，リリース間隔も短かくなる傾向にある．そこで，我々は，Java 言語 [1] [2] を対象として，サブシステムの互換性を確認する手法を提案する．提案する手法では，メソッドのシグネチャの不整合があった場合に，それによって起きる問題と，その原因を検出することができる．

2. 問　　題

サブシステムの組み合わせとして作られたシステムは，サブシステムのバージョンが上がってメソッドが無くなったり，サブシステム間で想定しているインターフェイスが異なる場合に，うまく動作しなくなる．このような問題を， Java 言語を例に用いて説明する．

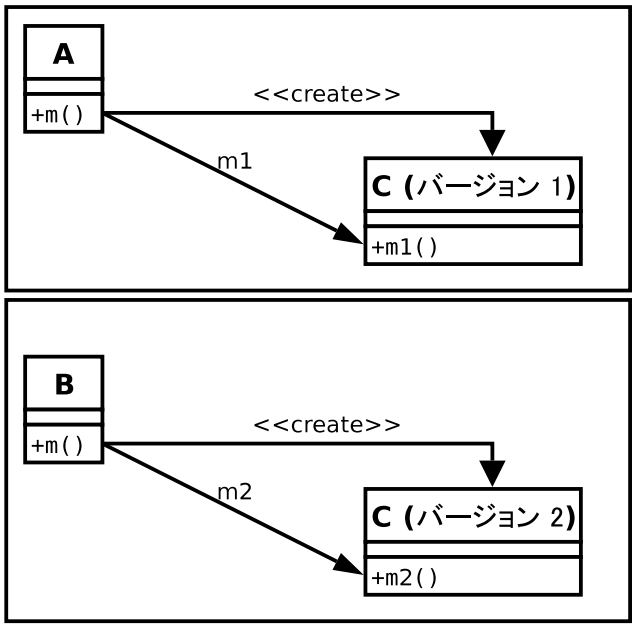


図 1　A と B は異なるバージョンの C を要求する

本節の問題の説明では，メソッドが無くなった場合を想定しているが，メソッドのシグネチャ（メソッド名前，仮引数の型，戻り値の型）の変更を，メソッドが無くなって新しいメソッドが作られたと見なすことで，これらの問題はシグネチャが変更された場合にも一般化することができる．

2.1　例 1: 相反する要求

それぞれ別の開発者によって作られているクラス A B C から成るシステムがあったとする．

クラス C には二つのバージョン 1，2 があり，異なるメソッド m1 m2 を実装している．クラス A は，クラス C のバージョン 1 で実装されているメソッド m1 を要求する．(図 1) 同様に，クラス B は，クラス C のバージョン 2 で実装されているメソッド m2 を要求する．(図 1) クラス A と B を同時に使用しない場合には，クラス C の適切なバージョンと組み合わせれば，問題は起こらない．

ここで，更に別の開発者が，クラス A と B を同時に利用したいと考えたとする．しかし，図 2 のように，C のバージョン 1 またはバージョン 2 のどちらを用いたとしても，クラス A B の要求を同時に満たすことは出来ない．このため，部品 A と B を同時に用いることは出来ない．

2.2　例 2: 失われたメソッド

それぞれ別の開発者によって作られているクラス A B C から成るシステムがあったとする．クラス A のメソッド `main` は，クラス C のインスタンスを作成し，そのインスタンスを引数として，クラス B のメソッド `m` を呼出す．B のメソッド `m` は，引数である C のインスタンスのメソッド `obsoleteMethod` を呼出す．この組み合わせで作られたシステムは，問題なく動作する．

ここで， C のバージョン 2 がリリースされたとする．C のバージョン 2 では，メソッド `obsoleteMethod` は無くなった．図 3 のシステムの C のバージョン 1 を，単純に C のバージョン 2 で差し替えると，図 4 のようになる．このシステムでは，B の要求する C のメソッド `obsoleteMethod` が存在しないため， B のメソッド `m` が C のメソッド `obsoleteMethod` を呼出そうとした時点で MethodNotFoundException が生成され，正しく動作することはできない．

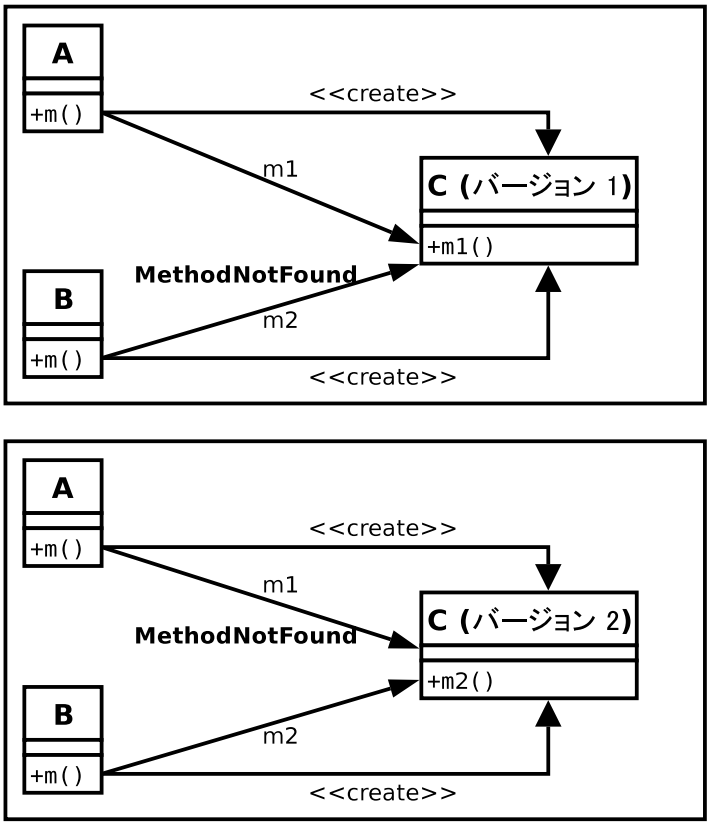


図 2　相反する要求

2.3　例 3: 実行されないメソッド

それぞれ別の開発者によって作られているクラス A B C から成るシステムがあったとする (図 5).

クラス A はクラス B のメソッド m を呼出し，そこから C のメソッド m が呼出される．このシステムでは，クラス B のメソッド `neverCalled` と，クラス C のメソッド `obsoleteMethod` が存在するが，呼出されることは無い．

ここで，クラス C のバージョン 2 がリリースされた (図 6). クラス C のバージョン 2 では，メソッド `obsoleteMethod` が削除されてる．

このシステムでは， B のメソッド `neverCalled` からクラス C への要求が満たされていない．しかし， B のメソッド `neverCalled` は使用されることは無いため，このシステムは正常に動作する．

3.　提案手法 (オブジェクトのインタフェース情報を利用したバージョン管理)

前節で説明したような問題を，人間が管理するのは手間が大きく，システムの運用や管理の妨げとなる．特に，近年はセキュリティ上の問題などにより，サブシステムが頻繁に更新されるため，大きな問題となっている．

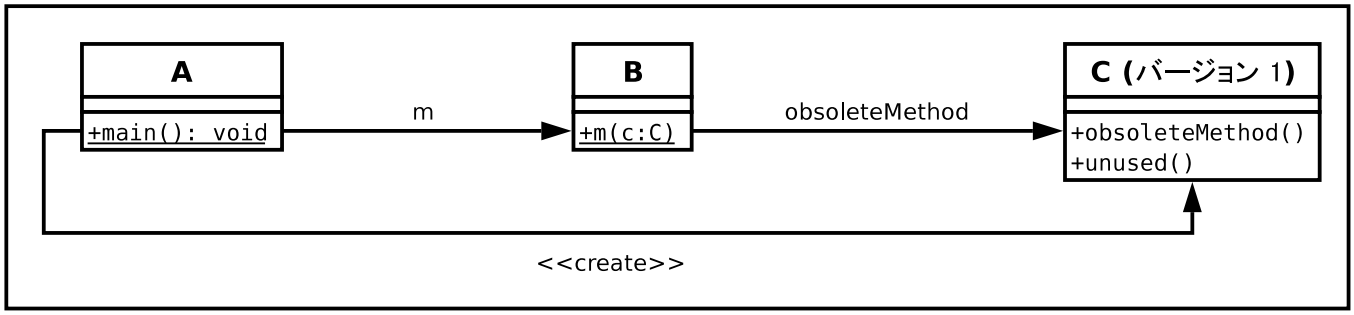


図 3　C バージョン 1

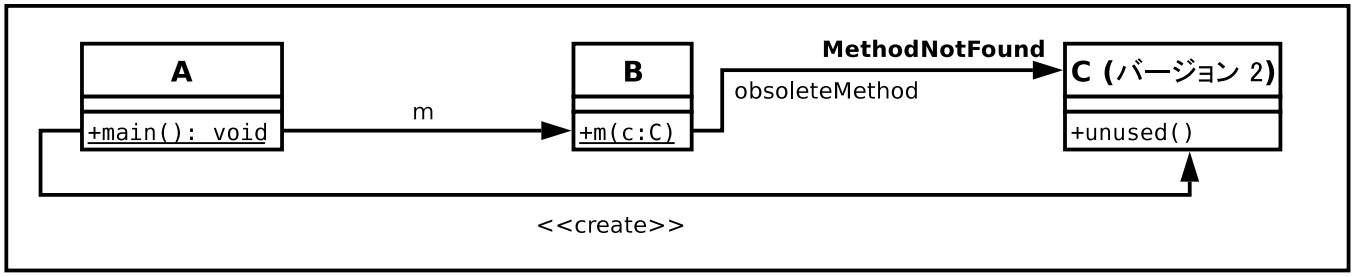


図 4　C バージョン 2

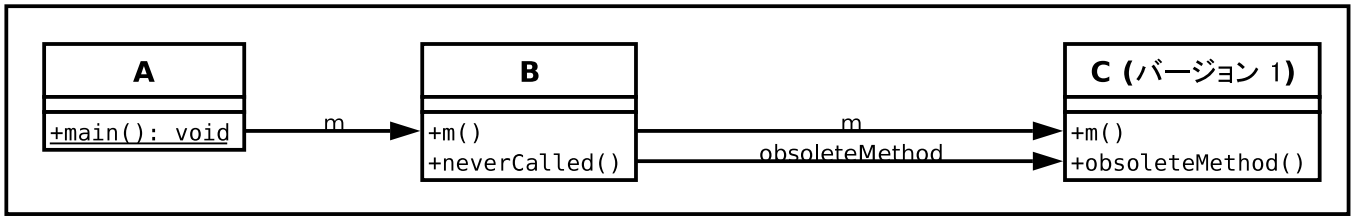


図 5　C バージョン 1

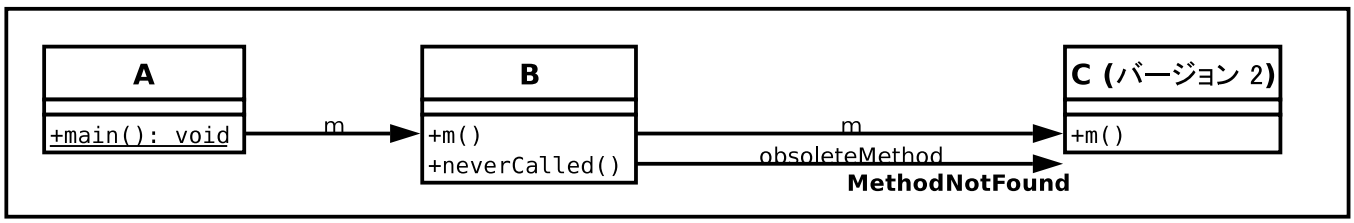


図 6　C バージョン 2

この問題を解決するために，サブシステムを組み合わせてシステムを作る場合ににシステムが動作するかどうかを自動的に検査するする手法を提案する．

以下，提案手法で用いられるモデルおよび解析アルゴリズムを順に説明する．この説明では，特定のサブシステムが 2 つのバージョンを持つ場合を想定しているが，3 つ以上のバージョンが存在する場合には，それぞれのバージョンの組み合わせを総当たりで調べればよい．

3.1　サブシステムの粒度

互換性を検査する対象であるサブシステムの粒度としては，システムの規模に応じて，例えば，jar ファイルなどによる配布系， package， クラス， メソッドなどが考えられる．ここでは，これらのうち最小単位であるメソッドを対象とした分析手法を述べる．一般に，より大きな粒度を対象とした分析を行うには，それらをメソッドの集合として扱えばよい．

3.2　メソッド間でのオブジェクトの受渡し

提案手法のモデルでは，メソッド，オブジェクト，クラスに以下の関係がある．

- メソッド間でオブジェクトが授受される
- あるオブジェクトはあるクラスのコンストラクタによって生成される．すなわち，あるオブジェクトは一つのクラスに属する．
- あるクラスはいくつかのメソッドを持つ．あるオブジェクトを callee とするメソッド呼び出しにより，そのオブジェクトが属するクラスが持つメソッドが実行される．もし，そのクラスが呼出されようとしているメソッドを持たなければ，MethodNotFound となり，プログラムの実行が停止される．
- クラス間には継承関係がある．すなわち， is_a による半

順序関係が存在する．

一般に， Java 言語のメソッドの中で，オブジェクトが授受されるのは以下の場合である．

（1） new (コンストラクタ) により作られる

（2） メソッドの引数として渡される

（3） メソッドの返り値として渡される

（4） 公開変数に代入される

（5） 公開変数から読み出される

このうち，(4) 公開変数への代入と (5) 読み出しについては，仮想的な setter/getter メソッドがあるものとして扱うことで，メソッドの引数と返り値の枠内で扱うことができる．

また，(1) コンストラクタもまた，静的なメソッドと同じように考えることができる．コンストラクタ以外のメソッドとの違いは，コンストラクタはオブジェクトの発生源となることである．すなわち，通常のメソッドは，自身に渡されたオブジェクト以外のオブジェクトを，他のメソッドに渡すことはないが，コンストラクタは新しいオブジェクトを作成して他のメソッドに渡すことができる．

従って，以降の説明では，コンストラクタ，メソッド，getter/setter をまとめて「メソッド」と呼ぶ．

3.3 データフロー解析

まず，実際に呼び出されるメソッドを特定し，さらに，それらのメソッドで参照されるオブジェクトが属するクラスを特定する．

実際に呼び出されるメソッドを特定しておかないと，すべてのメソッドが呼び出される可能性があるとして扱うことになり，2.3 節の例のような，実際には不要なメソッドを，必要なメソッドと判断してしまう場合がある．

まず，すべてのメソッド間の呼び出し関係（コールグラフ）を作成し，コンストラクタをオブジェクトの発生源として，コールグラフ上でオブジェクトの受け渡し（データフロー）を調べる．これにより，それぞれのメソッド中で参照されるオブジェクトがどのコンストラクタで作成されたか，すなわち，そのオブジェクトがどのクラスに属するかを特定することができる．

従って，それぞれのメソッド呼び出しについて， callee のクラスを特定することができる．一般に，そのようなクラスは複数存在する可能性があるが，簡単に説明するため，以下ではそのようなクラスが 1 つ求まる場合を想定する．

3.4 インタフェースの要求 (require)

メソッドが要求するインタフェースとは，そのメソッドの中で呼び出しているメソッドのことである．

メソッド内での全てのメソッド呼び出しについて，メソッドの名前とシグネチャ，callee の静的なクラスの組を記録する．

メソッドの要求するインタフェース R は，クラスとメソッドの組の集合で，以下のように表される．

$$R = \{(C_1, m_1), (C_1, m_2), (C_2, m_3) \cdots\}$$

3.5 インタフェースの提供 (provide)

メソッドが提供するインターフェイスとは，そのメソッドが他のメソッドに渡すオブジェクトのクラスに定義されているメソッドである．

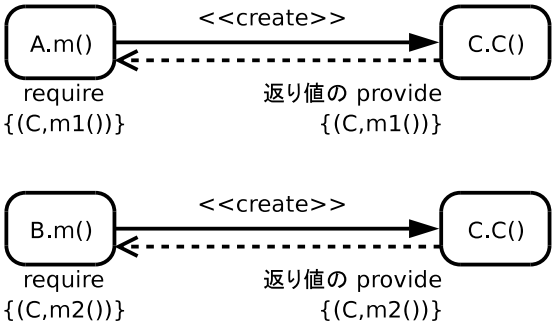


図 7 例 1 で， A と B を混在させない場合

ここで，あるメソッドは他のメソッドからオブジェクトを受け取ることがあるため，データフローを参照してそのクラスが受け取るオブジェクトを突き止める必要がある．

上述のインターフェイスの要求と同様に，メソッドが提供するインターフェイスを，クラスとメソッドの組の集合 P として表すことができる．

3.6 provide-require 関係に基づく非互換性の検出

前述のデータフローにおいて，オブジェクトを渡す側のメソッドが提供するインターフェイスは，受け取る側のメソッドの要求するインターフェイスを「満たす」必要がある．

すなわち，要求するインターフェイス R の要素（クラス C とメソッド m）について，提供するインターフェイス P が，以下の条件を満たす場合，提供するインターフェイスは，要求するインターフェイスを満たす．

$$\forall (D, n) \in P, \forall (C, m) \in R,$$
$$(D = C) \vee (D \,\textbf{is\_subclass\_of}\, C) \longrightarrow (D, m) \in P$$

データフロー内の隣接するメソッド間に，この条件を満たさないものがある場合には，そのシステムの実行中に MethodNotFound が発生する可能性がある．

4. 提案手法を使った構成管理

4.1 例 1 に適用した場合

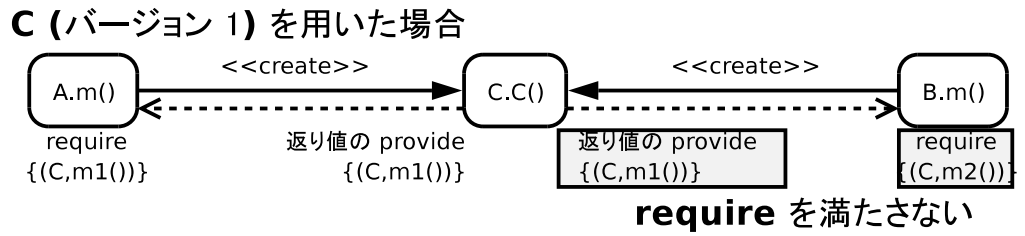
2.1 節の例 1 に，提案する手法を適用した場合を説明する．

図 7 は，クラス A B を混在させないで用いる場合の，provide-require 関係を表した図である．

頂点はメソッドを表し，実線の矢印はメソッドの呼出し関係を表している．また，点線は，メソッド呼出しによって起きるデータフローを表しており，矢印の基点に，受渡されるオブジェクトの提供するインタフェースが記述されている．

この図では，クラス A, B の要求は満たされており，MethodNotFound が発生することは無いと分かる．

図 8 は， A B を混在させて用いた場合の provide-require 関係を表した図である．この図では， C のバージョン 1, 2 のどちらを用いたとしても， A B 両方の要求を満たすことは出来ないことが分かり， MethodNotFound が起きる可能性があることが分かる．条件を満たさない部分から，データフローを



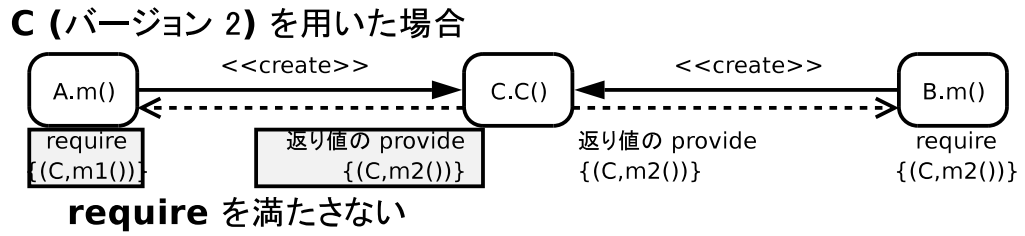


図 8　例 1 で，A と B を混在させた場合

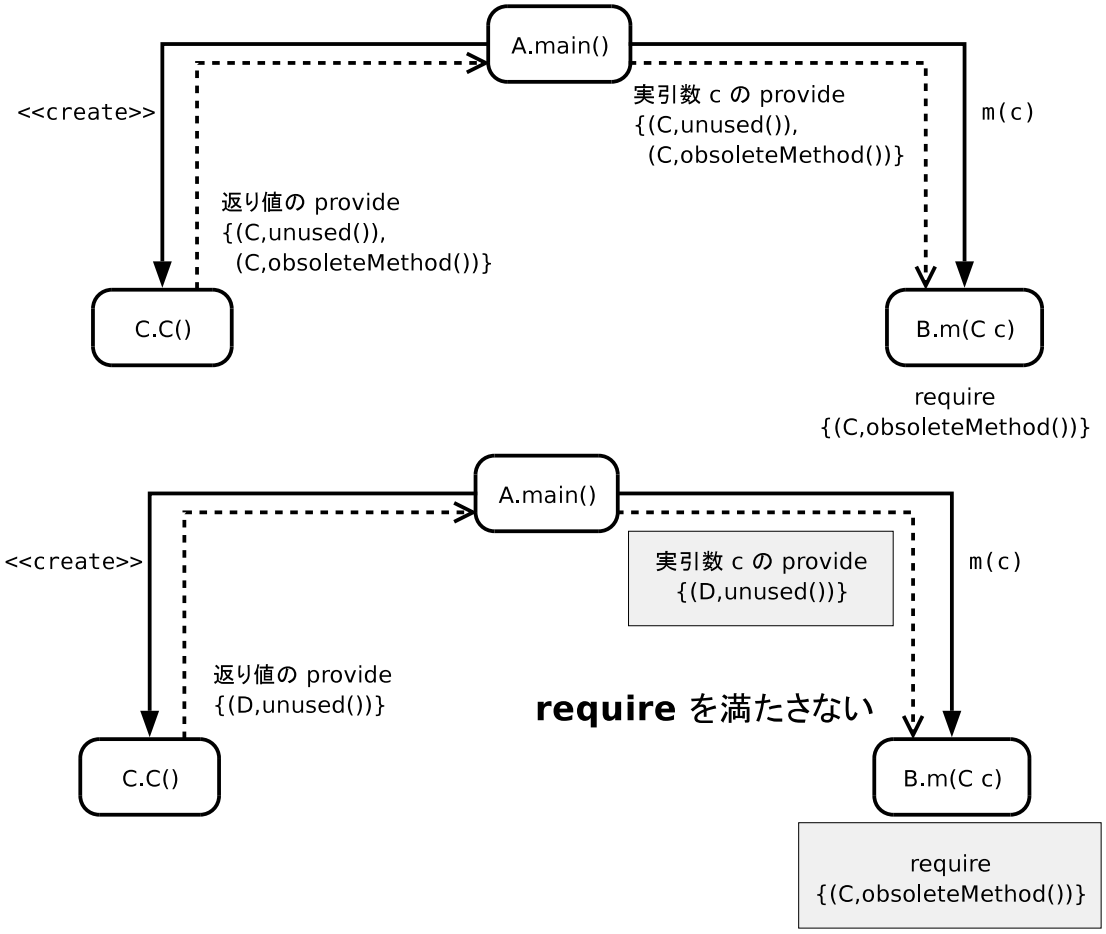


図 9　問題 2 の初期状態と C をバージョン 2 で差し替えた状態

逆にたどり，クラスのコンストラクタに至るまでの経路に含まれるメソッドが問題の原因である．

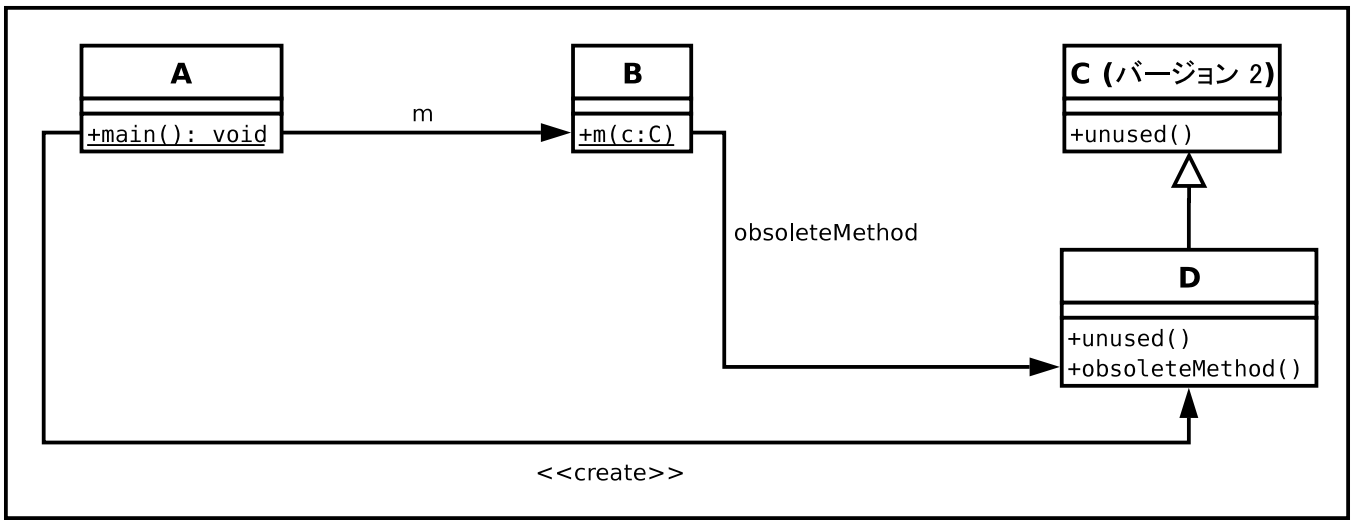
4.2　例 2 に適用した場合

2.2 節の問題は，一見，クラス B に問題があるように見える．しかし実際には，クラス A で生成されたオブジェクトが B に渡される時に，要求を満たしていない事が分かる．(図 9)

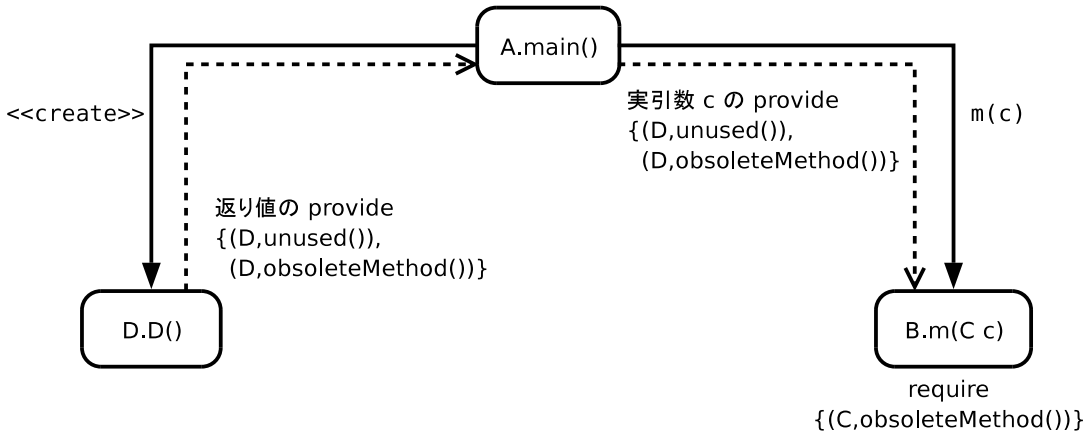
そこで，図 10(b) のように，クラス A は，C を継承したクラス D を作り，C の代わりに B に渡すことで，解決できる．

5.　ま　と　め

本論文では，Java 言語を対象として，サブシステムの互換性を確認する手法を提案した．提案した手法では，データフロー解析を利用して，オブジェクトの提供するインタフェース情報と，サブシステムの要求するインタフェース情報の関係か



(a) C を継承したクラス D で代用



(b) provide-require 関係

図 10　問題 2 の C をバージョン 2 で差し替えた状態

ら，サブシステムの互換性を計算する．

将来の課題としては，まず，手法を実装し，実際のソフトウェアシステムに適用することが挙げられる．次に，互換性の無いシステムであると判断した場合に，修正方法を自動的な提案する手法を考案したいと考えている．また，システムを検査した結果，用いられることの無いメソッドやフィールドがあると分かった場合に，それらのメソッドやフィールドを除去することが出来れば，少ない資源でシステムを動作させることが出来るだろう．

Java 以外のオブジェクト指向言語への適用も，課題の一つである．

文　　献

[1] The java language specification. http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html.

[2] The java virtual machine specification. http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.