情報処理学会論文誌 Vol. 49 No. 7 2265–2291 (July 2008)

UML 記述の変更波及解析に利用可能な依存関係の自動生成

小谷 正行[†1] 落水 浩一郎[†1]

本論文では，UML1.5 の依存関係を解析し変更波及に有用な要因を抽出することにより，変更波及解析に有用でかつ自動生成可能な基本依存関係（情報共有，同一概念，生存従属，コピーの 4 つ）を新たに定義する．また，開発プロセスのフェーズと，各フェーズに含まれる UML 図面群を入力とし，照合規則，付加規則，選択規則から構成される依存関係生成モデルに従って，基本依存関係を自動生成する手法を提案する．基本依存関係が UML 図および，その構成要素間に付加されることによって，基本依存関係の追跡による変更波及解析が可能になる．エレベータ制御システムと ATM システムの事例研究を題材として，基本依存関係の自動生成とそれを利用した変更波及解析結果を報告することにより，本論文で提案する手法の有用性を示す．

Automatic Generation of Dependency Relationships between UML Elements for Change Impact Analysis

Masayuki Kotani[†1] and Koichiro Ochimizu[†1]

In this paper, we propose a set of new generable basic dependency relationships (BDRs) useful for change impact analysis, by analyzing dependency relationships of UML1.5 to extract effective factor for change impact analysis. We also propose a method that generates BDRs from phases of a development process and UML diagrams included in the phases. The method uses the Dependency Generation Model consisting of comparison rules, addition rules and selection rules. The change impact analysis can be realized by tracing the BDRs generated among UML diagrams and their components. Finally, we show the effectiveness of our approach based on the experimental results both on the generation of BDRs and the impact analysis using them for UML diagrams produced through case studies of elevator control system development and ATM system development.

1. はじめに

ソフトウェア開発においては，要求定義書，設計図，ソースコードなどのさまざまな中間成果物が生成される．これらの中間成果物は，開発時あるいはソフトウェアテストや保守時に，方針変更やバグなどの発生にともない頻繁に変更される．開発者は，既存の中間成果物を参照しながら新たな中間成果物を作成するため，ある中間成果物を変更したとき，それをもとにして作成された別の中間成果物を変更する場合がある．どの中間成果物を再検討すべきかを特定する作業は多大な労力をともなうことが多い．変更に関連する中間成果物は，作成された中間成果物と参照された中間成果物間に発生する依存関係を用いて特定する．UML1.5 版において，依存関係は**図 1** のように表現され，「ソースはターゲットに依存する」と読み，「ターゲットが変更されるとソースも変更される場合がある」という意味を持つ．

変更された中間成果物から依存関係を追跡することで，中間成果物の更新や削除に関する変更波及解析が可能となる．このとき，依存関係の付加作業を開発者自身で行う必要があり，不必要な依存関係の付加や必要な依存関係の見落としをする場合がある．

本論文は，UML 図とその構成要素（以下，UML 図式要素と呼ぶ）を対象に，変更波及解析に有用な依存関係を定義し，それを自動生成する手法を提案する．UML の対象版を既存システムで多く利用されている 1.5 版[1)] とする．以下，UML 図と UML 図式要素をまとめて UML 記述と呼ぶ．

2 章において，基本依存関係を定義する．基本依存関係とは，変更波及解析に利用でき，かつ，自動生成可能であるという条件を満たすものとする．ターゲットとソースの間の存在従属と共有情報に注目した分析を行うこととする．

また，**図 2** に示す手法による基本依存関係の自動生成法を提案する．図 2 において，UML 図面群とプロセス情報を入力し，依存関係生成モデルを利用して，UML 記述間の基本依存関係を出力する手法である．たとえば，クラスとそのクラスから生成されるオブジェクトの振舞いを示すステートチャート図の間に，クラスが存在しなければステートチャート図は存在しえないという関係を出力する．

本論文で提案する自動生成方式は，基本的には UML 記述間の名前の対応を利用する．す

†1 北陸先端科学技術大学院大学
Japan Advanced Institute of Science and Technology

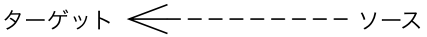

図 1　依存関係の表記
Fig. 1　A notation of dependency relationship.

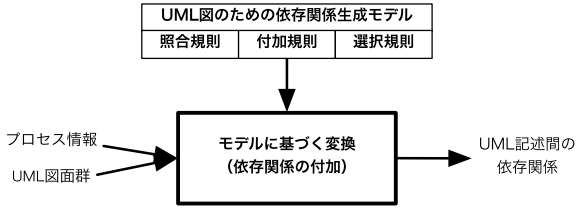

図 2　依存関係の自動生成法の概要
Fig. 2　An outline of dependency relationship generation.

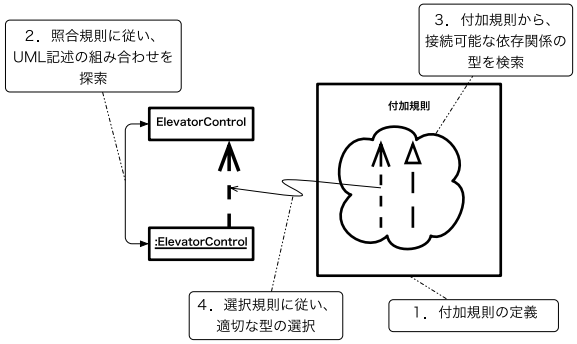

図 3　基本依存関係の自動生成の手順
Fig. 3　The procedure of basic dependency relationship generation.

なわち，同じ概念には，UML 図式要素の名前に類似した名前が用いられることを仮定する．しかし，フェーズ間にわたって名前の対応をとる場合，UML 記述にはフェーズの情報がないため，どちらがソースであるか分からない．このため，プロセス情報も入力とする．ここでプロセス情報とは，開発方法論に依存する情報であり，開発方法論を構成するフェーズ，フェーズの実行順序，各フェーズに含まれる UML 図面群を定義したものである．

自動生成の中核となる依存関係生成モデルは，照合規則，付加規則，選択規則で構成される．照合規則は，依存関係を付加できる UML 記述の組合せを取り出す規則である．3 章で定義する名前の対応付けルールを利用する．付加規則は，UML 記述間に接続可能なすべての基本依存関係を定義した規則である．基本依存関係の両端にくる UML 記述の型は開発方法論に依存するので，採用する開発方法論ごとに異なる付加規則を定義する必要がある．本論文では，Unified Process に基づく開発方法論に適用可能な付加規則を定義する．選択規則は，プロセス情報を用いて，適切な型を選択する規則である．

これら 3 つの規則を利用して依存関係を自動生成する手順を以下に示し，**図 3** に図示する．

(1)　**付加規則の定義**　基本依存関係の両端にくる UML 記述の型を，採用した開発方法論に基づき決定し，付加規則を定義する．本論文では，Unified Process にも適用できる付加規則を定義する．

(2)　**可能な UML 記述の組合せの検索**　照合規則に従って，基本依存関係が付加される可能性を持つ UML 記述のターゲットとソースの組合せを探す．図 3 では，クラス ElevatorControl とそのインスタンスの組合せが検索されたことを示す．

(3)　**接続可能な依存関係の型の列挙**　付加規則を利用し，検索された UML 記述の各組合せに付加可能な基本依存関係の型をすべて列挙する．図 3 では，雲形シンボル内にある 2 種類の依存関係の型が列挙されたことを示す．

(4)　**適切な型の選択**　選択規則に従って，(2) で得られた型から適切な型を選び，(1) の組合せに付加する．図 3 では，破線矢印の依存関係の型が選ばれたことを示す．

次章以降は，以下のように構成される．2 章では，基本依存関係を定義する．3 章では照合規則を，4 章では付加規則を，5 章では選択規則を定義する．6 章では，依存関係の自動生成法を述べ，7 章では，変更波及解析法を述べる．8 章では，実際的な例を用いて本手法の有効性を示す．9 章では UML2.0 への対応可能性を議論する．10 章では関連研究をサーベイし，本論文の成果を位置づける．11 章で本論文をまとめる．

2. 基本依存関係の定義

本章では，変更波及解析に有用で，かつ，自動生成可能な基本依存関係を定義する．

UML1.5 版では，UML 図で用いる依存関係として 13 種類のステレオタイプが定義されている．これらのセマンティクスは，UML メタモデルで定義された 4 種類の依存関係とそのステレオタイプで表現される．これらの依存関係のセマンティクスは，開発者が遭遇する状況（たとえば詳細化や仕様の実現など），UML 図式要素間の構造的関係，名前空間の公開や可視性のスコープなどに基づいて定義されている．しかし，依存関係の利用にあたっては設計者の意図が含まれるため，これらの依存関係を直接，自動生成することは困難である．

また，変更波及に関しては，多重度と OCL を利用して，さまざまな UML 図式要素間の数量的な存在条件を示しているのみである．

以下，まず 2.1，2.2 節における検討の結果得られた結論をまとめ，次に結論に至った分析過程を紹介する．

以下の 4 種類の基本依存関係は，UML1.5 版のメタモデル要素 “Dependency” を 2.1 節で分析し，また，設計者が暗黙的に認識する依存関係を 2.2 節で分析することにより得られた結果である．

(1)　**ある中間成果物が存在するために必要な中間成果物を示す依存関係**　このような依存関係は設計者の意図とは無関係に存在するため，これを自動生成することが可能である．たとえば，このような依存関係はステートチャート図と，それに対応するクラスやパッケージ間に発生する．両端要素の名前を比較することにより，この依存関係の自動生成が可能である．

(2)　**ターゲットの情報を示す依存関係**　ある中間成果物（ソース）が既存の中間成果物（ターゲット）を参照して作成されたとき，ターゲットとなる中間成果物中の関連する情報を，ターゲットの情報と呼ぶ．このとき，中間成果物間には次の 3 つの依存関係が発生する．

(a)　**ターゲットの情報がソースの情報の一部となる（すなわち，ターゲットとソースで情報が共有される）場合**　共有される情報をもとに，中間成果物より依存関係を抽出することができる．ただし，その情報は中間成果物に記述されない場合があり，その場合は，この依存関係の自動生成は困難である．しかし，(1) が成り立つ多くの場合，この依存関係も同時に成り立つ．そのため，(1) の自動生成法を利用して，この依存関係の自動生成が可能である．

(b)　**ターゲットの全情報がソースの全情報となる場合**　共有される情報をもとに，中間成果物より依存関係を抽出することができる．共有される情報に含まれる中間成果物の名前を比較することにより，この場合は自動生成が可能である．

(c)　**ターゲットの情報をもとにソースの情報が作成される場合**　異なるフェーズにある中間成果物間のトレーサビリティに対応している．この依存関係は，プロセス情報を利用して解析できる．ただし，この依存関係の両端にくる中間成果物の名前は一致しないことがあり，類似した名前などにより対応をとる必要がある．

2.1　メタモデル要素 “Dependency”

本節では，UML1.5 版のメタモデル要素 “Dependency” の 4 種類のサブクラスを分析する．

2.1.1　Abstraction

Abstraction は「異なる抽象レベルや異なる視点から同じコンセプト（概念）を表現する，2 つの要素または要素の集合を示す依存関係」と定義されている．ここで，概念は，設計者が UML 図式要素に与えた振舞いや機能を示す[⋆1]．UML 図では ≪derive≫，≪refine≫，≪trace≫，≪realize≫ を付加した依存関係として描かれる．この依存関係の特徴は「ソースは，ターゲットとは異なる立場から，概念を詳細化したものとなる」ことである．そのため，ターゲットの概念を変更した場合，ソースを見直す必要がある．これより，Abstraction における変更波及の要因は概念となる．これはターゲットの一部の情報であるが，ターゲットの概念をもとにソースの概念は作成される．このことより，Abstraction は依存関係の分類 (2)-(c) に対応する．

Abstraction セマンティクスの記述例を**図 4** に示す．図 4 は，分析モデルを構成するクラス Chessboard をもとに，設計モデルを構成するクラス Chessboard が設計されたことを示す．このように，この依存関係を自動生成する場合，プロセス情報が必要となる．ここで，我々はターゲットとソースが同じ概念を持つ場合，ターゲットとソースの名前が類似する（その逆も成り立つ）と仮定する．以上より，同じ概念には類似する名前が与えられるという前提とプロセス情報をもとに，同一概念を異なる抽象レベルや異なる視点から表現した UML 図式要素間には，依存関係 Abstraction の自動生成が可能である．

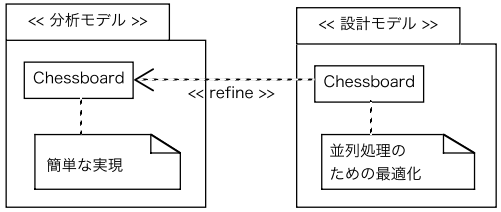

図 4　Abstraction セマンティクスの記述例
Fig. 4　A notation example of “Abstraction” semantics.

⋆1 UML 仕様書では，‘概念’ の定義が記されていないため，著者が定義した．

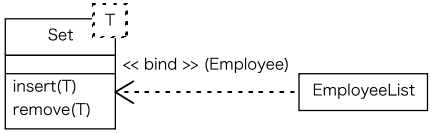

図 5　Binding セマンティクスの記述例
Fig. 5　A notation example of “Binding” semantics.

2.1.2　Binding

Binding は「テンプレート（ターゲット）と，テンプレートから作成された要素（ソース）間の依存関係であり，テンプレートパラメータに一致する属性リストを持つ」と定義されている．UML 図では ≪bind≫ を付加した依存関係として描かれる．この依存関係の特徴は「ソースは，ターゲットに属性リストの値を代入して生成された要素となる」ことである．そのため，ターゲットのテンプレートパラメータを変更した場合，ソースを見直す必要がある．これより，Binding における変更波及の要因はテンプレートパラメータとなる．これはターゲットの一部の情報であり，ターゲットとソースに共有される．このことより，Binding は依存関係の分類 (2)-(a) に対応する．

Binding セマンティクスの記述例を**図 5** に示す．図 5 は，テンプレート Set はテンプレートパラメータ T を持ち，その値として Employee を束縛したソース EmployeeList が作成されたことを示す．以上より，テンプレートパラメータがソースに記述された場合には，依存関係 Binding は自動生成できるが，非常に特殊な場合であるので通常は自動生成不可能であるとする．

2.1.3　Permission

Permission は「要素（ソース）に，他の名前空間の要素（ターゲット）へのアクセスを許可することを示す依存関係」と定義されている．UML 図では ≪import≫，≪access≫，≪friend≫ を付加した依存関係として描かれる．この依存関係の特徴は「ソースは，ターゲットへアクセスできる要素となる」ことである．そのため，ターゲット中のアクセス可能な要素を変更した場合，ソースを見直す必要がある．これより，Permission における変更波及の要因はアクセス可能な要素となる．これはターゲットの一部の情報であり，ターゲットとソースに共有される．このことより，Permission は依存関係の分類 (2)-(a) に対応する．

Permission セマンティクスの記述例を**図 6** に示す．図 6 は，パッケージ Client は，パッケージ Policies の要素を取り込む許可を持つことを示す．以上より，アクセス可能な要素がソースに記述されないので，依存関係 Permission は自動生成できない．

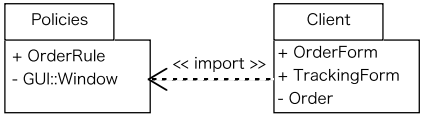

図 6　Permission セマンティクスの記述例
Fig. 6　A notation example of “Permission” semantics.

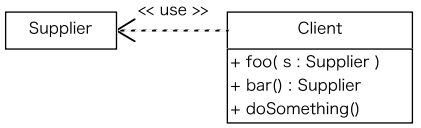

図 7　Usage セマンティクスの記述例
Fig. 7　A notation example of “Usage” semantics.

2.1.4　Usage

Usage は「ある要素（ソース）が完全な実装や操作のために，他の要素（ターゲット）を要求する依存関係」と定義されている．UML 図では ≪use≫，≪call≫，≪instanciate≫，≪create≫，≪send≫ を付加した依存関係として描かれる．この依存関係の特徴は「ソースは，ターゲットを利用して実装される要素となる」ことである．そのため，ターゲットを変更した場合，ソースを見直す必要がある．これより，Usage における変更波及の要因はターゲット自身となる．これはターゲットの一部の情報であり，ターゲットとソースに共有される．このことより，Usage は基本依存関係の分類 (2)-(a) に対応する．また，ソースはターゲットの一部の情報を用いて実現されるため，基本依存関係の分類 (1) にも対応する．

Usage セマンティクスの記述例を**図 7** に示す．図 7 は，クラス Client は，メソッド foo の引数にクラス Supplier を利用することを示す．以上より，ターゲットの名前を，ソースまたは属性またはメソッドの名前に用いた場合，依存関係 Usage の自動生成が可能である．

2.2　設計者が暗黙的に生成する依存関係

分析や設計において，さまざまなモデルを構築していく際に生成された UML 記述間の依存関係には，設計者が暗黙的に生成するものがある．そこで，前述の依存関係に加えて，設計段階に自然に発生する依存関係を 2 つ追加する．

2.2.1　コピーされたもの

たとえば，あるクラス図で描かれたクラスを，別のクラス図で利用することがある．それらのクラスには，同じ情報が描かれている必要があり，一方のクラス（ターゲット）が変更

されたとき他方のクラス（ソース）も変更される．そのため，同じ情報を持つ要素を示す依存関係が，これらのクラス間に発生したと見なせる．これより，この変更波及の要因はターゲット自身である．ターゲットの情報とソースの情報は同一である．このことより，コピーされたものの間に発生する依存関係は，依存関係の分類 (2)-(b) に対応する．

この依存関係は，同じ情報を持つ中間成果物間に発生する．ただし，コピーの関係は，同じフェーズにあり，名前と型が一致するという条件で判定するため，この条件を満たす限り内容に若干の差異があってもよい．たとえばクラスをコピーした場合，属性やメソッドに若干の違いがあっても，コピーと判定する．また，図面からどの UML 記述がオリジナルなものか判断できないため，この依存関係は双方向で設定される．

2.2.2　包含関係の記述

たとえば，クラスと内部クラス間，図面とその構成要素間のように，UML 記述間に包含関係が発生する場合がある．このとき，設計者はその UML 記述間に依存関係があると認識する．包含されるもの（ソース）は包含するもの（ターゲット）の一部であるため，ソースはターゲットを必要とする．このことより，包含関係の記述を示す依存関係は依存関係の分類 (1) に対応し，包含関係を示す依存関係の自動生成は，UML 記述の解析をもとに可能である．

2.3　基本依存関係の定義

2.1，2.2 節での分析結果を**表 1** に示す．表 1 において，左から依存関係の種類，2 章の冒頭に示した依存関係の分類，自動生成の方法を示す．たとえば，Abstraction は，依存関係の分類 (2)-(c) に対応し，開発方法論の情報を利用し，UML 記述の名前を比較することで自動生成可能である．

依存関係の各分類に対応させて，基本依存関係を定義する．以後，(1) に対応する基本依存関係を “生存従属”，(2)-(a) に対応する基本依存関係を “情報共有”，(2)-(b) に対応する基本依存関係を “コピー”，(2)-(c) に対応する基本依存関係を “同一概念” と呼ぶことにする．UML1.5 版では，ステレオタイプを利用して 13 種類の依存関係が定義されているが，我々は，2 章における議論を通じて，4 種類の基本依存関係として再定義した．

表 1　依存関係の分析結果
Table 1　Results of dependency relationship analysis.

	依存関係の分類	自動生成の方法
Abstraction	(2)-(c)	開発方法論の情報を利用し，UML記述の名前を比較する
Binding	(2)-(a)	なし
Permission	(2)-(a)	なし
Usage	(1)　(2)-(a)	UML記述の名前を比較する
コピーされた	(2)-(b)	UML記述の名前を比較する
包含関係	(1)	UML記述の包含を解析する

各基本依存関係の自動生成法を表 1 をもとにまとめる．

- **(1) 生存従属**　“生存従属” は，Usage と包含関係による依存関係から定義された．Usage と包含関係を示す依存関係は UML 記述の名前の比較や，包含の解析によって自動生成可能である．よって，“生存従属” は名前が比較可能であるかまたは包含関係が解析可能である場合に自動生成可能である．
- **(2)-(a) 情報共有**　“情報共有” は，Binding，Permission，Usage から抽出された．Binding と Permission は自動生成できないが，Usage は UML 記述の名前が比較可能である場合に自動生成可能である．よって，“情報共有” は，共有される情報群の少なくとも 1 つの名前が比較可能である場合に自動生成可能である．
- **(2)-(b) コピー**　“コピー” は，コピーされたものによる依存関係から抽出された．“コピー” は，UML 図式要素が同一フェーズ内に存在し，かつ，名前と型が一致するときに自動生成可能である．
- **(2)-(c) 同一概念**　“同一概念” は，Abstraction から抽出された．よって，“同一概念” は同じ概念には類似する名前が与えられるという前提とプロセス情報の存在を前提として自動生成可能である．

ただし，情報共有と生存従属がともに成り立つ場合，生存従属を設定せず，情報共有のみ設定する．クラス A とその振舞いを示すステートチャート図を用いて説明する．クラス A がコピーされて複数の図に存在する場合，クラス A の集合 S とステートチャート図の間に生存従属が成り立つ．これは集合 S が空集合のとき，すなわち，クラス A がすべて削除されたとき，ステートチャート図が削除されることを意味する．一方，各クラス A とステートチャート図の間に生存従属を設定したとする．これは，いずれかのクラス A が削除されたとき，ステートチャート図が削除されることを意味する．以上より，後者のような中間成果物間の依存関係を用いて，前者の意味を表現できない．そのため，情報共有と生存従属がともに成り立つ場合，生存従属を設定せず，情報共有のみ設定する．

3. 照 合 規 則

どのような UML 記述間に，基本依存関係を付加できるかは，UML 記述に直接表現され

ていない場合が多い．我々は，基本依存関係を設計者に付加させず，これらをすべて自動生成するという立場をとる．そのため，基本依存関係を付加できる UML 記述の組合せを探す規則を定義する必要があり，これを照合規則として形式化する．その手がかりとして，2 章において，基本依存関係の自動生成に利用する情報としてあげた，UML 記述の名前と，UML 記述の包含関係を利用する．これらを用いて，ターゲット（T）とソース（S）の比較する情報を示す，5 種類の照合条件を定義する．

- **Contained**　T の名前が，S の名前に含まれる．
- **Similar**　T の名前は，S の名前に似ている．
- **TypeSim**　T の型名が，S の名前に似ている．
- **SimType**　T の名前が，S の型名に似ている．
- **Include**　T は，S を包含する．

ただし「似ている」とはターゲットの名前や型名と，同じもの，複数形，語頭や語尾に何らかの単語が付加されたものを指す．

表 2 に 8 章の評価で利用する箇所の照合規則を示し，付録 A.1 にすべての照合規則を示す[⋆1]．最左列から順に，依存関係のターゲットの UML 記述の型，ソースの UML 記述の型（以下，それぞれターゲットの型，ソースの型と呼ぶ），その組合せの照合条件を示す．たとえば，ターゲットの型がユースケース図，ソースの型がアクタ，照合条件が Include のとき，これは「ユースケース図はアクタを包含する」条件を示す．

4. 付加規則

2 章で定義した 4 種類の基本依存関係は，その両端に設定可能な UML 記述の型に関して制限がある．この制限は開発方法論に依存する．4.1 節で開発方法論 Unified Process を利用した設計過程と UML 仕様を分析して，各種基本依存関係の両端にくる UML 記述の種類を分類した結果を**表 3** に示す．表 3 において，基本依存関係の両端に設定可能な UML 記述の型を生成モデル要素と呼ぶことにする．このとき，生成モデル要素間に 4 種類の基本依存関係の一部を設定することができる．生成モデル要素間にどのような基本依存関係が付加できるのかを定義した規則を付加規則と呼ぶ．4.2 節で述べる理由により，Unified Process においては，**図 8** に示すような付加規則を定義することが可能である．以下，4.1 節，4.2 節

⋆1 この規則は文献 7) および文献 8) の内容を吟味し作成した．ここで 4 番目の項目，‘ユースケース’ の名前が ‘ステートチャート図’ の名前に含まれるという関係に関しては，文献 8)「ステートチャート図は，状態および状態間の遷移という点から見て，ユースケースのインスタンスのライフサイクルを表します」という記述によった．

表 2　照合規則（一部）
Table 2　An example of comparison rule.

ターゲット	ソース	照合条件
ユースケース図	アクタ	Include
ユースケース図	ユースケース	Include
ユースケース	クラス図	Contained
ユースケース	ステートチャート図	Contained
ユースケース	コラボレーション図	Contained
ユースケース	ユースケース	Similar
アクタ	アクタ	Similar
アクタ	クラス	Similar
アクタ	オブジェクト	TypeSim
クラス図	クラス	Include
クラス	オブジェクト	SimType, Contained
クラス	ステートチャート図	Contained
クラス	クラス	Similar, Include
ステートチャート図	状態	Include
状態	状態	Similar, Include
コラボレーション図	オブジェクト	Include
オブジェクト	オブジェクト	Similar, Include

表 3　生成モデル要素
Table 3　Generation model elements.

生成モデル要素	UML記述
Classifier要素	アクタ、ユースケース、クラス、パッケージ、 ノード、コンポーネント、オブジェクト(オブジェクト図)
関係要素	関連、依存、集約、汎化、リンク
状態要素	状態、アクション状態
遷移要素	遷移、イベント、アクション
インスタンス要素	オブジェクト(コラボレーション図、シーケンス図)
メッセージ要素	メッセージ
関係図	ユースケース図、クラス図、オブジェクト図、 コンポーネント図、配置図
振舞い図	ステートチャート図、アクティビティ図
相互作用図	シーケンス図、コラボレーション図

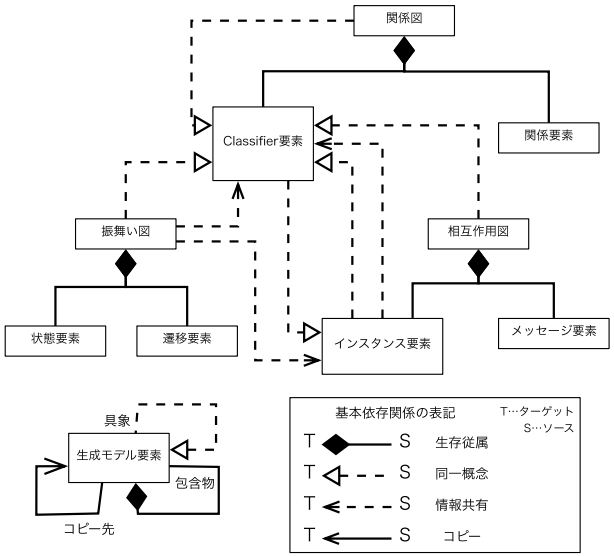

図 8　付加規則
Fig. 8　Addition rule.

において，表 3，図 8 を定義した過程を示す．

4.1　生成モデル要素の定義

Unified Process を利用した設計過程においてフェーズ間に発生する依存関係の分析例の一部を**図 9** に示す．図 9 において，長方形はフェーズを示し，その内部に作成された図の種類を示す．フェーズ間にある破線の矢印はフェーズ間の依存関係を示し，理由をコメントシンボルで示す．ただし，フェーズ間の依存関係の種類とフェーズの数は，開発方法論に依存する．

以下，図 9 を用いて，基本依存関係が設定される UML 記述の組に成り立つ，UML 記述の特性とフェーズに関する関係を検討する．その関係を満たす UML 記述の型の組を図 9 から取り出し，その型を表 3 に示す生成モデル要素として定義する．

4.1.1　情 報 共 有

基本依存関係「情報共有」は，ターゲットの情報がソースの情報の一部となることを示す．この特徴は，ターゲットとソースは何らかの ‘情報’ を共有することである．UML 記述では，‘情報’ は名前や属性，操作名などに対応する．以上より，図 9 において，「情報共有」が設定される UML 記述の組に成り立つ関係は「両 UML 記述は，同じフェーズにあり，かつ，異なる側面（型，実体）を表現すること」を示す関係である．

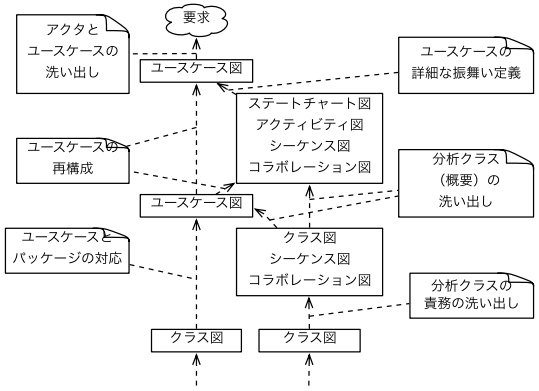

図 9　Unified Process においてフェーズ間に発生する依存関係の分析例（一部）
Fig. 9　An analysis example of dependencies among phases in Unified Process.

図 9 に基づき，この条件を満たす UML 記述の型の組を示し，型を生成モデル要素として定義する．

4.1.1.1　Classifier 要素とその振舞い

- **ターゲットとソース**　たとえば，クラスとステートチャート図のように，ターゲットが ‘役割’ でソースが ‘振舞い’ の組に発生する．‘役割’ には，アクタ，ユースケース，クラス，パッケージ，ノード，コンポーネント，オブジェクト図のオブジェクトがある．‘振舞い’ の表現には，ステートチャート図，アクティビティ図がある．
- **生成モデル要素**　‘役割’ に対応する生成モデル要素「Classifier 要素」を定義する．‘振舞い’ に対応する生成モデル要素「振舞い図」を定義する．

4.1.1.2　インスタンスとその振舞い

- **ターゲットとソース**　たとえば，オブジェクトとその振舞いを示す UML 図のように，ターゲットが ‘実体’ でソースが ‘その振舞い’ の組に発生する．
- **生成モデル要素**　‘実体’ に対応する生成モデル要素「インスタンス要素」を定義する．‘その振舞い’ に対応する生成モデル要素は「振舞い図」として定義済みである．

4.1.1.3　Classifier 要素とインスタンス

- **ターゲットとソース**　たとえば，クラスとオブジェクトのように，ターゲットが ‘型’ でソースが ‘実体’ の組に発生する．

- **生成モデル要素**　両型に対応する生成モデル要素は，4.1.1.1，4.1.1.2 において定義済みである．

4.1.2 コ ピ ー

基本依存関係「コピー」は，ターゲットの全情報がソースの全情報となる場合に発生する．この特徴は，ターゲットとソースは同一の情報を持つことである．以上より，図 9 において，「コピー」が設定される UML 記述の組に成り立つ関係は「両 UML 記述は，同じフェーズにあり，かつ，異なる図面にあり，かつ，同じ UML 記述の型であり，かつ，名前が同じであること」を示す関係である．

図 9 に基づき，この条件を満たす UML 記述の型の組を示し，型を生成モデル要素として定義する．

4.1.2.1 同一 UML 図式要素

- **ターゲットとソース**　たとえば，設計フェーズが設定される 2 枚のクラス図にあるクラス ‘ElevatorControl’ 間のように，ターゲットが ‘情報’ で，ソースが同じフェーズの同じ ‘情報’ を示す UML 図式要素の組に発生する．
- **生成モデル要素**　これはすべての UML 図式要素にあてはまる．そのため，‘情報’ に対応する生成モデル要素として，すべての UML 記述に対応する生成モデル要素，すなわち，すべての生成モデル要素の親クラス「生成モデル要素」を定義する．

4.1.3 同 一 概 念

基本依存関係「同一概念」は，ターゲットの情報をもとにソースの情報が作成される場合に発生する．この特徴は，ソースはターゲットをもとに作成され，かつ，ターゲットとソースは同じ目的を異なる情報で表現することである．以上より，図 9 において，「同一概念」が設定される UML 記述の組に成り立つ関係は「両 UML 記述は，隣接するフェーズにあること」を示す関係である．

図 9 に基づき，この条件を満たす UML 記述の型の組を示し，型を生成モデル要素として定義する．

4.1.3.1 Classifier 要素とその振舞い

- **ターゲットとソース**　たとえば，クラスとステートチャート図のように，ターゲットが ‘役割’ でソースが ‘振舞い’ の組に発生する．‘役割’ には，アクタ，ユースケース，クラス，パッケージ，ノード，コンポーネント，オブジェクト図のオブジェクトがある．‘振舞い’ の表現には，ステートチャート図，アクティビティ図がある．
- **生成モデル要素**　‘役割’ に対応する生成モデル要素は「Classifier 要素」として，‘振舞い’ に対応する生成モデル要素「振舞い図」を定義済みである．

4.1.3.2 協調の抽象と具象

- **ターゲットとソース**　たとえば，ユースケースとコラボレーション図[⋆1]の組のように，ターゲットが ‘協調の抽象’ で，ソースが ‘協調の具象’ の組に発生する．
- **生成モデル要素**　‘協調の抽象’，‘協調の具象’ に対応する生成モデル要素「相互作用図」を定義する．

4.1.3.3 非実装図の要素と実装図の要素

ただし，実装図はコンポーネント図と配置図を示し，非実装図は全種類の図から実装図を除いた 7 種類の図を示す．

- **ターゲットとソース**　たとえば，クラスとコンポーネントの組など，ターゲットが ‘非実装図の要素’ で，ソースが ‘実装図の要素’ の組に発生する．
- **生成モデル要素**　‘非実装図の要素’ と ‘実装図の要素’ に対応する生成モデル要素は，「Classifier 要素」として定義済みである．

4.1.3.4 非実装図の要素の抽象と具象

- **ターゲットとソース**　たとえば，クラス名だけ定義されたクラス図と操作のシグネチャや属性の型が定義されたクラス図の組のように，ターゲットが ‘抽象的な非実装図の要素’ で，ソースが ‘具象的な非実装図の要素’ の組に発生する．
- **生成モデル要素**　図面の抽象と具象間の依存関係は，構成要素の抽象と具象間の依存関係に置き換えて表現でき，この置き換えはすべての図で適用できる．そのため，‘抽象的な非実装図の要素’ と ‘具象的な非実装図の要素’ に対応する生成モデルとして「生成モデル要素」を用いる．

4.1.4 生 存 従 属

基本依存関係「生存従属」は，ある中間成果物が存在するために必要な中間成果物を示す場合に発生する．この特徴は，ソースはターゲットがないと存在しないことである．

2.3 節で述べたように，「生存従属」は「情報共有」とともに設定される場合がある．その場合，両端要素の型は「情報共有」に設定した型と同じになり，それは 4.1.1 項で定義された．一方，「生存従属」のみ設定される場合，2.3 節で述べたとおり，UML 記述の包含を示す場合である．以上より，図 9 において，「同一概念」が設定される UML 記述の組に成り立つ関係は「両 UML 記述は，同じ図面にあること」を示す関係である．

⋆1 UML2.0 では，コミュニケーション図と改められた．

図 9 に基づき，この条件を満たす UML 記述の型の組を示し，型を生成モデル要素として定義する．

4.1.4.1　UML 図と UML 図式要素間

- **ターゲットとソース**　たとえば，クラス図とそれを構成するクラスの組のように，ターゲットが ‘包含するもの’ で，ソースが ‘包含されるもの’ の組に発生する．
- **生成モデル要素**　‘包含するもの’ に対応する生成モデル要素は「振舞い図」「相互作用図」のほかに，複数の Classifier 要素を関連，依存，集約，汎化，リンクなどで結合した図の生成モデル要素「関係図」を定義する．Classifier 要素を結合するパスの生成モデル要素を「関係要素」と定義する．振舞い図は，状態またはアクション状態を遷移で結合したものである．状態とアクション状態の生成モデル要素を「状態要素」，その間の遷移の生成モデル要素を「遷移要素」と定義する．相互作用図は，インスタンス要素間のメッセージの流れを表現する．メッセージの生成モデル要素を「メッセージ要素」と定義する．

4.1.4.2　UML 図式要素間

- **ターゲットとソース**　たとえば，クラスと内部クラスの組のように，ターゲットが ‘包含するもの’ で，ソースが ‘包含されるもの’ の組に発生する．
- **生成モデル要素**　‘包含する’/‘される’ UML 図式要素に対応する生成モデル要素として「生成モデル要素」を利用する．

4.2　付加規則の定義

4.1.1 から 4.1.4 項における分析によって，基本依存関係の種類ごとに，その両端に対応する UML 記述の型の組を示した．以下に，それらの組をそれぞれターゲット，ソースの順に再掲する．

- **情報共有**　Classifier 要素と振舞い図の組，Classifier 要素とインスタンス要素の組，インスタンス要素と振舞い図の組．
- **コピー**　生成モデル要素と生成モデル要素の組．
- **同一概念**　Classifier 要素と関係図の組，Classifier 要素と相互作用図の組，Classifier 要素と振舞い図の組，生成モデル要素と生成モデル要素の組，Classifier 要素とインスタンス要素の組，インスタンス要素と Classifier 要素の組．
- **生存従属**　関係図と Classifier 要素の組，関係図と関係要素の組，振舞い図と状態要素の組，振舞い図と遷移要素の組，相互作用図とインスタンス要素の組，相互作用図とメッセージ要素の組，生成モデル要素と生成モデル要素の組．

以上の生成モデル要素の組に基本依存関係を設定した規則を，図 8 に図示する．図 8 において，長方形は生成モデル要素を示し，4 種類の矢印は各種基本依存関係を示す．

これらの基本依存関係が発生する UML 記述の組は，図 9 に示す Unified Process を用いて解析した．そのため，Unified Process が用いられた場合，付加規則に従って UML 図面間に基本依存関係を付加することができる．

5. 選 択 規 則

付加規則において，いくつかの生成モデル要素の組に数種類の基本依存関係が設定された．ここで，基本依存関係を UML 記述間に付加するとき，設定されたすべての種類を付加するかどうか検討する．4.1.1 から 4.1.4 項において示した，基本依存関係が設定される UML 記述の組に成り立つ関係を再掲する．

- **情報共有**　同じフェーズにあり，かつ，異なる側面を表現する（すなわち，同じ図面で型と実体の双方が使われることがない．たとえば，クラス図とオブジェクト図は別の図面として存在する）．
- **コピー**　同じフェーズにあり，かつ，異なる図にあり，かつ，同じ UML 記述の型であり，かつ，同じ名前である．
- **同一概念**　隣接するフェーズにある．
- **生存従属**　同じ図面にある．図面はいくつかのファイルに分割されていてもよい．

このように，これらの関係にはフェーズ，図面，UML 記述の型，名前を用いて排他的な基準が存在する．そのため，基本依存関係を付加する場合，設定された数種類の中から適切なものだけ付加することになる．これらの基準を用いて，各 UML 記述の組合せに適切な種類を選択する規則として選択規則を定義する．

5.1　選択規則の定義

まず，基本依存関係が設定される UML 記述の組に成り立つ関係を，4 つの基準（フェーズ，図面，UML 記述の型，名前）を用いて再定義する．基準に関する制約がないときには，特に記載しない．

- 情報共有
  - **フェーズ**　同じである．
  - **図面**　異なる．
  - **UML 記述の型**　異なる．

表 4　選択規則
Table 4　Selection rule.

<table>
<tr><td rowspan="2"></td><td colspan="4">フェーズ</td><td rowspan="2"></td></tr>
<tr><td colspan="2">同じ</td><td>隣接</td><td>離れている</td></tr>
<tr><td rowspan="4">UML記述の型</td><td rowspan="2">同じ</td><td rowspan="4">生存従属</td><td>コピー</td><td rowspan="4">同一概念</td><td rowspan="4">-</td><td>同じ</td><td rowspan="4">名前</td></tr>
<tr><td>-</td><td>異なる</td></tr>
<tr><td rowspan="2">異なる</td><td rowspan="2">情報共有</td><td>同じ</td></tr>
<tr><td>異なる</td></tr>
<tr><td></td><td>同じ</td><td colspan="3">異なる</td><td></td></tr>
<tr><td></td><td colspan="4">図面</td><td></td></tr>
</table>

- コピー
  - **フェーズ**　同じである．
  - **図面**　異なる．
  - **UML 記述の型**　同じである．
  - **名前**　同じである．
- 同一概念
  - **フェーズ**　隣接する．
- 生存従属
  - **フェーズ**　同じ図面であるため，フェーズも同じである．
  - **図面**　同じである

上記の分析結果をまとめた結果を選択規則と定義し，**表 4** に示す．表 4 において，表の上下左右部に 4 基準を示し，中央にその基準を満たす種類を示す．たとえば，フェーズが同じであり，かつ，図面が異なり，かつ，UML 記述の型が異なる場合，情報共有が選択される．ただし，種類が記されていないセルに対応する 4 基準からなる条件は，基本依存関係のどの種類の条件も満たさないことを意味する．すなわち，このセルの条件にあてはまる UML 記述の組には，基本依存関係を付加しない．

5.2　プロセス情報

上記の基準で用いた「フェーズ」の情報は，UML 図から取り出すことができない．そこで，特定の開発プロセスの各フェーズで作成された図の情報を設計者に記述させ，それを利用する．この情報をプロセス情報と呼ぶ．

プロセス情報は，**図 10** のように構成される．図 10 において，パッケージ名にフェーズの名前を，パッケージ内のクラス名に作成される図の名前を，パッケージ間の依存関係にフェーズの実行順序を示す．

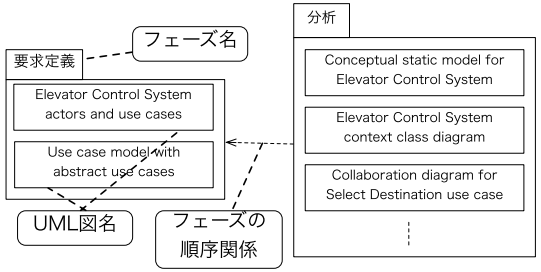

図 10　プロセス情報
Fig. 10　Process information.

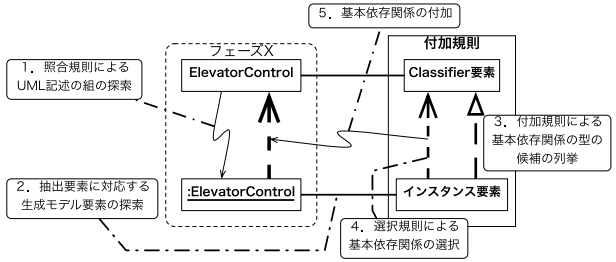

図 11　基本依存関係の自動生成例
Fig. 11　An example of basic dependency relationship generation.

6.　基本依存関係の自動生成

本章では，簡単な例を用いて基本依存関係を自動生成する過程を述べる．例として，クラス “ElevatorControl” とオブジェクト “:ElevatorControl” の間に基本依存関係を自動生成する過程を以下に述べ，**図 11** に示す．ただし，両 UML 記述とも，同じフェーズにあると仮定する．

(1)　**UML 記述の組合せの抽出**　照合規則より，ターゲットをクラス，ソースをオブジェクトとする規則「クラスの名前は，オブジェクトの名前に含まれる」を適用する．この例ではクラス名がオブジェクトの名前に含まれるため，この規則を満たす．

(2) **生成モデル要素の検索**　表 3 より，クラスとオブジェクトの生成モデル要素は，それぞれ Classifier 要素とインスタンス要素である．

(3) **基本依存関係の型の列挙**　図 8 の付加規則より，Classifier 要素をターゲット，インスタンス要素をソースとする基本依存関係の情報共有と同一概念を，両要素をつなぐ基本依存関係の候補として列挙する．

(4) **型の選択**　2 つの図面が同一フェーズで作成されたことを示すプロセス情報があるため，選択規則より「情報共有」を選ぶ．

(5) **基本依存関係の付加**　基本依存関係「情報共有」を，両 UML 記述間に付加する．

7. 変更波及解析の方法

設定された依存関係を追跡することにより変更波及解析を行う技術は，文献 2)，6)，13) などですでに提案されており，本論文でも同じ手法を採用する．クラス “ElevatorControl” の変更波及解析例を用いて，その解析方法を以下に述べる．

(1) **変更された要素から波及する UML 記述群の取り出し**　このクラスをターゲットとする依存関係を探し，そのソースの UML 記述を取り出す．

(2) **波及した UML 記述群から，さらに波及する新たな UML 記述群の取り出し**　前工程で取り出された UML 記述をターゲットとする依存関係を探し，そのソースの UML 記述を取り出す．

(3) **(2) の反復**　UML 記述は 1 度しか追跡されないとして，UML 記述が取り出されなくなるまで繰り返す．

本手法を利用して解析可能な変更波及は，論理的な波及である．変更波及は論理的な波及と，システムの性能に関する波及の 2 種類に分けられる[15]．論理的な波及とは，他の中間成果物との一貫性を論理的に保証するための変更波及であり，システムの性能に関する波及とは，システム性能の向上を目的として構造や振舞いを再構成するための変更波及である．

8. 評　　価

本章では，提案手法の有効性を評価する．題材として，Unified Process の一種である開発方法論 COMET[7] により作成されたエレベータ制御システムと ATM システムをとりあげる．エレベータ制御システムは，エレベータを 1 つのコントローラで制御する集中管理型のシステムであり，ATM システムは，ATM と銀行サーバとの連携を含む分散管理型のシステムである．これらの題材に本論文で提案した手法を適用して，自動生成された基本依存関係の精度，および変更波及解析における有用性を評価する．

8.1 評価対象の特徴

評価対象の選択にあたっては，ドメイン，アーキテクチャ，フェーズ構成，フェーズに出現する UML 図の頻度などの観点から以下の 2 つの例題をとりあげて評価する．以下にエレベータ制御システムと ATM システムの特徴をまとめる．

エレベータ制御システム

(1) **ドメイン**　センサ・アクチュエータベースのリアルタイム制御システム

(2) **アーキテクチャ**　集中制御型

(3) **フェーズ構成の特徴**　ユースケースによる機能要求定義，問題領域オブジェクトの発見と動的モデリング，サブシステム設計，タスク設計

(4) **各フェーズで利用している図面の種類と数**　ユースケースによる機能要求定義（ユースケース図 2 枚），問題領域オブジェクトの発見と動的モデリング（クラス図 2 枚，ステートチャート図 5 枚，コラボレーション図 5 枚），サブシステム設計（クラス図 1 枚，コラボレーション図 3 枚），タスク設計（クラス図 1 枚，コラボレーション図 8 枚）

ATM システム

(1) **ドメイン**　トランザクション管理

(2) **アーキテクチャ**　クライアントサーバ

(3) **フェーズ構成の特徴**　ユースケースモデルによる機能要求定義，エンティティオブジェクトに関するドメイン分析（種々のトランザクション，口座，カード情報など），アーキテクチャスタイルの決定（クライアントサーバ），クライアントサブシステムおよびサーバサブシステムに対するユースケースの割当て，サブシステムごとの問題領域オブジェクトの発見，サブシステム内部構造の詳細化，タスク設計

(4) **各フェーズで利用している図面の種類と数**　ユースケースモデルによる機能要求定義（ユースケース図 1 枚），エンティティオブジェクトに関するドメイン分析（種々のトランザクション，口座，カード情報など）（クラス図 6 枚），アーキテクチャスタイルの決定（クライアントサーバ），クライアントサブシステムおよびサーバサブシステムに対するユースケースの割当て（ユースケース図 1 枚，コラボレーション図 3 枚），サブシステムごとの問題領域オブジェクトの発見（クラス図 1 枚，ステートチャート図 6 枚，コラボレーション図 6 枚，シーケンス図 2 枚），サブシステム内部構造の詳細化（コラボレーション図 2 枚），タスク設計（コラボレーション図 2 枚）

表 5　エレベータ制御システムにおける UML 図での概念の記述（一部）
Table 5　A part of concept notations in UML diagrams of Elevator Control System.

概念	側面	フェーズ「要求定義」での名前[記述された図面No]	フェーズ「分析」での名前	フェーズ「サブシステム設計」での名前	フェーズ「タスク設計」での名前
ArrivalSensor	型	ArrivalSensor(A)[1,2]	ArrivalSensor(C)[3,4]		
	実体		:ArrivalSensor(A)[7]	:ArrivalSensor(O)[15, 16]	:ArrivalSensor(O)[22, 23, 24, 29]
ArrivalSensorInterfaceとタスク	型			ArrivalSensorInterface(C)[18]	
	実体		:ArrivalSensorInterface(O)[7, 14]	:ArrivalSensorInterface(O)[16]	:ArrivalSensorsInterface(O)[23, 24]

＊(A)…アクタ，(C)…クラス，(O)…オブジェクト

8.2　評価のための基礎データの作成と評価方式

8.1 節で述べたエレベータ制御システムの分析・設計例（27 枚の UML 図）と，ATM システムの分析・設計例（30 枚の UML 図）を対象にして，それらの UML 記述（UML 図および UML 図式要素）間に文献 7) の著者（設計者）が認識したであろう設計情報間のつながりを再現した．まず，それぞれのシステムの UML 図式要素を列挙し，それらの間の基本依存関係を我々が設定した．

(1) エレベータ制御システムの場合，UML 図式要素が 256 個，設定された基本依存関係は 1,189 個であった．ATM システムの場合，UML 図式要素が 259 個，設定された基本依存関係は 1,059 個であった．これを基礎データ 1 とする．エレベータ制御システムの結果を付録 A.2 の表 17〜表 19 に，ATM システムの結果を付録 A.3 の表 20〜表 24 に示す．表 17〜表 19 の一部を**表 5** に示す．

表 5 の読み方は以下のとおりである．たとえば，概念 “ArrivalSensor” は，フェーズ「要求定義」において，アクタ “ArrivalSensor” なる型として，図面 1，2 に記述されたことを示す．

表 5 をもとに，基本依存関係をどのように設定したかを説明する．

- 異なるフェーズのシンボル間には ‘同一概念’ の基本依存関係を設定する．
- 異なる UML 記述の型（型と実体）のシンボル間には ‘情報共有’ の基本依存関係を設定する．
- 同一フェーズのシンボル間には ‘コピー’ の基本依存関係を設定する．
- 図面とその構成要素間，および，包含関係となる構成要素間には ‘生存従属’ の基本依存関係を設定する．

表 17〜表 19 をもとに，表の最左端の「概念」に対応する，適切な UML 図式要素が適切なフェーズに存在することを確認することにより，表 17〜表 19 で設計者によって構築された分析設計結果が再現できたと考えた．ただし，本研究では，クラスなどのシンボルに依存関係を生成するので，シンボル間のパス（UML 用語であり，たとえばクラスに対する関連のように，シンボル間の結合を表す記号）の再現を省略した．

(2) さらに，最後のフェーズの設計結果（この場合はタスク設計結果）を出発点として，フェーズを逆にたどり，ある中間成果物（UML 図式要素）を作成するために，作成済みの中間成果物のどれを参照したかという参照関係を基礎データ 1 とは独立に分析した．これを基礎データ 2 とする．基礎データ 1 で分析した基本依存関係は，すべてこの参照関係に含まれたので，基礎データ 1 の妥当性も確認できたと思う．基礎データ 2 で新たに追加された参照関係は，エレベータ制御システムの場合，問題領域オブジェクトとユースケースの対応に関して 53 個，タスクとタスクを構成するオブジェクト群の対応に関して 10 個であり，ATM システムの場合，問題領域オブジェクトとユースケースの対応に関して 37 個，タスクとタスクを構成するオブジェクト群の対応に関して 6 個であった．前者は概念の分解，後者は概念の統合に関するものであった．本論文で提案する方式では，概念の分解の問題は「包含関係」で取り扱う方針を採用しているが，その適用の前提に適合しないケースである．結果として，基礎データ 2 においては，エレベータ制御システムの場合，UML 図式要素が 256 個，設定された依存関係は 1,252（1,189 + 63）個，ATM システムの場合，UML 図式要素が 259 個，設定された依存関係は 1,102（1,059 + 43）個であった．基礎データ 1 と基礎データ 2 は，本論文で提案した方式の有効性を検討するための土台として利用する．

自動生成された基本依存関係の精度および，変更波及解析への有用性は，再現率と適合率により評価した．ここで，$\#X$ を事象 X の個数として，再現率（Recall）と適合率（Precision）の式を以下に示す．

$$Recall(\%) = \frac{\#(A \cap B)}{\#A} \times 100$$

$$Precision(\%) = \frac{\#(A \cap B)}{\#B} \times 100$$

ただし，$\#A$ を設計者が認識したと思われる基本依存関係数，$\#B$ を自動生成された基本依存関係数とする．

評価は以下の 3 段階に分けて行う（**図 12** 参照）．

(1)　基礎データ 1 および 2 が現実世界の認識を近似していると仮定し，自動生成された基本依存関係が，基礎データ 1 や 2 とどの程度一致しているのかを再現率と適合率により評価する（8.3 節）

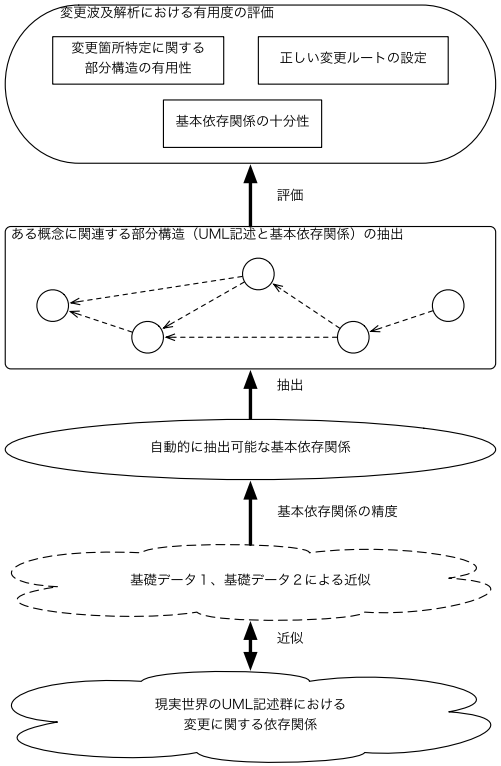

図 12　評価の内容
Fig. 12　The outline of evaluation.

(2)　本論文で提案した方式は，同じ概念には同じまたは類似の名前が与えられる（その逆も成立）と仮定している．この仮定のもとに，変更対象となる UML 図式要素の名前に類似する名前を持つすべての UML 記述を取り出し，それらの間に基本依存関係を設定することにより，変更波及解析に関与するであろう部分構造を取り出すアプローチである．概念の分解に関しては，包含関係で取り扱う．そこで，名前と包含関係を手がかりに同じ概念または関連する概念をどの程度抽出できるかを再現率と適合率により評価する（8.4 節）

(3)　抽出された構造が変更波及解析にどの程度有用であるかを，変更箇所特定に関する有用性，基本依存関係の十分性，正しい変更ルートの設定の観点から評価する（8.5 節）

8.3　自動生成された基本依存関係の精度

本手法を用いて自動生成された基本依存関係の精度を調べるため，生成された箇所と型を評価する．自動生成された 1 つの基本依存関係はその両端に，ある特定の UML 記述を持つ．両端の UML 記述と生成された基本依存関係の型を基礎データ 1 と比較し，すべて完全に一致した場合，正しく自動生成されたとする．

各題材において，自動生成された基本依存関係の調査結果を**表 6** に示す．表 6 において，正解数は設計者が認識したと思われる基本依存関係数を，ノイズ数は自動生成されたが設計者の認識外の基本依存関係数を，検索漏れ数は自動生成されなかったが設計者が認識した基本依存関係数を示す．ノイズ数，検索漏れ数に含まれる基本依存関係は，すべて基本依存関係「同一概念」であった．

基礎データ 2 の場合は正解数をそれぞれ，1,252 と 1,102 にすればよい，結果を**表 7** に示す．

ノイズ数，検索漏れ数に含まれる基本依存関係は，すべて，同一概念を示す UML 図式要

表 6　自動生成された基本依存関係（基礎データ 1）
Table 6　Generated basic dependency relationships (Base data 1).

	エレベータ制御システム	ATM システム	
正解数	1189	1059	#A
自動生成数	1333	1215	#B
検索漏れ数	20	42	#(A-B)
ノイズ数	164	198	#(B-A)
再現率	98.3%	96.0%	
適合率	87.7%	83.7%	

表 7　自動生成された基本依存関係（基礎データ 2）
Table 7　Generated basic dependency relationships (Base data 2).

	エレベータ制御システム	ATM システム	
正解数	1252	1102	#A
自動生成数	1333	1215	#B
検索漏れ数	83	85	#(A-B)
ノイズ数	164	198	#(B-A)
再現率	93.3%	92.3%	
適合率	87.7%	83.7%	

素間の基本依存関係であった．追加された参照関係はすべて検索漏れにカウントされている．

自動生成された基本依存関係は，表 6 と表 7 に示す精度を考慮にいれたうえで，利用できると判断した．

8.4　ある概念に関連する部分構造の抽出

本論文で提案した方式は，同じ概念には同じまたは類似の名前が与えられる（その逆も成立）として，類似する名前を持つすべての UML 図式要素を照合規則により取り出し，それらの間に自動的に設定される基本依存関係も含めて，変更波及解析の元データとして利用するというアプローチである．

自動設定される基本依存関係の精度については 8.3 節ですでに述べた．本節では，概念が同じものを，どの程度の精度で，照合規則が抽出できるかを評価する．また，包含関係が概念の分割・統合をどの程度取り扱えるか評価する．

ところで，抽出の精度を議論する際，評価するデータの選択は重要である．ここでは，同一の方法論で作成されたエレベータ制御システムと ATM システムを対象としており，8.1 節で説明した違いはあるにしても，必ずしも一般的とはいえない．そこで，評価の目標としては，本論文で提案する方式の一般的な有用性を主張するのではなく，その利点と欠点を明らかにするという立場をとる．

上記をふまえ，以下の 3 つの分類からなるテストケースを設定した．

(1)　概念の分解統合が存在しない，フェーズにわたる，同一概念を持つ UML 図式要素をもれなく抽出できるか．**表 8** と**表 9** の分類 (1) に対応する．

(2)　各フェーズで作成される主要中間成果物間の関係の抽出．ユースケースから始まり各フェーズで変換・詳細化されていく，主要中間成果物間の作成順序に関する情報が，どの程度正確に取り出せるか．表 8 と表 9 の分類 (2) に対応する．

表 8　基礎データ 1 および 2 を基準にした抽出精度（エレベータ制御システム）
Table 8　The extraction rate based on Base data 1 and 2 (Elevator Control System).

分類	サンプル番号	UML図式要素の名前 (UMLの型/役割)	正解	自動生成	共通	検索漏れ	ノイズ	再現率	適合率
(1)	1	ArrivalSensor (アクタ/外部オブジェクト)	10 [12]	14	10	0 [2]	4	100% [83.3%]	71.4%
	2	ElevatorButtonInterface (オブジェクト/インタフェース)	6 [6]	6	4	2 [2]	2	66.7% [66.7%]	66.7%
	3	ElevatorControl (クラス/コントロール)	48 [48]	71	48	0 [0]	23	100% [100%]	67.6%
	4	ElevatorStatus&Plan (オブジェクト/エンティティ)	9 [9]	16	9	0 [0]	7	100% [100%]	56.2%
	5	ElevatorController (オブジェクト/タスク)	8 [45]	9	8	0 [37]	1	100% [17.8%]	88.9%
(2)	6	Select Destination (ユースケース)	99 [117]	131	99	0 [18]	32	100% [84.6%]	75.6%
	7	Request Elevator (ユースケース)	100 [138]	127	98	2 [40]	29	98.0% [71.0%]	77.2%
	8	Dispatch Elevator (ユースケース)	138 [144]	177	138	0 [6]	39	100% [95.8%]	78.0%
	9	Stop Elevator at Floor (ユースケース)	154 [160]	185	150	4 [10]	35	97.4% [93.8%]	81.1%
(3)	10	FloorSubsystem (クラス/サブシステム)	23 [23]	23	23	0 [0]	0	100% [100%]	100%
	11	Scheduler (クラス/サブシステム)	26 [26]	26	26	0 [0]	0	100% [100%]	100%
	12	ElevatorSubsystem (クラス/サブシステム)	46 [46]	76	46	0 [0]	30	100% [100%]	60.5%

(3)　分析設計のある段階で定義されたある概念がその後詳細化，分解される場合，それを追跡することが可能か．表 8 と表 9 の分類 (3) に対応する．

表 8 と表 9 に，基礎データ 1 および基礎データ 2 を基準にして計算した評価値（再現率・適合率）を示す．なお，基礎データ 2 を基準にした結果は，表中に括弧書きで示す．

8.4.1　概念の分解統合が存在しない，同一概念を持つ UML 図式要素の抽出結果

表 8 と表 9 の分類 (1) は，同一概念を持つ UML 図式要素が照合規則によりどの程度の精度で抽出できるかを示したものである．

表 8 と表 9 のサンプル 1 から 5 は，以下の方針で選択した．システムの外部に存在するもの，インタフェースオブジェクト，コントロールオブジェクト，エンティティオブジェクト，タスクオブジェクトから 1 つずつ選んだ．表 8 と表 9 において，

(1)　再現率は表 8 のサンプル 2 を除き，基礎データ 1 においてはいずれも高い．

(2)　表 8 のサンプル 2 の再現率が低い理由は，動的モデリングにおける名前と，タスク

表 9　基礎データ 1 および 2 を基準にした抽出精度（ATM システム）
Table 9　The extraction rate based on Base data 1 and 2 (ATM system).

分類	サンプル番号	UML図式要素の名前 (UMLの型/役割)	正解	自動生成	共通	検索漏れ	ノイズ	再現率	適合率
(1)	1	CardReader (クラス／外部オブジェクト)	10 [11]	20	10	0 [1]	10	100% [90.9%]	50.0%
	2	Account (クラス／エンティティ)	3 [5]	7	3	0 [2]	4	100% [60.0%]	42.9%
	3	: CustomerInterface (オブジェクト/インタフェース)	10 [10]	10	10	0 [0]	0	100% [100%]	100%
	4	BankTransactionServer (オブジェクト/コントロール)	13 [13]	14	11	2 [2]	3	84.6% [84.6%]	78.6%
	5	ATMController (オブジェクト/タスク)	20 [41]	18	17	3 [24]	1	85.0% [41.5%]	94.4%
(2)	6	Withdraw Funds (ユースケース)	145 [17]	145	145	0 [17]	0	100% [89.5%]	100%
	7	Client Withdraw Funds (ユースケース)	126 [134]	126	126	0 [8]	0	100% [94.0%]	100%
	8	Server Withdraw Funds (ユースケース)	95 [95]	95	95	0 [0]	0	100% [100%]	100%
	9	Validate PIN (ユースケース)	120 [140]	120	120	0 [20]	0	100% [85.7%]	100%
	10	Client Validate PIN (ユースケース)	108 [115]	108	108	0 [7]	0	100% [93.9%]	100%
	11	Server Validate PIN (ユースケース)	96 [96]	96	96	0 [0]	0	100% [100%]	100%
(3)	12	ATMClientSubsystem (パッケージ/サブシステム)	98 [98]	90	90	8 [0]	0	91.8% [91.8%]	100%
	13	BankServerSubsystem (パッケージ/サブシステム)	26 [26]	0	0	26 [26]	0	0% [0%]	0%

設計における名前が異なっていたためである．

(3)　適合率は全般的に高いとはいえない．文献 9) には，情報検索技術に基づく再現率と適合率がどの程度になるかが報告されており，本実験でも，傾向としては同じような結果が得られている．

(4)　基礎データ 2 にした場合，表 8 のサンプル 5 と表 9 のサンプル 2 と 5 の再現率が落ちた．双方とも基礎データ 2 で新たに認識された参照関係に関与しており，複数のオブジェクトが統合されてタスクが定義されるという例である．統合に関する取扱いは現在検討中であり，今後の課題である．

以上の結果より，分類 (1) においては本方式の欠点に基づく，いくつかの例外を除いて，高い再現率と妥当な適合率が得られたものと判断する．

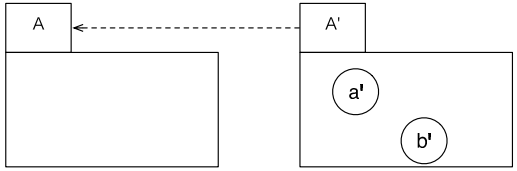

図 13　分割される概念
Fig. 13　Divided concepts.

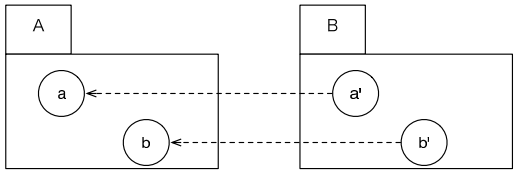

図 14　分割される概念を追跡できる場合
Fig. 14　The case of traceable divided concepts.

8.4.2　ユースケースから始まり各フェーズで変換・詳細化されていく，主要中間成果物間の作成順序に関する情報が，どの程度正確に取り出せるか

ユースケース定義から始まり，あとに続くフェーズで変換・詳細化されて定義される種々の UML 図式要素（コラボレーション図に含まれる要素，クラス図に含まれる要素，サブシステムに含まれる要素，タスク設計図に含まれる要素など）が表 8 と表 9 の分類 (2) に示す精度で抽出されており，再現率・適合率とも良い結果を示している．

8.4.3　分析設計のある段階で定義された，ある概念がその後，分解される場合

表 8 と表 9 の分類 (3) は，分解された概念に対応する UML 図式要素をどの程度の精度で追跡できるかを示したものである．

本論文で提案する方式では，分割前の概念（**図 13** の名前 A に対応）と分割後の概念群（図 13 の名前 a’，b’ に対応）の基本依存関係は，分割後の概念群をまとめる名前 A’ と分割後の概念群（図 13 の名前 a’，b’ に対応）に与えられた名前との間に包含関係を設定し，名前 A と A’ の間に照合規則で類似性が認識されたときに設定される．

我々の方式で追跡可能なのは，A と A’ が照合規則によりマッチする場合である．または，A が a を含み，a と a’ が照合規則でマッチした場合も追跡できる（**図 14**）．

表 8 のサンプル 10，11，12 と表 9 のサンプル 12 は，上記の条件が成立した場合に対応している．表 9 のサンプル 13 は，A と A’ の間に照合規則で類似性が認識されなかった場

合である．

ところで，表 8 と表 9 において分割される場合の適合率が 100%であるという意味について説明する．図 13 に示すように，名前 A を持つある概念が存在するとして，次のフェーズで部分概念 a’，b’ に分解され，部分概念の集合には A’ なる名前が与えられたとする．

このとき，適合率をカウントする方式に 2 つの立場が考えられる．包含されるものがすべて取り出されたので，再現率 100%，適合率 100%とする立場と，実際の変更作業において，確認すべきものはその一部分（たとえば a’）であり，a’ が含まれているため再現率は 100%，余分なもの（b’）を含むため適合率はたとえば 50%とする立場である．表 8 と表 9 における値は前者の立場をとっている．その根拠は，ほとんどの場合，部分概念の集合は 1 枚の図面に表現されるので，その中の関連要素を特定する作業は人手に任せるという立場である．後者の立場に立ったときの評価値を**表 10**，**表 11** の括弧内に示す．

どちらの立場を採用しても，再現率はほとんど変化しないが，後者の場合，適合率が当然，急減する．ところで，エレベータ制御システムのサンプル 10 で再現率が落ちているのは，変更に関連する要素間に基本依存関係が生成されなかったためである．

8.4.4　サンプルの有効性について

表 8，表 9 に示した分類 (1) のサンプルは，特定のものがとりあげられている．そのサンプルをどのようにとりあげたかを説明する．同じ型のサンプルについては，すべてのサンプルについて再現率と適合率を計算し，その平均値，分散，最大値，最小値を計算した．その中で平均値に近いものを選択した．**表 12**，**表 13** に，選択の基礎になった基本統計量を示す．なお，ユースケースとサブシステムについては，表 8，表 9 の分類 (2)，(3) の基本統計量を計算したものである．

表 10　エレベータ制御システムにおける変更波及解析結果
Table 10　Impact analysis result of Elevator Control System.

分類	サンプル番号	UML図式要素の名前 (UMLの型/役割)	正解	自動生成	共通	検索漏れ	ノイズ	再現率	適合率
(2)	6	Select Destination (ユースケース)	99 (8)	131	99 (8)	0 (0)	32 (123)	100% (100%)	75.6% (6.1%)
	7	Request Elevator (ユースケース)	100 (8)	127	98 (8)	2 (0)	29 (119)	98.0% (100%)	77.2% (6.3%)
	8	Dispatch Elevator (ユースケース)	138 (8)	177	138 (8)	0 (0)	39 (169)	100% (100%)	78.0% (4.5%)
	9	Stop Elevator at Floor (ユースケース)	154 (8)	185	150 (8)	4 (0)	35 (178)	97.4% (100%)	81.1% (4.3%)
(3)	10	FloorSubsystem (クラス/サブシステム)	23 (3)	23	23 (1)	0 (2)	0 (22)	100% (33.3%)	100% (4.3%)
	11	Scheduler (クラス/サブシステム)	26 (3)	26	26 (3)	0 (0)	0 (23)	100% (100%)	100% (8.3%)
	12	ElevatorSubsystem (クラス/サブシステム)	46 (4)	76	46 (4)	0 (0)	30 (72)	100% (100%)	60.5% (5.3%)

8.5　変更波及解析における有用性

ある概念に関連する部分構造の変更波及における有用性を，変更箇所特定への有用性，基

表 11　ATM システムにおける変更波及解析結果
Table 11　Impact analysis result of ATM system.

分類	サンプル番号	UML図式要素の名前 (UMLの型/役割)	正解	自動生成	共通	検索漏れ	ノイズ	再現率	適合率
(2)	6	Withdraw Funds (ユースケース)	145 (7)	145	145 (7)	0 (0)	0 (138)	100% (100%)	100% (4.8%)
	7	Client Withdraw Funds (ユースケース)	126 (7)	126	126 (7)	0 (0)	0 (119)	100% (100%)	100% (5.6%)
	8	Server Withdraw Funds (ユースケース)	95 (3)	95	95 (3)	0 (0)	0 (92)	100% (100%)	100% (3.2%)
	9	Validate PIN (ユースケース)	120 (7)	120	120 (7)	0 (0)	0 (113)	100% (100%)	100% (5.8%)
	10	Client Validate PIN (ユースケース)	108 (7)	108	108 (7)	0 (0)	0 (101)	100% (100%)	100% (6.5%)
	11	Server Validate PIN (ユースケース)	96 (3)	96	96 (3)	0 (0)	0 (93)	100% (100%)	100% (3.1%)
(3)	12	ATMClientSubsystem (パッケージ/サブシステム)	98 (6)	90	90 (6)	8 (0)	0 (84)	91.8% (100%)	100% (6.7%)
	13	BankServerSubsystem (パッケージ/サブシステム)	26 (2)	0	0 (0)	26 (2)	0 (0)	0% (0%)	0% (0%)

表 12　エレベータ制御システムにおけるサンプルの基本統計量
Table 12　Basic statistics of samples of Elevator Control System.

エレベータ制御システム	概念数	再現率（単位：%）				適合率（単位：%）			
		最小	最大	平均	標準偏差	最小	最大	平均	標準偏差
システム外部オブジェクト	11	100	100	100	0	1.0	100	56.4	29.6
インタフェースオブジェクト	8	66.7	100	79.7	16.8	50	71.4	62.8	8.4
コントロールオブジェクト	5	100	100	100	0	62.5	100	76.9	16.2
エンティティオブジェクト	2	100	100	100	0	56.3	68.4	62.3	8.6
タスクオブジェクト	9	28.6	100	62.4	35.7	56.3	100	88.1	17.7
ユースケース	4	97.4	100	98.9	1.3	75.6	81.1	77.9	2.3
サブシステム	3	100	100	100	0	60.5	100	86.8	22.8

表 13　ATM システムにおけるサンプルの基本統計量
Table 13　Basic statistics of samples of ATM system.

ATM システム	概念数	再現率（単位：%）				適合率（単位：%）			
		最小	最大	平均	標準偏差	最小	最大	平均	標準偏差
システム外部オブジェクト	11	60	100	96.4	12.1	1.2	100	42.3	39.7
インタフェースオブジェクト	5	100	100	100	0	100	100	100	0
コントロールオブジェクト	5	84.6	100	87.5	6.1	78.6	100	82.6	8.5
エンティティオブジェクト	13	100	100	100	0	42.9	100	87.1	8.6
タスクオブジェクト	4	76.9	85.7	82.5	3.9	50.0	94.4	73.6	19.4
ユースケース	6	100	100	100	0	100	100	100	0
サブシステム	2	0	91.8	45.9	64.9	0	100	100	70.7

本依存関係の十分性，正しい変更ルート設定への寄与の観点から評価する．

8.5.1　変更箇所特定への有用性

表 8，表 9 における，分類 (1) の評価結果より，対象とする UML 図面群より，ある概念と同じ概念を持つすべての UML 図式要素を良い精度で特定できることが分かった．このことは変更波及解析においては，それらの UML 図式要素を含む UML 図面群を特定することに寄与する．

また，分類 (2) の評価結果より，開発方法論の流れにそって作成された UML 図面群のうち，変更に関与するサブセットも良い精度で特定できるといえよう．

ただし，いずれの場合にも，概念の分解・統合が存在する場合には，再現率が落ち，必要な調査対象を見落とす恐れがある．

変更波及解析者に，我々のモデルが提供する情報は，何を調査しなければならないかに関する情報であり，どのような順序で何を変更すればよいかに関しては，人手にゆだねられる．

8.5.2　基本依存関係の十分性

文献 14) に，本論文で提案した方法と方向性が同じ研究成果が独立に報告されている．共通する点は，フィーチャモデルを除き，すべての UML モデルを対象にしていることである．すなわち，文献 14) に提案されているトレースモデルとルールに基づくトレーサビリティの実現は，トレースモデルが我々の付加規則に対応し，ルールが照合規則と付加規則にマージしたものに対応している．トレースモデルで取り扱っているモデル要素は，我々の生成モデル要素に，同じ粒度で対応している．文献 14) におけるモデル要素間のトレーサビリティ関係は，我々の基本依存関係に対応している．**表 14** に対応を示す．表 14 において，e1，e2 はモデル要素を表す．

表 14　トレーサビリティ関係と基本依存関係の比較
Table 14　Correspondences between traceability in Ref. 14) and basic dependency relationships.

トレーサビリティ関係	意味	対応する基本依存関係
Satisfiability	e1 が e2 の期待や要求を満たす	同一概念
Encompass	e1 が e2 の内容を含んでいる	生存従属，情報共有
Dependency	e1 が e2 の存在に依存している	生存従属，同一概念，情報共有
Overlap	e1 と e2 がシステムまたはドメインの共通側面を参照する	同一概念，情報共有
Evolution	e1 が e2 によって置き換えられる	該当なし
Implement	e1 が e2 を実現する	同一概念
Refinement	e1 が e2 を詳細化する	生存従属，同一概念，情報共有
Cotainment	e1 が e2 の要素を利用する	生存従属，情報共有
Similar	プロダクトライン固有	該当なし
Different	プロダクトライン固有	該当なし

我々の基本依存関係は解析可能な言語的手がかりをもとに分類されているのに対して，文献 14) では，より意味を精密にしたトレーサビリティ関係を定義しており，変更活動に関する示唆を与えうるという点において，文献 14) の方式のほうが有用であろう．しかし，文献 14) のルールは我々の照合規則と付加規則を一体化した形で定義されており，組織が採用するネーミングルールと開発方法論は独立したものであるので，我々の方式のように別個に定義するほうが，対応性や保守性にすぐれているものと思われる．

また，文献 16) より，UML モデル要素間の対応について我々の付加規則によく似た分析結果が独立に報告されている．違いは記述レベルの粒度である．文献 16) では，モデル間の整合性検証を実現するため，モデル間の対応をメソッド名やイベント名などの細粒度で対応づけている．我々のモデルの目的は，UML 図式要素の特定であるため，文献 16) より粒度は粗い．

8.5.3　正しい変更ルートの設定

変更波及解析の担当者に，現在の我々のモデルが提供できる情報は，どの図面や UML モデル要素を調査しなければならないかに関する情報であり，どのような順序で何を変更すればよいかに関しては，現在のところ提供できないことはすでに述べた．

表 8 における分類 (2) の 4 つのユースケースの解析において，**図 15** に示す解析結果が得られた．

異なる概念だが名前の類似によって生成された基本依存関係を，自動的に除去する方法を

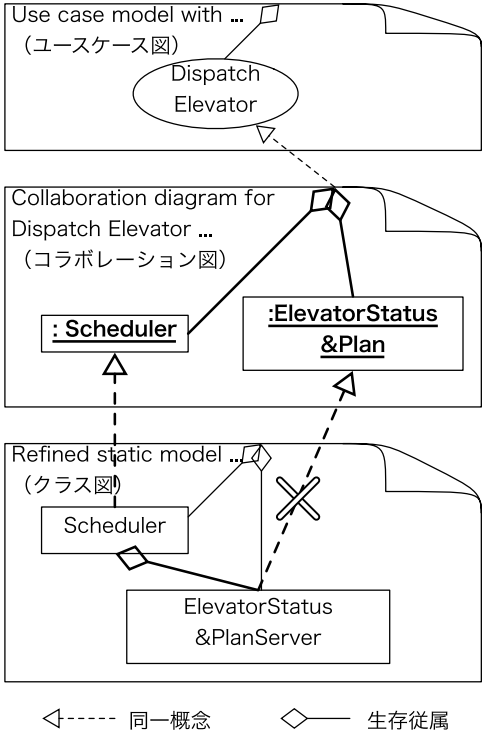

図 15　エレベータ制御システムにおける変更波及解析結果
Fig. 15　An example of impact analysis in Elevator Control System.

考察する．

図 15 において，3 種類の図はそれぞれ異なるフェーズで作成されたユースケース図，コラボレーション図，クラス図であり，各図は基本依存関係に結合された UML 図式要素を持っている．ここで，コラボレーション図中の ElevatorStatus&Plan とクラス図中の ElevatorStatus&PlanServer の間には，照合規則により誤った対応が認識され，結果として誤った基本依存関係が設定されている．この誤って認識された基本依存関係は，現在は人手によって削除することになる．

誤って認識された基本依存関係を削除するルールを定義することができれば，変更順序を示唆するワークフローの自動生成に発展させる可能性を持ち，今後の課題である．

8.6　本論文で提案した方式の有効性と課題

8 章での評価結果をまとめる．

限定された評価実験での範囲内ではあるが，以下のような本方式の利点が認識された．

- 自動生成された基本依存関係は，基礎データ 2 を定義したときに分析した実際の参照関係を十分に反映するものであり，8.3 節に示した精度を考慮にいれたうえで，有用であると判断できる．
- 本論文で提案した方式は，同じ概念には同じまたは類似の名前が与えられる（その逆も成立）という前提のもとで，同一概念を持つ UML 図式要素の抽出，ユースケースから始まり各フェーズで変換・詳細化されていく，主要中間成果物間の作成順序に関する情報の抽出に有用であると判断できる．

一方，本方式の持つ現在の限界も明らかになった．

- 分析設計の過程で認識定義された概念がそれ以降の図面で分解される場合，名前だけによる追跡は限界を持つ．ある概念が分解された場合，全体名の間に対応がある場合にのみ追跡可能である．
- また，概念の統合の取扱いは今後の課題として残された．

9. 議　　論

現在 UML2.0 が公開され，使われ始めている．本章では，本論文で提案した基本依存関係と依存関係生成モデルが UML2.0 に対してどの程度有効であるのかを検討する．

はじめに基本依存関係の有効性を検討する．本論文では，依存関係の分析対象を UML1.5 版のメタモデル要素 “Dependency” のサブクラスとした．UML2.0 版の分析対象を同様としたとき，分析対象はメタモデル要素 “Dependency” のサブクラスである “Abstraction” と “Usage” である[⋆1]．UML2.0 版では，両依存関係のステレオタイプ[⋆2]は変更されたが，そのセマンティクスの定義は変更されていない．また，開発作業の考察から定義された基本依存関係は，作成される図の版に非依存である．ここで，2 章において定義した基本依存関係とそれらを導出した依存関係の型の対応を**表 15** にまとめた．最左列には，メタモデル要素 “Dependency” の 4 種類のサブクラスと，開発作業から 2 種類の依存関係が発生する状況を示す．中央列には，本論文の対象とした UML1.5 版において，最左列の依存関係・状況

⋆1 UML2.0 版における “Binding” と “Permission” の定義は，単なる単方向矢印と改められた．
⋆2 UML2.0 版ではキーワードと呼ばれる．

表 15　版による基本依存関係の違い
Table 15　Differences of basic dependency relationship by versions.

		UML 1.5	UML 2.0
UMLの依存関係	Abstraction	同一概念	同一概念
	Binding	情報共有	—
	Permission	情報共有	—
	Usage	生存従属、情報共有	生存従属、情報共有
開発作業の依存関係	コピーされたもの	コピー	コピー
	包含関係の記述	生存従属	生存従属
基本依存関係の種類数		4種類	4種類

から導出された基本依存関係を示す．最右列には，UML2.0 版において，UML1.5 版の分析結果から流用できる基本依存関係を示す．最下行に，各版の基本依存関係の種類の数を示す．その数は両版とも 4 となり，種類の数は UML2.0 においても同じである．以上より，本論文で定義した基本依存関係が UML2.0 においても利用可能であると考えられる．

次に，UML2.0 の図面に対応可能な依存関係生成モデルへの拡張可能性を述べる．依存関係生成モデルを構成する 3 規則のうち，照合規則と選択規則は，UML 記述の名前と包含関係を利用する．そのため，UML2.0 の UML 記述に対応するルールを追加することは容易である．また，付加規則には，追加された図に対応する生成モデル要素の定義が必要になると考える．UML2.0 では，フレームの導入や，既存の図式要素の複合など，記述の参照や統合を可能にする仕様の拡張が行われた．これらの拡張は，既存の図式要素で用いられる包含や名前による参照など，照合規則で用いた特性を利用したものである．そのため，必要な生成モデル要素と基本依存関係を付加規則に追加することで UML2.0 に対応可能であると考えている．

以上より，本研究で提案した基本依存関係および，依存関係生成モデルによる自動生成のアプローチは UML2.0 にも適用可能であると判断する．なお，UML2.0 に対する拡張は今後の課題とする．

10. 関 連 研 究

変更作業支援のため，変更の対象となる中間成果物の量と順序を予測するさまざまな手法が提案されている．たとえば，(1) 形式言語 OCL によって記述された波及ルールを用いる手法がある[10]．波及ルールとは，変更された要素と波及する要素間の関係をルールで定義したものであり，OCL で記述される．まず変更の起点となる波及ルールを適用することで，波及解析が可能となる．また，(2) 中間成果物間のトレース依存関係を適切に設定するための指針を整備した研究もある[2),6),13)]．その多くは，トレース依存関係の分類を行っており，実際のトレース依存関係の設定は，開発者に任される．(1)，(2) の手法では，波及ルールやトレース依存関係は開発者自身が記述する必要があり，また，それらのメンテナンスコストも大きい．それらのコストを軽減するため，(3) 情報検索技術を利用する手法も提案されている．この手法では，UML 図やソースコードのキーワードを用いて波及解析を行うため，さまざまな種類の中間成果物に適用可能であるが，実際に波及しない中間成果物を多く含むという欠点を持つ[12]．また，この手法は，文章が不完全なものやタイトルがない文書群にも適用可能という優位な点を持つ．文献 9) においては，実際的なデータをもとに評価実験を行い，情報検索技術を適用した場合，再現率や適合率がどの程度の値になるかを示す興味深い報告が行われている．

(2) の手法において，依存関係の設定と波及解析の自動化に関する研究も行われている．たとえば，von Knethen らが開発した波及解析ツール QuaTrace[3] がある．変更波及解析に利用する中間成果物間の関係を，表記方法が異なる中間成果物間の関係を意味する ‘表現関係’，ある中間成果物とそれを具象化した中間成果物間の関係を意味する ‘改良関係’，ある中間成果物とそれを作成するのに必要となる中間成果物間の関係を意味する ‘依存関係’ に分類している．これらの分類のもとに，基本的には文書の名前を利用してトレース情報を生成する．波及解析ビューワは，表現関係，改良関係，依存関係を利用して，それらのトレース情報を分類する．

そのほか，Unified Process で定義されたモデル間の依存関係を形式的に表現する研究[5)]，依存関係に変更に役に立つ情報を付加した参照モデルの提案[4)] や，分散したコンポーネント間に発生する依存関係の定義[1)] などの研究がある．

また，本論文は，基本依存関係を自動生成するために UML モデルの分類を行った．このような特定の視点に特化した UML モデルの分類はいくつか行われている．8.5.2 項ですでに詳細な比較を行ったが，たとえば，Hayes らはプロダクトライン開発に特化した UML モデルの分類を提案し[9)]，大西は UML モデルの検証に特化した UML モデルの分類を提案した[16)]．

11. ま と め

本論文では，UML1.5 版のモデル要素として定義された依存関係を検討し，変更波及解

析に有用な基本依存関係（情報共有，同一概念，生存従属，コピーの 4 つ）を新たに定義した．また，基本依存関係を自動生成する手法を提案した．

COMET 法によるエレベータ制御システムと ATM システムの事例研究を題材として，要求定義から設計段階までの UML 図面群において，提案した依存関係の自動生成法の有用性と現段階での適用の限界を示した．

波及解析法について，変更の型（更新・削除）ごとに，変更順序を指示できるワークフローの自動生成法および，ソースコードへの波及解析をするための，図面とソースコード間の対応関係の生成法を現在考察中である．

謝辞　本研究は文部科学省科学研究費特定領域研究（2）課題番号 16016239 の補助をもとに実施された．また，本論文で提案した方式の評価法について，査読者の皆様から有益なコメントをいただいた．記して謝意を表する．

参 考 文 献

1) Keller, A., Blumenthal, U. and Kar, G.: Classification and Computation of Dependencies for Distributed Management, *Proc. 5th IEEE Symposium on Computers and Communications* (*ISCC 2000*), pp.78–83 (2000).
2) Fasolino, A.R. and Visaggio, G.: Improving Software Comprehension through an Automated Dependency Tracer, *7th International Workshop on Program Comprehension*, pp.58–65 (1999).
3) von Knethen, A. and Grund, M.: QuaTrace: A Tool Environment for (Semi-) Automatic Impact Analysis Based on Traces, *19th IEEE International Conf. on Software Maintenance* (*ICSM '03*), pp.246–255 (2003).
4) Ramesh, B. and Jarke, M.: Toward Reference Models for Requirements Traceability, *IEEE Trans. Softw. Eng.*, Vol.27, No.1, pp.58–93 (2001).
5) Pons, C., Giandini, R. and Baum, G.: Dependency Relations Between Models in the Unified Process, *Proc. 10th Intl. Workshop on Software Specifiaction and Design* (*IWSS '00*), pp.149–157 (2000).
6) Visaggio, G.: Structural Information as a Quality Metic in Software Systems Organization, *Proc. 1997 International Conference on Software Maintenance* (*ICSM '97*), pp.92–99 (1997).
7) Gomma, H.: *Designing concurrent, distributed, and real-time applications with UML*, Addison Wesley, Inc. (2000).
8) Jacobson, I., Booch, G. and Rumbaugh, J.: *The Unified Software Development Process*, Addison Wesley Longman, Inc. (1999).
9) Hayes, J.H., Dekhtyar, A. and Osborne. J.: Improving Requirements Tracing via Information Retrieval, *Proc. 11th IEEE Intl. Requirements Engineering Conference*, pp.138–147 (2003).
10) Briand, L.C., Labiche, Y. and O'Sullivan, L.: Impact Analysis and Change Management of UML Models, *19th IEEE International Conf. on Software Maintenance* (*ICSM '03*), pp.256–265 (2003).
11) Object Management Group: Unified Modeling Language (UML), version 1.5. http://www.omg.org/cgi-bin/doc?formal/03-03-01
12) Settimi, R., Cleland-Huang, J., Khadra, O.B., Mody, J., Lukasik, W. and DePalma, C.: Supporting Software Evolution through Dynamically Retrieving Traces to UML Artifacts, *Proc. 7th Intl. Workshop on Principles of Software Evolution* (*IWPSE '04*), pp.49–54 (2004).
13) Ibarahim, S., Idris, N.B., Munro, M. and Deraman, A.: A Software Traceability Validation For Change Impact Analysis of Object Oriented Software, *The 2006 World Congress in Computer Science, Computer Engeneering, and Applied Computing* (*SERP '06*), pp.453–459 (2006).
14) Jirapanthong, W. and Zisman, A.: Supporting Product Line Development through Traceability, *Proc. 12th Asia-Pacific Software Engineering Conference* (*APSEC05*), pp.506–514 (2005).
15) Yau, S.S., Collofello, J.S. and MacGregor, T.: Ripple Effect Analysis of Software Maintenance, *Proc. IEEE COMPSAC*, pp.60–65 (1978).
16) 大西　淳：UML における整合性検証支援システム，電子情報通信学会論文誌，Vol.J84-D-I, No.6, pp.671–681 (2001).

付　　録

A.1　照 合 規 則

表 16 に照合規則を示す．表 16 は，左からターゲットの型，ソースの型，照合条件の順に並んでいる．たとえば，ターゲットの型がクラスでソースの型もクラスの場合，ターゲットのクラスの名前がソースのクラスの名前に似ている場合と，ターゲットのクラスがソースのクラスに包含されている場合，その組合せが取り出されることを示す．

A.2　ElevatorControl システムにおける UML 図での概念の記述

表 17，**表 18**，**表 19** に，評価に用いた ElevatorControl システムの設計を示す UML 図から，各フェーズにおける概念の名前を示す．たとえば，概念 "ArrivalSensor" は，フェーズ「要求定義」において，アクタ "ArrivalSensor" なる型として，図面 1，2 に記述されたことを示す．

表 16　照合規則
Table 16　Comparison rule.

ターゲット	ソース	照合条件
ユースケース図	アクタ	Include
ユースケース図	ユースケース	Include
ユースケース	クラス図	Contained
ユースケース	ステートチャート図	Contained
ユースケース	アクティビティ図	Contained
ユースケース	コラボレーション図	Contained
ユースケース	シーケンス図	Contained
ユースケース	ユースケース	Similar
アクタ	アクタ	Similar
アクタ	クラス	Similar
アクタ	オブジェクト	TypeSim
クラス図	クラス	Include
クラス図	パッケージ	Include
パッケージ	クラス	Include
パッケージ	パッケージ	Similar
クラス	パッケージ	Similar
クラス	オブジェクト	SimType, Contained
クラス	ステートチャート図	Contained
クラス	アクティビティ図	Contained
クラス	クラス	Similar, Include
オブジェクト図	オブジェクト	Include
オブジェクト	クラス	TypeSim
オブジェクト	ステートチャート図	TypeSim
オブジェクト	アクティビティ図	TypeSim
オブジェクト	オブジェクト	Similar, Include
コンポーネント図	コンポーネント	Include
コンポーネント	コンポーネント	Similar
配置図	ノード	Include
ノード	ノード	Similar
ステートチャート図	状態	Include
状態	状態	Similar, Include
アクティビティ図	アクション状態	Include
アクション状態	アクション状態	Similar, Include
コラボレーション図	オブジェクト	Include
シーケンス図	オブジェクト	Include

表 17　エレベータ制御システムにおける UML 図での概念の記述 (1/3)
Table 17　Concept notations in UML diagrams of Elevator Control System (1/3).

概念	側面	フェーズ「要求定義」での名前[描写された図面No]	フェーズ「分析」での名前	フェーズ「サブシステム設計」での名前	フェーズ「タスク設計」での名前
ElevatorUser	型	ElevatorUser(A) [1,2]			
	実体		: ElevatorUser(O) [5,6]		
ArrivalSensor	型	ArrivalSensor(A) [1,2]	ArrivalSensor(C) [3,4]		
	実体		:ArrivalSensor(O) [7]	:ArrivalSensor(O) [15, 16]	:ArrivalSensor(O) [22, 23, 24, 29]
ArrivalSensorInterface	型			ArrivalSensorInterface(C) [18]	
	実体		:ArrivalSensorInterface(O) [7, 14]	:ArrivalSensorInterface(O) [16]	:ArrivalSensorsInterface(O) [23, 24]
Select Destination	機能	Select Destination(UC) [1,2]			
Request Elevator	機能	Request Elevator(UC) [1,2]			
Dispatch Elevator	機能	Dispatch Elevator(UC) [2]			
Stop Elevator at Floor	機能	Stop Elevator at Floor(UC) [2]			
Floor	型		Floor(C) [3]		
FloorButton	型		FloorButton(C) [3,4]		
	実体			:FloorButton(O) [15, 17]	:FloorButton(O) [22, 25, 26, 29]
FloorButtonInterface	型			FloorButtonInterface(C) [18]	
	実体		:FloorButtonInterface(O) [6,14]	:FloorButtonInterface(O) [17]	:FloorBottonsInterface(O) [25, 26]
FloorLamp	型		FloorLamp(C) [3,4]		
	実体		:FloorLamp(O) [9]	:FloorLamp(O) [15, 17]	:FloorLamp(O) [22, 25, 26, 29]
FloorLampInterface	型			FloorLampInterface(C) [18]	
	実体		:FloorLampInterface(O) [9, 14]	:FloorLampInterface(O) [17]	FloorLampsMonitor(O) [25, 26]
DirectionLamp	型		DirectionLamp(C) [3,4]		
	実体		:DirectionLamp(O) [7,9]	:DirectionLamp(O) [15, 17]	:DirectionLamp(O) [22, 25, 26, 29]
DirectionLampInterface	型			DirectionLampInterface(C) [18]	
	実体		:DirectionLampInterface(O) [7,9, 14]	:DirectionLampInterface(O) [17]	:DirectionLampsMonitor(O) [25, 26]
Elevator	型		Elevator(C) [3,4]		
ElevatorButton	型		ElevatorButton(C) [3,4]		
	実体			:ElevatorButton(O) [15, 16]	:ElevatorButton(O) [22, 23, 24, 29]
ElevatorButtonInterface	型			ElevatorButtonInterface(C) [18]	
	実体		:ElevatorButtonInterface(O) [5,14]	:ElevatorButtonInterface(O) [16]	:ElevatorButtonsInterface(O) [23, 24]

表 18　エレベータ制御システムにおける UML 図での概念の記述（2/3）
Table 18　Concept notations in UML diagrams of Elevator Control System (2/3).

概念	側面	フェーズ「要求定義」での名前[描写された図面No]	フェーズ「分析」での名前	フェーズ「サブシステム設計」での名前	フェーズ「タスク設計」での名前
ElevatorLamp	型		ElevatorLamp(C) [3,4]		
	実体		:ElevatorLamp(O) [7]	:ElevatorLamp(O) [15, 16]	:ElevatorLamp(O) [22, 23, 24, 29]
ElevatorLampInterface	型			ElevatorLampInterface(C) [18]	
	実体		:ElevatorLampInterface(O) [7,14]	:ElevatorLampInterface(O) [16]	
Motor	型		Motor(C) [3,4]		
	実体		:Motor(O) [7,9]	:Motor(C) [15, 16]	:Motor(O) [22, 23, 24, 29]
MotorInterface	型			MotorInterface(C) [18]	
	実体		:MotorInterface(O) [7,9,14]	:MotorInterface(O) [16]	
Door	型		Door(C) [3,4]		
	実体		:Door(O) [7,9]	:Door(O) [15, 16]	:Door(O) [22, 23, 24, 29]
DoorInterface	型			DoorInterface(C) [18]	
	実体		:DoorInterface(O) [7,9,14]	:DoorInterface(O) [16]	
DoorTimer	型			DoorTimer(C) [18]	
	実体		:DoorTimer(O) [7,14]	:DoorTimer(O) [16]	
ElevatorManager	型			ElevatorManager(C) [18]	
	実体		:ElevatorManager(O) [5,6, 14]	:ElevatorManager(O) [16]	:ElevatorManager(O) [23, 24]
ElevatorControl	型			ElevatorControl(C) [18]	
	実体		:ElevatorControl(O) [5,6,7,9,14]	:ElevatorControl(O) [16]	:ElevatorController(O) [23, 24]
ElevatorStatus&Plan	型			ElevatorStatus&Plan(C) [18]	
	実体		:ElevatorStatus&Plan(O) [5,6,7,9,14]	:ElevatorStatus&Plan(O) [16]	
LocalElevatorStatus&Plan	型				LocalElevatorStatus&Plan(C) [21b]
	実体				:LocalElevatorStatus&Plan(O) [23, 24]
ElevatorStatus&PlanServer	型			ElevatorStatus&PlanServer(C) [18]	
	実体				:ElevatorStatus&PlanServer(O) [27, 28]
OverallElevatorStatus&Plan	型			OverallElevatorStatus&Plan(C) [18]	OverallElevatorStatus&Plan(C) [21b]
	実体				:OverallElevatorStatus&Plan(O) [27, 28]
ElevatorScheduler	型			ElevatorScheduler(C) [18]	
	実体				:ElevatorScheduler(O) [27,28]
SchedulerSubsystem	型			Scheduler(C) [18]	
	実体		:Scheduler(O) [5,6,7,9, 14]	:Scheduler(O) [15, 16, 17]	:Scheduler(O) [22, 23, 24, 25, 26, 27, 28, 29]
ElevatorControlSystem	型		ElevatorControlSystem(C) [4]		
	実体			:ElevatorControlSystem(O) [15]	:ElevatorControlSystem(O) [22, 29]

表 19　エレベータ制御システムにおける UML 図での概念の記述（3/3）
Table 19　Concept notations in UML diagrams of Elevator Control System (3/3).

概念	側面	フェーズ「要求定義」での名前[描写された図面No]	フェーズ「分析」での名前	フェーズ「サブシステム設計」での名前	フェーズ「タスク設計」での名前
ElevatorSubsystem	型			ElevatorSubsystem(C) [18]	
	実体			:ElevatorSubsystem(O) [15, 16, 17]	:ElevatorSubsystem(O) [22, 23, 24, 25, 26, 27, 28, 29]
FloorSubsystem	型			FloorSubsystem(C) [18]	
	実体			:FloorSubsystem(O) [15, 16, 17]	:FloorSubsystem(O) [22, 23, 24, 25, 26, 27, 28, 29]
Elevator Starting Up	状態		Elevator Starting Up(S) [10, 11, 13]		
Elevator Starting Down	状態		Elevator Starting Down(S) [11, 13]		
Elevator Moving	状態		Elevator Moving(S) [8,10, 11, 13]		
Elevator Stopping	状態		Elevator Stopping(S) [8, 11, 13]		
Elevator Door Opening	状態		Elevator Door Opening(S) [8, 11, 13]		
Elevator at Floor	状態		Elevator at Floor(S) [8, 11, 13]		
Checking Next Destination	状態		Checking Next Destination(S) [8, 10, 11, 12, 13]		
Elevator Idle	状態		Elevator Idle(S) [8, 10, 11, 12, 13]		
Door Closing to Move Up	状態		Door Closing to Move Up(S) [10, 11, 13]		
Door Closing to Move Down	状態		Door Closing to Move Down(S) [11, 13]		
Preparing to Move Up	状態		Preparing to Move Up(S) [12, 13]		
Preparing to Move Down	状態		Preparing to Move Down(S) [12, 13]		
Moving to Floor	状態		Moving to Floor(S) [12, 13]		

＊(UC)…ユースケース，(A)…アクタ，(C)…クラス，(O)…オブジェクト，(S)…状態

A.3　ATM システムにおける UML 図での概念の記述

表 20，**表 21**，**表 22**，**表 23**，**表 24** に，評価に用いた ATM システムの設計を示す UML 図から，各フェーズにおける概念の名前を示す．

表 20　ATM システムにおける UML 図での概念の記述（1/5）
Table 20　Concept notations in UML diagrams of ATM system (1/5).

概念	側面	フェーズ「要求」 での名前[描写された図面No]	フェーズ「静的モデリング」 での名前	フェーズ「オブジェクト配置」 での名前	フェーズ「動的モデリング」 での名前	フェーズ「サブシステム」 での名前	フェーズ「タスク設計」 での名前
Withdraw Funds Function	機能	Withdraw Funds(UC) [1]					
Client Withdraw Funds Function	機能			Client Withdraw Funds(UC) [9]			
Server Withdraw Funds Function	機能			Server Withdraw Funds(UC) [9]			
Query Account Fuction	機能	Query Account(UC) [1]					
Client Query Account Function	機能			Client Query Account(UC) [9]			
Server Query Account	機能			Server Query Account(UC) [9]			
Transfer Funds Function	機能	Transfer Funds(UC) [1]					
Client Transfer Funds Fuction	機能			Client Transfer Funds(UC) [9]			
Server Transfer Funds Function	機能			Server Transfer Funds(UC) [9]			
Validate PIN Function	機能	Validate PIN(UC) [1]					
Client Validate PIN Function	機能			Client Validate PIN(UC) [9]			
Server Validate PIN Function	機能			Server Validate PIN(UC) [9]			
Add Cash Function	機能	Add Cash(UC) [1]		Add Cash(UC) [9]			
Startup Function	機能	Startup(UC) [1]		Startup(UC) [9]			
Shutdown Fuction	機能	Shutdown(UC) [1]		Shutdown(UC) [9]			
Bank	型		Bank(C) [2, 4, 5]				
BankingSystem	型		BankingSystem(C) [2]	BankingSystem(P) [8, 10]			
	実体					:BankingSystem(O) [27, 28]	
Customer	型		Customer(C) [4, 5]	Customer(C) [8]			
CustomerInterface	型			CustomerInterface(C) [10, 11]	CustomerInterface(C) [25]		
	実体				:CustomerInterface(O) [12, 13, 16, 17, 24]		:CustomerInterface(O) [29, 30]

表 21　ATM システムにおける UML 図での概念の記述（2/5）
Table 21　Concept notations in UML diagrams of ATM system (2/5).

概念	側面	フェーズ「要求」での名前[描写された図面No]	フェーズ「静的モデリング」での名前	フェーズ「オブジェクト配置」での名前	フェーズ「動的モデリング」での名前	フェーズ「サブシステム」での名前	フェーズ「タスク設計」での名前
ATMCustomer	型	ATM Customer(A) [1]	ATMCustomer(C) [2,3]	ATMCustomer(C) [8, 10]			
	実体				:ATMCustomer(O) [12, 13, 16, 17, 24]	:ATMCustomer(O) [27, 28]	:ATMCustomer(O) [29, 30]
Operator	型	Operator(A) [1]	Operator(C) [2,3]	Operator(C) [10]			
	実体				:Operator(O) [24]	:Operator(O) [27, 28]	:Operator(O) [29, 30]
OperatorInterface	型			OperatorInterface(C) [10, 11]	OperatorInterface(C) [25]		
	実体				:OperatorInterface(O) [24]		:OperatorInterface(O) [29, 30]
ATM	型		ATM(C) [2]				
ATMInfo	型		ATMInfo(C) [4, 7]				
ATMControl	型			ATMControl(C) [11]	ATMControl(C) [25]		
	実体				:ATMControl(O) [12, 13, 16, 17, 24]		:ATMController(O) [29, 30]
CardReaer	型		CardReader(C) [2,3]	CardReader(C) [8, 10]			
	実体				:CardReader(O) [12, 16, 24]	:CardReader(O) [27, 28]	:CardReader(O) [29, 30]
CardReaderInterface	型			CardReaderInterface(C) [10, 11]	CardReaderInterface(C) [25]		
	実体				:CardReaderInterface(O) [12, 13, 16, 17, 24]		:CardReaderInterface(O) [29, 30]
ATMCard	型		ATMCard(C) [2, 7]	ATMCard(C) [11]	ATMCard(C) [25]		
	実体				:ATMCard(O) [12, 13, 24]		:ATMCard(O) [29, 30]
CashDispenser	型		CashDispenser(C) [2,3]	CashDispenser(C) [8, 10]			
	実体				:CashDispenser(O) [16, 24]	:CashDispenser(O) [27, 28]	:CashDispenser(O) [29, 30]
CashDispenserInterface	型			CashDispenserInterface(C) [10, 11]	CashDispenserInterface(C) [25]		
	実体				:CashDispenserInterface(O) [16, 17, 24]		
ATMCash	型		ATMCash(C) [2, 7]	ATMCash(C) [11]	ATMCash(C) [25]		
	実体				:ATMCash(O) [16, 17, 24]		:ATMCash(O) [29, 30]

表 22　ATM システムにおける UML 図での概念の記述（3/5）
Table 22　Concept notations in UML diagrams of ATM system (3/5).

概念	側面	フェーズ「要求」での名前[描写された図面No]	フェーズ「静的モデリング」での名前	フェーズ「オブジェクト配置」での名前	フェーズ「動的モデリング」での名前	フェーズ「サブシステム」での名前	フェーズ「タスク設計」での名前
ReceiptPrinter	型		ReceiptPrinter(C) [2,3]	ReceiptPrinter(C) [8, 10]			
	実体				:ReceiptPrinter(O) [16, 24]	:ReceiptPrinter(O) [27, 28]	:ReceiptPrinter(O) [29, 30]
ReceiptPrinterInterface	型			ReceiptPrinterInterface(C) [10, 11]	ReceiptPrinterInterface(C) [25]		
	実体				:ReceiptPrinterInterface(O) [16, 17, 24]		
Receipt	型		Receipt(C) [2]				
Account	型		Account(C) [4, 5]				
	実体				:Account(O) [19]		
CardAccount	型		CardAccount(C) [4, 7]				
	実体				:CardAccount(O) [15, 26]		
CheckingAccount	型		CheckingAccount(C) [4, 5]				
	実体				:CheckingAccount(O) [26]		
SavingsAccount	型		SavingsAccount(C) [4, 5]				
	実体				:SavingsAccount(O) [26]		
DebitCard	型		DebitCard(C) [4, 5]				
	実体				:DebitCard(O) [15, 19, 26]		
BankTransactionServer	実体				:BankTransactionServer(O) [26]		
ATMTransaction	型		ATMTransaction(C) [4, 6]	ATMTransaction(C) [11]	ATMTransaction(C) [25]		
	実体				:ATMTransaction(O) [12, 13, 16, 17, 24]		:ATMTransaction(O) [29, 30]
WithdrawalTransaction Manager	実体				:WithdrawalTransaction Manager(O)[26]		
WithdrawalTransaction	型		WithdrawalTransaction(C) [4, 6]		WithdrawalTransaction(C) [25]		
QueryTransactionManager	実体				:QueryTransactionManager(O) [26]		
QueryTransaction	型		QueryTransaction(C) [4, 6]		QueryTransaction(C) [25]		

表 23　ATM システムにおける UML 図での概念の記述（4/5）
Table 23　Concept notations in UML diagrams of ATM system (4/5).

概念	側面	フェーズ「要求」での名前[描写された図面No]	フェーズ「静的モデリング」での名前	フェーズ「オブジェクト配置」での名前	フェーズ「動的モデリング」での名前	フェーズ「サブシステム」での名前	フェーズ「タスク設計」での名前
TransferTransaction Manager	実体				:TransferTransactionManager (O)[26]		
TransferTransaction	型		TransferTransaction(C) [4, 6]		TransferTransaction(C) [25]		
PINValidationTransaction Manager	実体				:PINValidationTransaction Manager(O)[15, 19, 26]		
PINValidationTransaction	型		PINValidationTransaction(C) [4, 6]		PINValidationTransaction(C) [25]		
TransactionLog	実体				:TransactionLog(O) [19, 26]		
ATMClientSubsystem	型			ATMClientSubsystem (C)[8] (P)[9, 11]			
	実体				:ATMClient(O) [15, 19, 24, 26]	:ATMClient(O) [27, 28]	:ATMClient(O) [29,30]
BankServerSubsystem	型			BankServerSubsystem (C)[8] (P)[9]			
	実体				:BankServer(O) [12, 13, 16, 17, 24, 26]	:BankServer(O) [27, 28]	:BankServer(O) [29, 30]
Idle	状態				Idle(S) [14, 18, 20, 21, 23]		
Waiting for PIN	状態				Waiting for PIN(S) [14, 21]		
Validating PIN	状態				Validating PIN(S) [14, 21]		
Waiting for Customer Choice	状態				Waiting for Customer Choice(S)[14, 18, 21]		
Terminating	状態				Terminating(S) [18, 23]		
Ejecting	状態				Ejecting(S) [18, 23]		
Printing	状態				Printing(S) [18, 23]		
Dispensing	状態				Printing(S) [18, 23]		
Processing Withdrawal	状態				Processing Withdrawal(S) [18, 22]		
Closed Down	状態				Closed Down(S) [20, 23]		
Processing Customer Input	状態				Processing Customer Input(S) [20, 21]		
Terminating Transaction	状態				Terminating Transaction(S) [20, 23]		
Processing Transaction	状態				Processing Transaction(S) [20, 22]		

表 24　ATM システムにおける UML 図での概念の記述（5/5）
Table 24　Concept notations in UML diagrams of ATM system (5/5).

概念	側面	フェーズ「要求」での名前[描写された図面No]	フェーズ「静的モデリング」での名前	フェーズ「オブジェクト配置」での名前	フェーズ「動的モデリング」での名前	フェーズ「サブシステム」での名前	フェーズ「タスク設計」での名前
Processing Transfer	状態				Processing Transfer(S) [22]		
Processing Query	状態				Processing Query(S) [22]		
Confiscating	状態				Confiscating(S) [23]		

＊(UC)…ユースケース，(A)…アクタ，(C)…クラス，(P)…パッケージ，(O)…オブジェクト，(S)…状態

(平成 19 年 10 月 16 日受付)
(平成 20 年 4 月 8 日採録)

小谷　正行

昭和 53 年生．平成 15 年北陸先端科学技術大学院大学情報科学研究科情報システム学専攻修士課程修了．平成 20 年同大学院博士課程修了．同年（株）インターネットイニシアティブ入社．ソフトウェア開発支援環境に興味を持つ．博士（情報科学）．日本ソフトウェア科学会会員．

落水浩一郎（正会員）

1946 年生．1969 年大阪大学基礎工学部卒業．1974 年同大学大学院基礎工学研究科博士課程修了．工学博士．静岡大学工学部講師，助教授，教授を経て，1992 年より北陸先端科学技術大学院大学情報科学研究科教授．ソフトウェア工学，特に，オブジェクト指向開発方法論とその支援環境，分散共同開発のプロセスモデルと支援環境，ソフトウェアアカウンタビリティ機能の実現に関する研究に従事．著書に『ソフトウェア工学実践の基礎（日科技連）』，『オブジェクトモデリング（アジソン・ウェスレイ・パブリッシャーズ・ジャパン）』等．IEEE，日本ソフトウェア科学会，教育システム情報学会各会員．