ソフトウェア工学 92－5
（1993．5．26）

オブジェクトの独立性に関する考察

藤崎智宏　荒野高志　今瀬真
NTT ソフトウェア研究所ソフトウェア基礎技術研究部

ソフトウェア開発に用いられる方法論としてオブジェクト指向手法の有効性が認識され、広く使われ始めている。オブジェクト指向手法を支える方法論も数多く提唱されており開発に適用されているが、方法論に従うだけでは十分でなく分析 / 設計に対する評価手法が必要となる。従来のオブジェクト指向設計に関する評価は静的なオブジェクト構造に関するものが主であった。本研究では、動的側面からのオブジェクト指向設計に関する評価の指針としてオブジェクトの依存関係を定義し、分析 / 設計における生産物を評価するためのメトリクスを与える。その際に、直接メッセージを送りあっているオブジェクト間だけでなく間接的な依存関係も考慮に入れ、依存関係値を定量化する。

A study about object indenendency

Tomohiro Fujisaki　Takashi Arano　Makoto Imase
NTT Software Laboratories, Software Research Laboratory

This paper proposes a metric for object dependency from a 'dynamic' viewpoint, which focuses on the number of fixed procedural accesses to a server object and takes into account not only direct dependency but also indirect one. The metric improves the conventional object coupling metrics which simply consider the number of direct object accesses, in that it matches one's intuition better. The metric can be used to identify reusable parts from the whole system and to verify the validity of subsystem decomposition.

1 はじめに

ソフトウェア開発に用いられる方法論としてオブジェクト指向手法の有効性が認識され、広く使われはじめている。オブジェクト指向手法の適用ははじめはプログラミング言語中心であったが、最近では分析 / 設計段階からこの手法を適用することが重要であるとされており、Object Modeling Technique [RJ](以下 OMT) など分析 / 設計 / プログラミングの各段階を通したオブジェクト指向方法論も数多く提唱されている。オブジェクト指向方法論を用いた場合には従来の SA/SD などの方法論に比べ、拡張性・再利用性の高いシステムを構築することが出来る可能性があるといわれているが、単にオブジェクト指向方法論を用いて分析 / 設計を行なったからといって、必ず拡張性・再利用性に優れたシステムを構築できるとは限らない [BM]。その原因の一つは、分析 / 設計による生産物が拡張性・再利用性を持っているかを評価することが難しいためである。

拡張性・再利用性を直接測定することは出来ないので、代替の評価尺度 (メトリクス) が必要となる。従来のオブジェクト指向モデルに対する評価法は静的な、オブジェクトの関係に注目しているものが主であり、動的な観点からの評価は少ない。また、動的な観点からの評価法でも 2 オブジェクト間の直接的なメッセージのやり取りに着目しているだけである。

本研究では分析 / 設計の際の生産物を評価するためのメトリクスとして、オブジェクトの独立性 (相互依存関係) に注目する。動的な観点としてメッセージのやり取りに注目し、メッセージパッシングによる直接的依存関係だけでなく、間接的な依存関係にも着目する。設計したシステム中のオブジェクト間の直接 / 間接の依存関係を定量的に測定し、システム中からの再利用可能なオブジェクト群の切り出し (ライブラリ等に使用)、オブジェクト群のサブシステムへの分割などに利用できるメトリクスを提供する。

2 オブジェクト指向設計モデルの評価法

2.1 従来の評価法

オブジェクト指向で設計されたモデルの評価法としては Demeter System[LH] が存在する。これは、設計時のクラスの関係についての法則、すなわちクラス間の静的な関係 (OMT における object model) についての評価を提供しているのみで、動的な側面には触れていない。そのため、設計したシステム全体から一部のオブジェクト群を切り出すなどオブジェクト間の相互作用に関連した用途には使用できない。

また、メッセージパッシングに注目した動的な評価として Chandrashekar らの Coupling Measure[CR] がある。これはメッセージを送っている相手とその回数に注目した評価基準である。Coupling Measure の問題点として、メッセージを送る順序の有無に関わらず、依存度が同じ値になってしまうという点が挙げられる。メッセージを決まった順番でしか受け取らないオブジェクトとの依存度は、順番なしにメッセージを受け取るオブジェクトとの依存度より高い、と見るべきである。これは、メッセージの受信に順序関係をもつオブジェクトでは再利用・拡張を行なう際にオブジェクトが受け取るメッセージの順序関係に注意し、他のメッセージに影響がないかどうかを確認せねばならないためである。よって、メッセージ送信の回数に注目するだけでは十分でない。

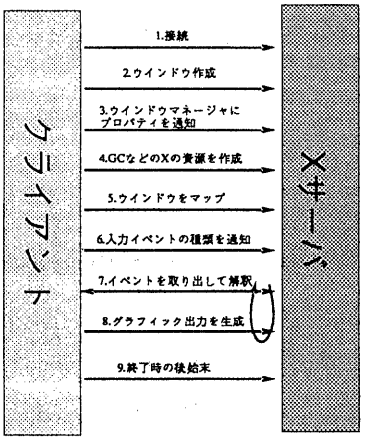

図 1: X サーバとクライアントの動的依存関係

2.2 本評価手法

本研究では、オブジェクトを呼び出す際に送るメッセージに順序が決まっているものに注目する。このようなオブジェクトの例としては、スタックオブジェクトが挙げられる。スタックオブジェクトは pop メッセージを受け取る前に、必ず push メッセージを受け取っていなければならない。このように、メッセージを受け取る順

番に規定があるオブジェクトを、呼び出したオブジェクトとの間に動的依存関係があるオブジェクトと呼ぶ。順番のあるメッセージ送受信が連続して起こる場合には動的依存関係値が高い、とする。動的依存関係の高いオブジェクト群は実際のシステムにも頻出する。電話と交換機のやりとり、X-Window のプリミティブを使ったプログラミングなどはその例である。図1に動的依存関係の高いクライアントオブジェクトとXサーバオブジェクトの間の通信例を示す。この呼び出し順序は、どんな小さなアプリケーションプログラム(クライアント)でも守らねばならない手順である。このように手順を持った呼び出しをしなければならない場合、両オブジェクト間は高い相互依存関係にある。

システムを構築する際に、上流工程での設計の良否の判断が重要である。これは、設計工程の下流になればなるほど変更はしにくくなるためである。動的依存関係値は、OMTによる設計のシステム設計時におけるサブシステムへの分割のための指標に使うことが出来る。また、設計したシステムから再利用単位を抜き出す時に利用する。動的依存関係値が低い部分でシステムから切り出しを行なうことで、設計システムからライブラリなどを抽出する際の目安となる。

3 動的依存関係

3.1 動的依存関係の評価対象

動的依存関係を検出する対象として、複数の状態遷移機械が通信しながら動作する複数状態遷移モデルを考える。各状態遷移機械がオブジェクトの動作を表現する。オブジェクトの動作を状態遷移図で表記することはOMTなどの方法論でも採用されている手法である。OMTでは状態遷移図で各オブジェクトの動作を記述するのみでオブジェクト間のメッセージパッシングについては扱っていないが、ここでは状態遷移図どうしがメッセージをやり取りする複数状態遷移モデルに拡張する。

複数状態遷移モデルの例を図2に挙げる。図2で遷移に付いているラベルはメッセージであり、各メッセージにはその送信相手 / 受信相手を特定するラベルが付加している。また、'+'記号のついたメッセージは受信メッセージを、'-'記号のついたメッセージは送信メッセージを表す。複数状態遷移モデル中に出現するメッセージの種類は以下の通りである。

- 入力信号

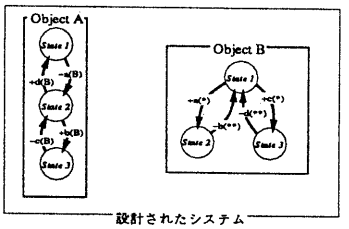

図 2: 複数状態遷移モデルの例

1. *anonymous receive*
   任意の相手からのメッセージ受信が可能であることを示す。メッセージの送信者の名前は保存され、後の送受信で参照することが出来る(その際には送信先 / 受信先の相手として'**'を指定する)。このメッセージは、サーバとなるオブジェクトを記述するために導入した。サーバオブジェクトは任意の相手からメッセージを受け取り、その相手と通信し合いながら仕事を行なう。その1仕事の単位はある *anonymous receive* から次の *anonymous receive* までと考えることが出来る。状態遷移図中では'*'と表記される。
2. *recoginized receive*
   anonymous receive で受けとった相手からのメッセージ受信を示す。クライアントとのやり取りを行なう時に用いる(状態遷移図中では'**'と表記)。
3. *specified recieve*
   相手を明示してメッセージを受信することを示す。

- 出力信号

1. *recoginized send*
   anonymousreceive で受けとった相手へのメッセージ送信を示す(状態遷移図中では'**'と表記)。
2. *specified send*
   相手を指定してメッセージを送る時に用いる。

メッセージパッシングは同期で、各状態遷移機械はキューを持たないとする。ここで、複数の状態遷移機械

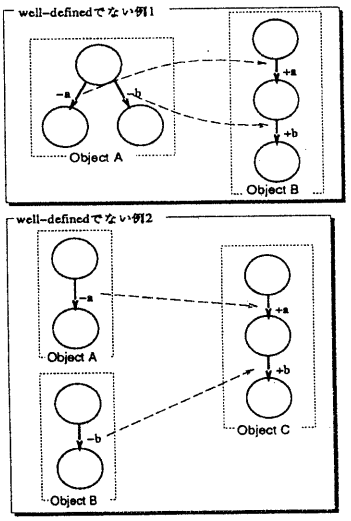

図 3: well-defined でない状態遷移モデル

からなる系において、その系が well-defined であるかどうかの判断を与える。

複数の状態遷移機械からなる系が well-defined であるとは、デッドロックに陥るようなメッセージ送受信を含まないものとする。メッセージ送信で送ることができるメッセージが複数あり、どれかを選択することにより受理されないメッセージパッシングが起こり得る場合に系がデッドロックに陥るとする。

図3において、例1では、Object A は自分の都合で -a,-b どちらのメッセージを送っても良いが、メッセージ -b を先に送ると系がそれ以上動作しなくなる。また、例2においては Object B が Object A より先に動作すると動作しなくなる。このような系は well-defined でない。

以上のような複数状態遷移モデルにおいて、動的依存関係を考える。

3.2 オブジェクト間のインタラクション

オブジェクトどうしの動的依存関係を考えるためにオブジェクト間のインタラクションについて述べる。インタラクションとはオブジェクト間のメッセージの送受信である。インタラクションには、直接、間接の 2 種類がある。

直接インタラクションとは、メッセージを直接やり取りしている場合であり、間接インタラクションとは第三者のオブジェクトを介してメッセージが到達する場合のインタラクションを示す。

3.3 インタラクションと動的依存関係

3.3.1 動的依存関係値

A の B に対する動的依存関係値を、“A から見て意識しなければならない実行可能な B のパスの長さ” とする。図1では動的依存関係値を 8 とする。

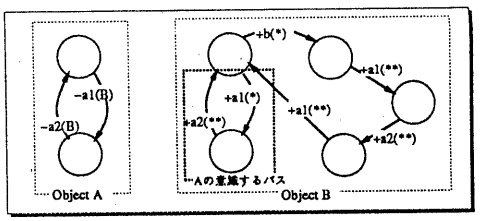

図 4: 動的依存関係値

また、図4において、Object A が意識するべき Object B 中の状態遷移は OBject B の破線で囲んだ部分であり、他の遷移は関係していない。よって、この場合の動的依存関係値を 2 とする。

3.3.2 直接インタラクション

システム中の 2 オブジェクトがメッセージを直接やり取りしている場合について動的な依存関係を考える。図5はオブジェクト数 2、メッセージの種類 2 の時、オブジェクト A が送信者、オブジェクト B が受信者の場合のメッセージの送受信パターンを表している。メッセージの種類が増えた時もこの変形として扱うことが出来る。

pattern 1 はオブジェクト A、オブジェクト B ともにメッセージの送受信に順序がないインタラクションの例である。この場合オブジェクト A が意識する B のパスの長さは 1 であるので A の B に対する動的依存関係値は 1 となる。

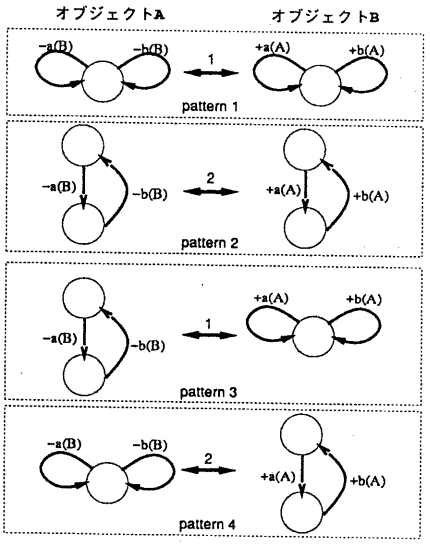

図 5: 直接インタラクションと動的依存関係値

pattern 2 はオブジェクトAとオブジェクトBとのメッセージの送受信に明確な順番が決まっている場合である。この場合、AのBに対する動的依存関係値は2となる。

pattern 3 はオブジェクトBの受信するメッセージの順序が決まっていない例である。この場合、オブジェクトAでのメッセージ送信の順番はa,b どちらを先に送出しても良く、状態遷移図上での順序関係はオブジェクトAの都合により決められている。この場合にはAのBに対する動的依存関係値は1となる。

pattern 4 はオブジェクトAとオブジェクトBとからなる系がwell-definedでない例である。この系ではオブジェクトAの送るメッセージによっては、系が動作しなくなってしまう。この場合にはAのBに対する動的依存関係値は2となる。

3.4 間接インタラクション

設計された複数状態遷移モデルから直接インタラクションだけを抜き出すのならば問題は単純であるが、間接的に関係を持っているオブジェクトもモデルの中に存在する可能性がある。間接的に依存関係を持っているオブジェクトの例を図6に示す。このような例は、ディスパッチャの役目を果たすオブジェクトを介して通信し合うオブジェクトの間に見られる。

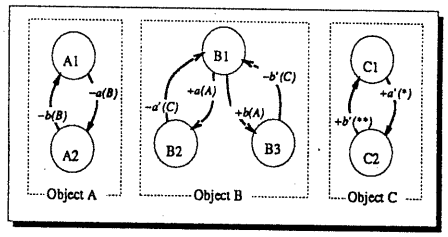

図 6: 間接的な動的依存関係

図6において、第3.3.2節で述べた直接インタラクションの考え方のみではObject AのCに対する動的依存関係値は0となるが、Aの持つメッセージの順序関係はオブジェクトCがメッセージをa,bの順序でしか受け取らないために起こってる。このような場合にはAとCが依存関係を持つと考えねばならない。Aが意識するべきCの持つパス長は2であるので、A,C間の動的依存関係値は2となる。よって、システム中のオブジェクト間の動的依存関係値は表1のように見るべきである。

	A	B	C
A	-	1	2
B	-	-	2
C	-	-	-

表 1: 間接依存の例の動的依存関係値

4 動的依存関係値の検出

4.1 設計モデルからの依存関係の検出

直接、間接インタラクションを複数状態遷移モデルから検出する方法について説明する。例として3オブジェクトA,B,Cからなるモデル図6を考える。

図6において、オブジェクトCが他のオブジェクトに対して持つ依存度を検出する。

まず、Object Cにおいてセッションとなるパスを検出する。セッションとはクライアントから要求されたある一つの仕事をするために必要なパスで、状態遷移図上ではあるanonymous receive('*'による受信)から次の

anonymous receive までの間もしくは初期状態から再び初期状態に戻るまでの間のパスの部分列となる。図6では、Object C のパス C1-C2 が1セッションになる。

次に、C のセッション中の遷移を引き起こすメッセージを送っているオブジェクトを探す。各オブジェクトでは送信メッセージに送信相手が明示されているので、システム中の全オブジェクトについて該当するメッセージ送信を検索できる。図6では Object B が Object C に対してメッセージを送っている。メッセージを送っているオブジェクトが見つかったら、そのオブジェクト(図6では B) の状態遷移図上で Object C にメッセージを送っている以外の部分を縮退する (図7)。縮退したオブジェクトを B' とする。メッセージを送っているオブジェクトが複数ある場合には、それぞれのオブジェクトを同様に縮退する。Object B' 中の点線部分は C に関係するメッセージを送出するまでに何らかの遷移を行なっていることを示す。

こうして出来たオブジェクト群を一つの系として、その系が well-defined になっているかどうかを考える。この場合には B', C が一つの系となる。系が well-defined になっていればインタラクションは系中に閉じていることになる。この例では、B',C のみで well-defined になっているとすれば C の持っているメッセージの順序に関係するオブジェクトはその系中にあることになるため、C と系中にある他のオブジェクト間で依存関係を考慮すれば良い。縮退後のオブジェクトを含んだ系が well-defined になっていない場合には、系中のオブジェクトに順序付けをしているものが他にあるということで、縮退したオブジェクト群にメッセージを送っているオブジェクトを含めて新たな系を作成する。

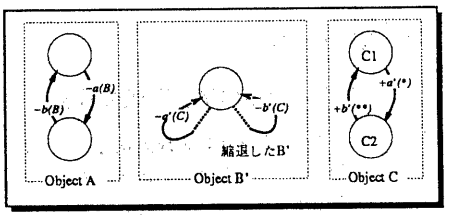

図 7: 状態遷移モデルからのインタラクションの抽出

図7の例の場合、B',C だけの系では well-defined になっていない。これは、B' におけるメッセージ送信が他のオブジェクトによるメッセージ受信と関連して行なわれているためである。このような場合、C に送信するメッセージの順番を規定する原因となる分岐点におけるメッセージ送信者まで考えなければならない。図7の場合では、点線の遷移部分の分岐の原因となるメッセージを送っているオブジェクトによって動作パスが決められる。この場合には Object A が分岐に関連したメッセージを送っている。そのため、A まで系に入れてインタラクションを考える。A の状態遷移図を B' の分岐に関連する部分だけを残して縮退し (これを A' とする)、A',B',C からなる系を考える。

このようにして、注目するオブジェクトから生成する部分系が well defined になるまで部分系を拡張していき、well defined になった時点でその系中の他のオブジェクトとのインタラクションを考える。図7では A まで含めると系が well-defined になるため、この系中で C に対する依存関係を考えれば良い。

4.2 動的依存関係値の決定

図7において、各オブジェクト間の動的依存関係値を求める。

Object A の B に対する動的依存関係値と B の C に対する動的依存関係値は、メッセージ送受信が直接インタラクションであるため、第3.3.1節の方法によって求めることが出来る。

A の C に対する動的依存関係値を考える。縮退した後の A では、状態遷移図上に残っているメッセージ送信はすべて Object C の状態に依存したものになっている。このため、縮退した A の状態遷移図のパスの長さが C に依存したものになる。

よって A,C 間の動的依存関係値は C の2となる。

系中の各オブジェクト間の依存関係値は、表1のようになる。

5 設計と動的依存関係

5.1 動的依存関係値を下げるオブジェクト

図8のようなシステムを考える。

図8中の Object B は Object C の持つ動的依存を減らすために挿入されたオブジェクトである。一般的に、"ライブラリ" と呼ばれるものはこの例のように下部にあるオブジェクト (C) の持つ複雑さをユーザ (A) から隠し、利用しやすくするために構築される。

この例の場合、Object C を直接使おうとすると4の手順を踏まねばならないが、Object B を使えば2の手

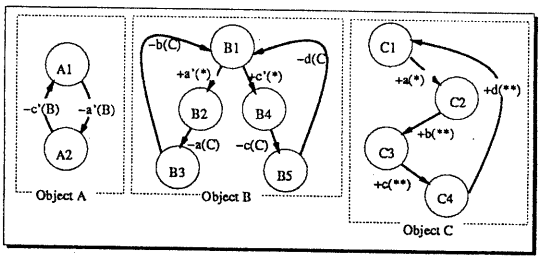

図 8: 動的依存関係を下げるオブジェクトの例

順で済む。第4.1節の方法にしたがって決定した各オブジェクトの動的依存関係値は表2のようになる。

	A	B	C
A	-	1	2
B	-	-	2
C	-	-	-

表 2: 依存関係値

5.2 オブジェクトの動作の記述法

オブジェクトが受け取るメッセージに順番が規定されている場合には、図8中の Object C のように、設計情報としての状態遷移モデルにおいてもメッセージの順序関係を明示してあることを仮定する。

オブジェクトの動作をどのように記述するかは設計者に任されているため、設計者は図8中の Object C を設計する際に、図9のように動作を記述することもできる。

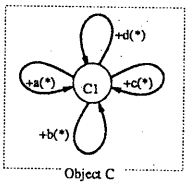

図 9: オブジェクト C の別の設計

Object C が図9のように記述されているとすると、C に対する他のオブジェクトの動的依存関係値は1になってしまう。

これは、オブジェクト指向のライブラリの利用において起こり得る問題点を含んでいる。現状では、オブジェクト指向ライブラリはそのドキュメントとしてクラス全体の意味、クラスのメソッド毎の仕様だけであるものが大部分である。そのようなクラスを用いてシステムを構築する場合には、メソッド呼び出しの順番がわからず、利用が困難である。ソースコードがあったとしても、C++ などではメソッドの順序関係を調べるために、そのクラス全体を理解せねばならない。Eiffel のような言語ではメソッド呼びだしの際に、そのメソッドを実行して良いかどうか(pre-condition)、メソッド実行後、仕様通りの結果になっているかどうか(post-condition)を記述でき、順序関係がわかりやすい記述が出来るようになっている。また、OMT などでも仕様をシステムのドキュメントとして与えるよう提言している。このように、システム記述は動的依存度がわかるように設計するべきである。

6 ライブラリへの適用

既存のライブラリの動的依存関係値を調べた例を示す。評価対象として X-Window システムの C++ で記述されたインタフェース作成ライブラリである InterViews クラスライブラリを用いた。

InterViews クラスライブラリは Stanford 大学で開発された、対話的なグラフィカルアプリケーションを作るためのクラスライブラリで、現在は X-Window システム上に実装されている。今回評価の対象としたものはバージョン InterViews2.6 のライブラリと付属のアプリケーションプログラムである。

	main	World	Mailbox	Xserver
main	-	2	2	1
World	-	-	0	5
Mailbox	-	-	-	2
Xserver	-	-	-	-

表 3: mailbox 中のの依存関係値

評価のために C++ のソースコードから状態遷移図を作成した。図10に InterViews を使って記述された mailbox(電子メールの到着をユーザに知らせるプログラム)

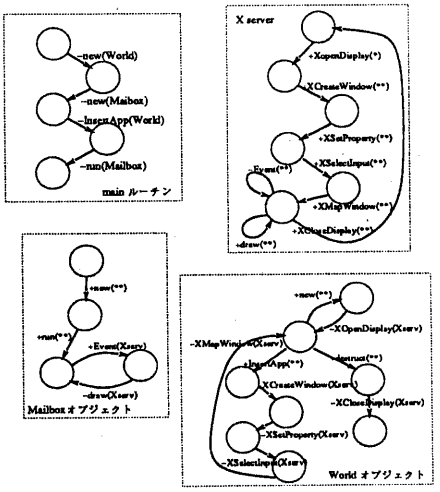

図 10: Mailbox の動作図

の動作図を示す。

第4.1節の方法による各オブジェクト間の動的依存関係値は表3のようになる。

X server に直接アクセスして mailbox と同様の機能を実現するには少なくとも8の手順を踏まねばならないが、InterViews クラスライブラリを用いた場合には1の手順ですむことになっている。InterViews クラスライブラリはライブラリとして複雑な手順を減らす、という点で成功している。

7 まとめ

従来のメトリクスで扱われていたオブジェクト間の直接的な関係だけでなく、間接的なオブジェクト間の依存関係を含め、オブジェクトどうしの動的な依存関係を定量的に決定する手段を提案した。動的依存関係値を求める際にはメッセージ送信の相手、送信の回数のみでなく、オブジェクトにメッセージを送る“手順”に注目する。

動的依存関係値はオブジェクトのサブシステムへの分割、オブジェクト群の切り出しの目安として用いることが出来る。

問題点としては、以下のようなことが挙げられる。

1. 設計に置ける動的依存関係値のさらなる考察
   動的依存値の定式化を行ない、状態遷移図上にループ構造がある時にどうするか、などの課題を解決する。

2. コードからの動的依存関係の抽出
   既存のライブラリの評価に動的依存関係を適用する。この際、コードと状態遷移表記の変換を自動的に行なえるようにする。

   表3で main ルーチンと Xserver との間の動的依存関係値が1と出ているが、main での new(Mailbox) のメッセージ、InsertApp(World) のメッセージとの間には明確な順序関係がある。これは実コード上では、new(Mailbox) の戻り値を InsertApp(World) メッセージの引数として渡しているためである。よって、main と Xserver との間の動的依存関係値は2となるべきであるが、表3の例では実コードでの引数、パラメータの受渡しによる依存関係が状態遷移図上に表現されていないためこのような値になっている。状態遷移図が無限状態を含むことを許せば実コードをほぼそのまま状態遷移表記に変換することができ、動的依存関係値を正確に求めることができるが、これは実用的ではない。動的依存関係値を正しく求められるように実コードを状態遷移表記に変換する方法について検討が必要である。

参考文献

[LH] Karl J.Lieberherr,Ian Holland. “Assuring good style for object-oriented programs.” IEEE Software,pages 38-48,September 1989.

[CR] Chandrashekar Rajaraman, Michael R.Lyu “Some Coupling Measures for C++ Programs.” TOOLS USA ’92

[RJ] Rumbaugh J, Blaha M, Premerlani W, Eddy F, Lorensen W. “Object-Oriented Modeling and Design.” Prentice Hall, 1991.

[BM] Bertrand Meyer, “オブジェクト指向入門” アスキー出版局