修 士 論 文

共同ソフトウェア開発における
ソフトウェア文書の変更管理法

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

小谷 正行

2003 年 3 月

修 士 論 文

共同ソフトウェア開発における
ソフトウェア文書の変更管理法

指導教官　落水 浩一郎 教授

審査委員主査　落水 浩一郎 教授
審査委員　篠田 陽一 教授
審査委員　権藤 克彦 助教授

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

110044 小谷 正行

提出年月: 2003 年 2 月

概 要

本論文では、UML 図の変更管理に関するメタモデルを利用して、UML 図面間の依存関係を自動的に生成する方式を提案する。具体的には、UML 図および UML 図要素を抽象化したメタ要素を定義し、同一、生存の主従というメタ要素の生存期間を示す関係を定義する。メタ要素間の関係を定義して、UML 図の変更管理モデルを定義する。UML 図とメタ要素の集合を対応させるため領域を定義し、領域を越えたメタ要素間の関係を UML 図面間に依存関係を付加する手法を提案する。簡単な具体例を用いて、この手法の有効性を示す。

目 次

第1章　はじめに

1.1　背景

共同ソフトウェア開発では、多数の中間成果物（ソース、設計書、ドキュメントなど）が生成される。ほとんどの中間成果物は、作成済みの中間成果物を参照して生成される。このため、中間成果物間において複雑に何らかの依存関係が存在する。ここで、依存関係とは「B が A に依存する」とき「A を変更した場合、B を変更する必要がある」という関係である。この依存関係を管理・利用することで、変更の影響を受ける中間成果物の特定が可能になる。しかし、開発者自身が依存関係を定義することは、そのメンテナンスに多大なコストを要する。

本研究は、設計図として用いられる UML[1] 図面群を対象とする。UML はモデリング言語として、さまざまな開発プロセスにおいて用いられており [2] [3]、 UML 図の作成支援ツールも開発されている [4] [5] 。これらの作成支援ツールは UML 図面間において、削除に関する依存関係を手動で付加できる。この機能を用いて共同開発を行う場合、開発者は自身が作成する UML 図と関連する UML 図を理解する必要がある。これを自動化することで、開発者が関連する UML 図を理解するコストなどを軽減することが期待できる。

1.2　目的

本研究の目的は、UML 図の変更管理に着目したメタモデルを用いて UML 図面間の依存関係を付加する方式（model-based translation[6] ）を提案する。ここで用いるメタモデルを変更管理モデルと呼ぶ。この方式は、UML 図およびその構成要素を UML 図の変更管理モデルに用いる要素（メタ要素）に対応させ、変更管理モデルを利用してメタ要素間に関係を付加する。そして、その関係を利用して、UML 図面間の依存関係を生成する方式である。

UML 図面間に依存関係を生成するシステムは、入力を UML 図面群、出力を UML 図面間の依存関係とする。このとき、システム内部では UML 図を解析し、それに基づいて、UML 図面間の依存関係を生成する処理を行う。

UML 図の解析は、UML 図面間の依存関係が判断できる粒度で解析する。UML 図はユースケース図、クラス図、オブジェクト図、コンポーネント図、配置図、状態遷移図、アクティビティ図、シーケンス図、コラボレーション図の９種類ある。９種類の図面間の関係をすべて羅列すると、多数の関係をあげることができる。その中の、いくつかをあげる。

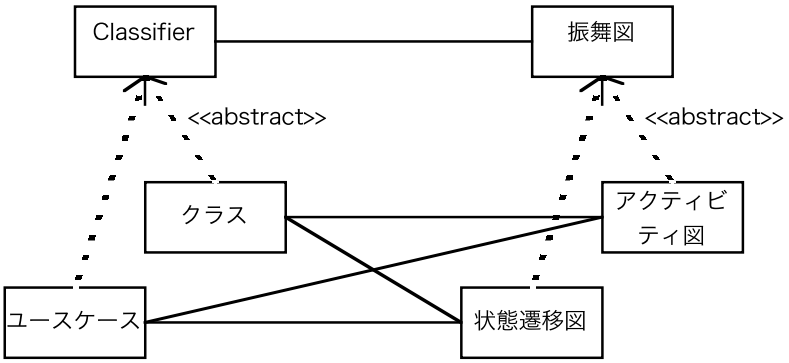


図 1.1: 関係の抽象

- ユースケースとその振舞を示す状態遷移図
- ユースケースとその振舞を示すアクティビティ図
- クラスとその振舞を示す状態遷移図
- クラスとその振舞を示すアクティビティ図

上記の４つの関係は、それぞれユースケース図と状態遷移図、ユースケース図とアクティビティ図、クラス図と状態遷移図、クラス図とアクティビティ図の関係である。このようにすべての図の間の関係を羅列すると条件が煩雑になり、図面間の依存関係が理解しづらい。そこで、各 UML 図、UML 図要素の抽象化を行う。上記の例では、振舞を表現する図として状態遷移図とアクティビティ図があげられている。そこで状態遷移図とアクティビティ図を抽象した図として振舞図を定義する。また、ユースケースとクラスは、それぞれ振舞を示す図として状態遷移図とアクティビティ図がある。そこでユースケースとクラスを抽象化した UML 図要素として Classifeir を定義する。すると、上記の４つの例は以下の１つの抽象した関係で表現することができる。

- Classifier と振舞図 (図 1.1)

また、依存関係の生成は抽象した要素間に用いる関係の種類の一部を用いる。例えば、上記の例の場合、振舞図が単独で存在可能か考慮する。振舞図は、振舞を行うものがないと図として成り立たない。ゆえに “Classifier が存在しなければ、振舞図は存在しない” という関係を定義できる。つまり、Classifier を削除すると、これと関係を持つ振舞図も削除するという関係である。逆に、Classifier は振舞図と関係を持つ必要があるか考慮する。この場合、Classifier は例えば振舞を行わないエンティティかもしれないため、Classifier は必ず振舞図を持つとは限らない。ゆえに “Classifier は振舞図の存在の有無に関係ない” という関係を定義できる。これら２つの定義した関係を “生存の主従” という関係で定義する。つまり上記の例では以下のように定義できる。

- 振舞図は Classifier を主として生存の主従関係がある

このように、UML 図面間の依存関係を生成できる粒度で、要素の抽象化、および特殊な関係の付加を行う。ここでは、図の構成要素の粒度で、変更管理モデルを定義する。

UML 図面間の依存関係を生成するシステムの処理を、変更管理モデルを用いて述べる。まず、UML 図面群をシステムの入力とする。つぎに UML 図面の構成要素が変更管理モデルのどの構成要素に対応するか判断する。そして、変更管理モデルで定義した関係を利用して、UML 図要素間の依存関係を生成し、これを UML 図面間の依存関係に読み替えて出力する。これを図 1.2 に示す。

この手法の有効性の検証を具体例を用いて、人間が定義した UML 図面間の依存関係と、この手法で生成した依存関係を比較し吟味する。

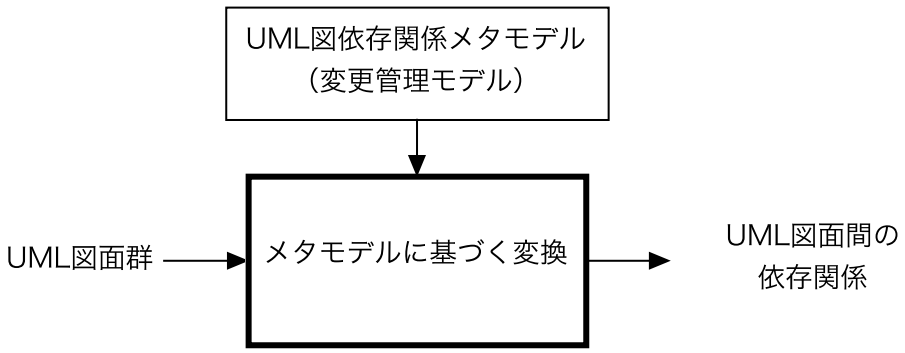


図 1.2: 依存関係の自動生成の概要

1.3 構成

本論文の構成は以下の通りである。２章で各 UML 図が表現することを中心に述べる。３章で UML 図面間に存在する依存関係を洗いだし、それに基づいてメタ要素とメタ要素間の関係を定義する。そしてメタ要素間の関係から UML 図面間の依存関係を自動生成する方法を述べ、UML 図の変更管理モデルを定義する。４章で簡単な利用例を用いて本方式の有効性を示し、５章で結論を述べる。

第2章　UML図

本章では、研究の対象であるUML図が用いられる用途を述べ、各UML図が何を表現しているか述べる。

2.1　概要

Unified Modeling Language(UML)は、開発中のソフトウェアシステムの中間成果物を視覚化、詳細化、構造化、文書化するための言語である。UMLを使うことで、開発者は自分たちが作る製品の青写真、つまり図面で視覚化が可能になる。UMLは、分析および設計の成果物であるセマンティックモデル、構文的な表記、図を標準化している。

UMLでは、システムをモデリングする観点をビューと呼び、その視覚表現をダイアグラムと呼ぶ。図は、あるシステムを５つのビューでとらえ、モデル化したものを概念的に表している。中央にかかれているユースケースビューは、利用者やテスタの視点から見て、このシステムがどのように振る舞うべきかを表現している。左上に書かれている論理ビューは、システムの内部構造に着目して、このシステムが提供すべき機能を静的および動的に表現している。右上の実装ビューは、ソースコードやライブラリなどといった、システムの物理コンポーネントの依存関係を示すものである。左下のプロセスビューは、並行性や同期といったメカニズムに着目して、システムをモデル化したものである。そして、右下にある配置ビューは、ハードウェアの物理コンポーネントの配置という切り口から、システムをモデル化したビューである。

UMLでのモデリングは、各種ビューの視覚的表現であるダイアグラムを書く作業である。UMLで定義している図は、ユースケース図、クラス図、オブジェクト図、シーケンス図、コラボレーション図、状態遷移図、アクティビティ図、配置図、コンポーネント図である。これらの図の詳細を次節で述べる。

2.2　UML図

本節は、各UML図の表現する内容について詳細に述べる。

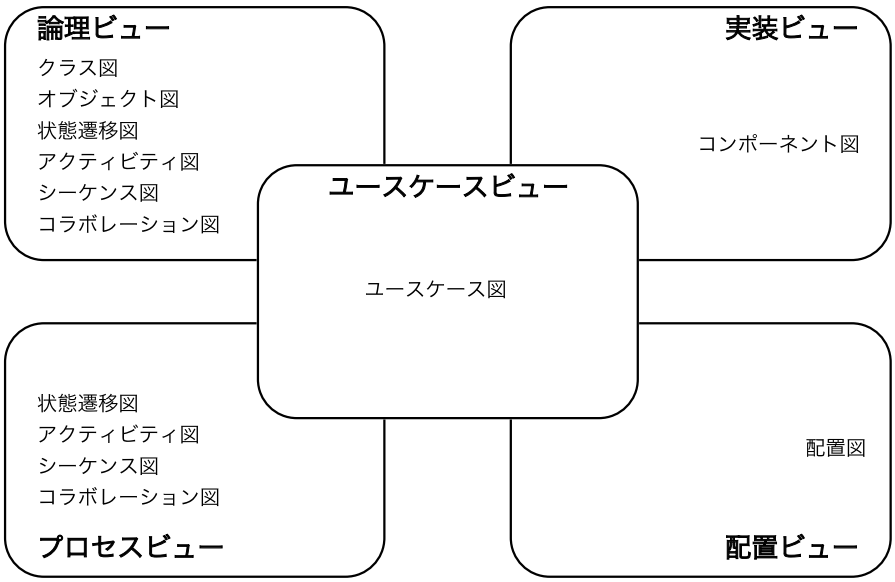


図 2.1: システムのビュー

ユースケース図

ユースケース図は、ユースケースとアクターの相互関係を示す図であり、システムの振舞を組織化またはモデリングする図である。アクターは、モデリングするシステムの外部に存在する抽象を表現する。例えば、顧客、管理者、データベースなどである。ユースケースは、いくつかのアクションシーケンスを記述したものである。これは常にアクターによって起動され、それによりアクターが望むべき結果を得ることができる。アクターは人型、ユースケースは楕円で表記される。また、アクター間、ユースケース間に用いられる関係は、依存と汎化である。依存は２つの要素間の意味的関係であり、依存する要素は、もう一方の要素の変更により影響を受ける関係を示す。汎化は一般的な要素とより特化した要素の関係であり、より特化した要素は一般的な要素の情報や振舞をすべて含み、さらに追加の情報や振舞を含む関係を示す。依存の関係は破線の矢印で示し、変更の影響を与える要素側に矢印を向けて表記する。汎化の関係は白抜きの三角形が片方の端についた直線で示し、一般的な要素側に白抜きの三角形を向けて表記する。さらに、アクターとユースケース間にはコミュニケーション関連がある。これは関連の一種であり、ユースケースを起動するアクター、ユースケースの活動に必要なアクター、および、ユースケースの活動の結果、望むべき結果を得ることができるアクターとユースケースはコミュニケーション関連を持つ。コミュニケーション関連は、直線で表記される。

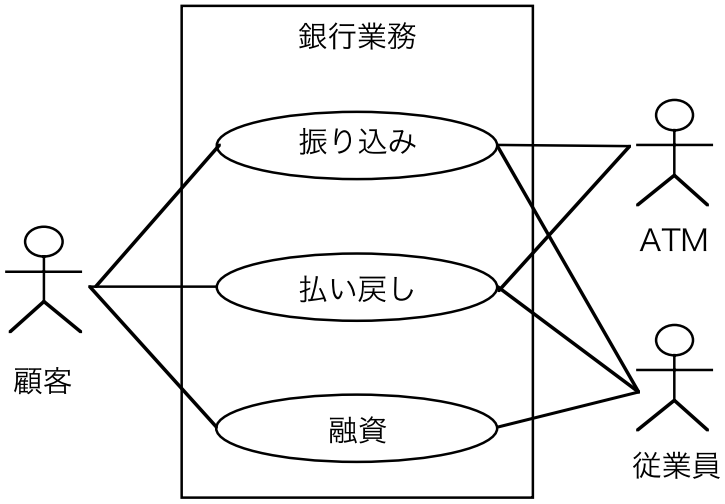


図 2.2: ユースケース図

クラス図

クラス図は、クラスおよびインターフェイスの相互関係を示す図であり、システムの静的な側面をモデリングする図である。クラスは、オブジェクトの型を記述するものであり、型の属性と振舞を記述する。クラス間の関係は、クラス間の結びつきを示す関連、クラスの拡張を示す汎化、変更の影響を示す依存、同じものを示すが抽象化のレベルが違う改良、全体を示すクラスとその部分を示すクラスの関係を示す集約、部分は全体と共に消滅することを示すコンポジション集約などがある。これらの定義された関係に意味を追加するステレオタイプがある。クラスは四角形、クラス間の関係は直線や破線の矢印などで表記され、ステレオタイプは ≪ stereotype≫ の形式で関係の近くに付加される。また、オブジェクト化した場合のオブジェクト数の関係を示す多重度を、関係の両端に付加して示す。

オブジェクト図

オブジェクト図は、オブジェクトとそれらの相互関係を示す図であり、クラス図に登場するクラスなどをインスタンス化したオブジェクト間のリンクを示す図である。オブジェクト名と型となるクラス名は “(オブジェクト名) : (クラス名)” で示す。オブジェクトは四角形、オブジェクト間のリンクは直線で表記する。

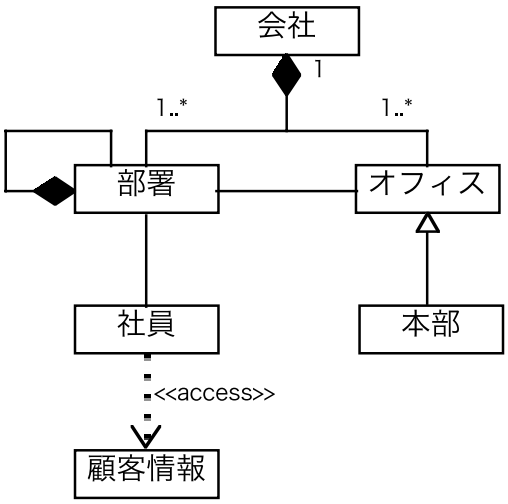


図 2.3: クラス図

コンポーネント図

コンポーネント図は、コンポーネントの構成とそれらの間の依存関係を示す図であり、システムのソフトウェアの静的な側面をモデリングする図である。コンポーネントは、システムの物理的なものを示す。例えば、実行可能なプログラムやライブラリなどである。コンポーネントはタブのついた四角形で表記する。

配置図

配置図は、実行時処理ノードとそこに存在するコンポーネントの配置を示す図であり、システムのハードウェアの静的な側面をモデリングする図である。ノードは、実行時に存在するハードウェアを示す。例えば、プロセッサを持つコンピュータ、プリンタ、カードリーダなどである。ノードは、立方体で表記される。

状態遷移図

状態遷移図は、状態、遷移、イベント、アクティビティからなる状態遷移を示す図であり、外部から受けるイベントにより振舞が変化するオブジェクトの動的な側面をモデリングする図である。状態は、オブジェクトの属性値と他の状態へのリンクで示す。遷移は、２つの状態間の関係で、特定のイベントが発生し、特定の条件が満たされたとき、特定のアクションが実行され、状態が変化することを示す。アクティビティは状態遷移内部で進行する実行を示す。状態は角丸の四角形、遷移は直線で表記される。

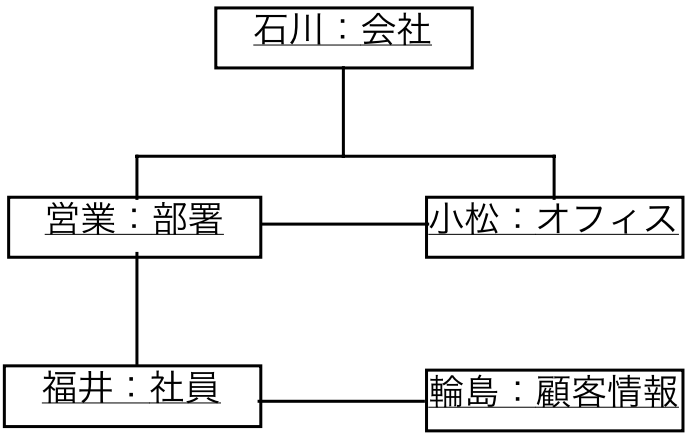


図 2.4: オブジェクト図

アクティビティ図

アクティビティ図は、アクティビティからアクティビティへのフローを示す図であり、実行プロセスの逐次的な（並行的な）ステップを示す動的なモデリングをする図である。アクティビティはアクション状態と呼ばれ、属性値を設定、何らかの値を返す式を評価、オブジェクトに対して操作を呼び出しなどを行う。 また、並行的なプロセスを示すフォークや、プロセスの分岐を示すガード式がある。

シーケンス図

シーケンス図は、オブジェクトとその間でやり取りされるメッセージ通信からなる相互作用を示す図であり、メッセージの時間軸に沿った順序を強調した動的な側面をモデリングする図である。相互作用に参加するオブジェクトを横軸に沿って配置し、これらのオブジェクトが送受信するメッセージを上から下への時間経過の順序で縦軸に添って配置する。また、オブジェクトの生存期間を示すオブジェクト生存線と、オブジェクトがアクションを実行している期間を示す制御フォーカスがある。

コラボレーション図

コラボレーション図も、シーケンス図と同じくオブジェクトとその間でやり取りされるメッセージ通信からなる相互作用を示す図であり、相互作用に参加するオブジェクトの構成を強調した動的な側面をモデリングする図である。メッセージは、オブジェクト間に接続しているリンクの近くに流れる方向を矢印で示され、内容はメッセージの時間順を示すシーケンス番号を付加したシーケンスによって示される。

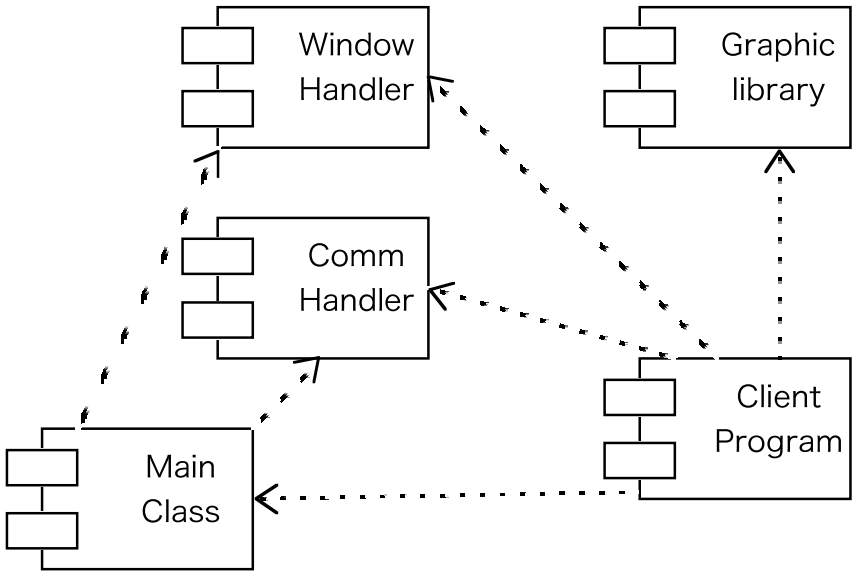


図 2.5: コンポーネント図

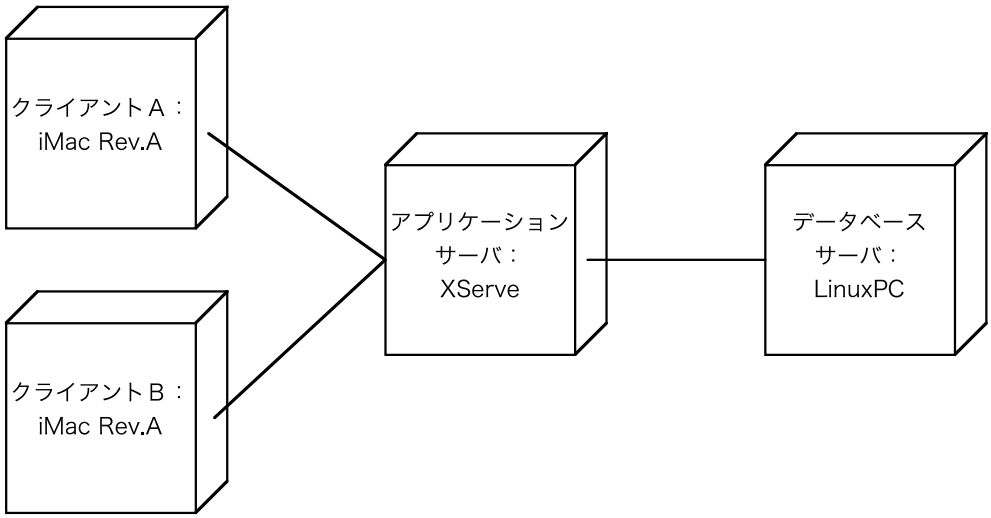


図 2.6: 配置図

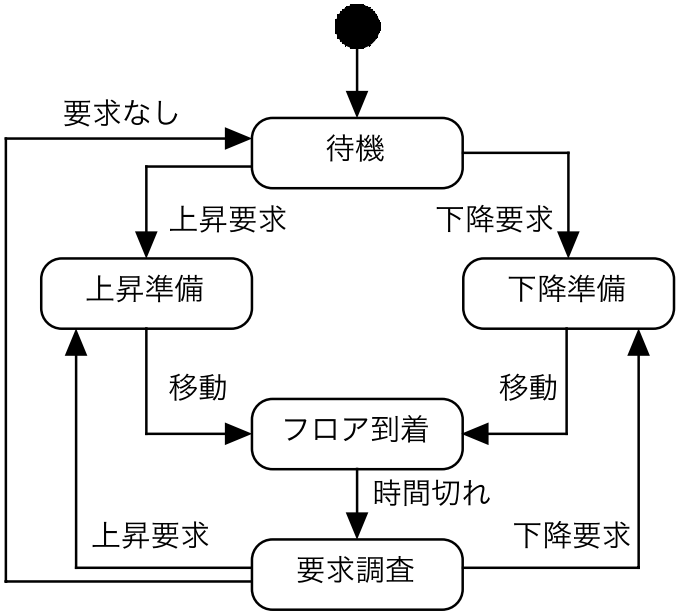


図 2.7: 状態遷移図

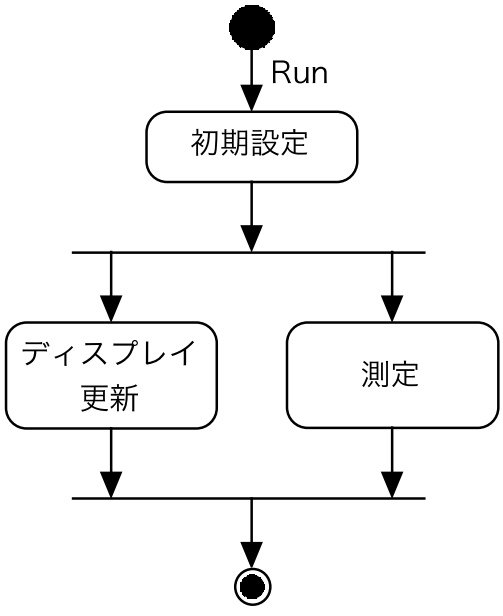


図 2.8: アクティビティ図

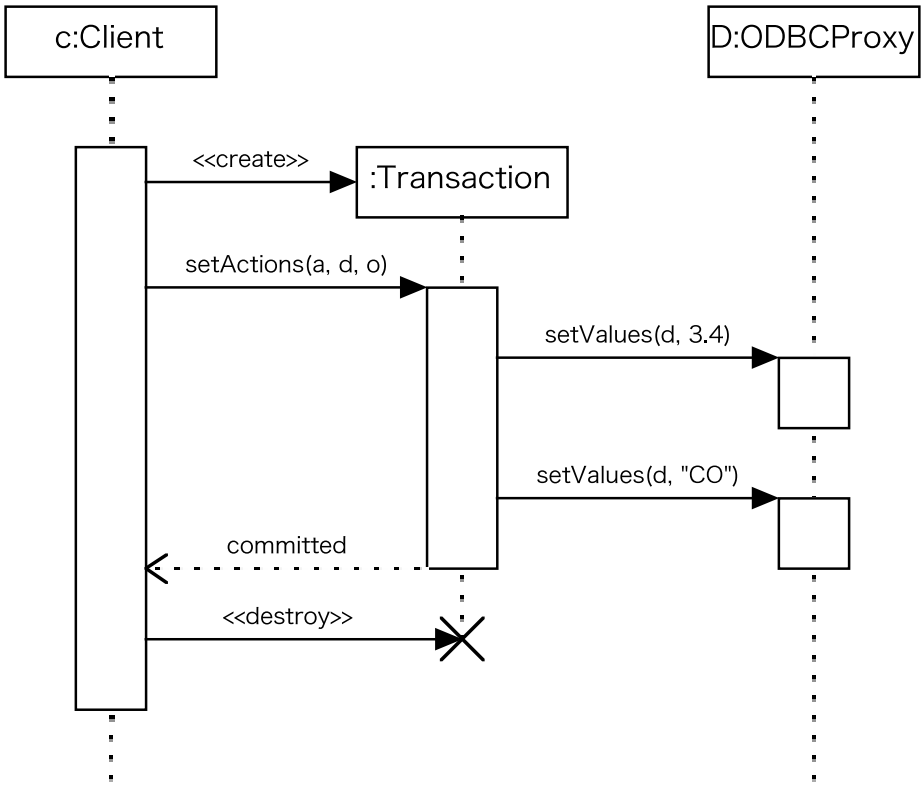


図 2.9: シーケンス図

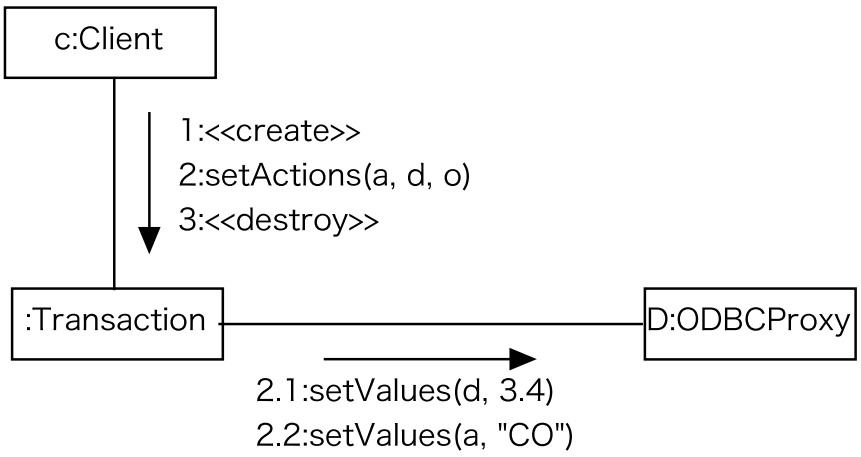


図 2.10: コラボレーション図

第3章　UML図の変更管理モデル

変更管理において必要な情報は「ある要素が変更されたとき、関係をもつ要素は変更するか」である。ここで要素はUML図、またはUML図の構成要素を示す。そこで、この方針に基づいて、UML図およびその構成要素の特徴と、UML図面間の関係を調査し、UML図面間の依存関係を定義可能なUML図の粒度の要素を用いて、変更管理モデルを定義する。ここで変更管理モデルに用いる要素をメタ要素と定義する。そしてメタ要素群と１枚のUML図との関係を示し、メタ要素間の関係からUML図面間の依存関係を生成する手法を述べる。

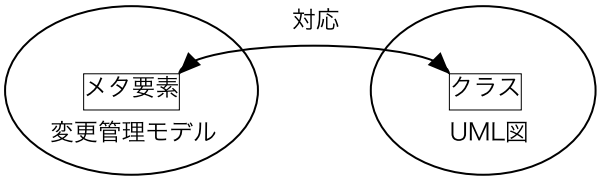


図 3.1: 変更管理モデルの位置づけ

3.1　変更管理モデルに用いる構成要素の粒度

メタ要素に用いるUML図の粒度を定義するため、クラス図間の関係、クラス図と状態遷移図の関係、クラス図とコラボレーション図の関係を例にあげ、UML図面間に存在する依存関係と、それを断定可能な粒度を述べる。

クラス図間の関係

まず、２つのクラス図の関係を考える。この２つのクラス図をそれぞれクラス図A、クラス図Bとする。２つの図が無関係であることを除いて、このとき考えられる関係は以下の３つある。

1. クラス図Aはクラス図Bと一部分を共有している。
2. クラス図Aはクラス図Bの一部分である。

3. クラス図Aはクラス図Bを具体化している。

これらの関係は、2つに分類可能である。1と2で示す関係は、双方のクラス図で全く同じクラスをいくつか用いている、という関係である。また、3で示す関係は、同じことを示すクラス図であるが具体的に示している、という関係である。これらの関係を断定するには、クラス図の内容を知る必要がある。この場合、UML 図面間の関係は、UML 図の構成要素間の関係で生成される。

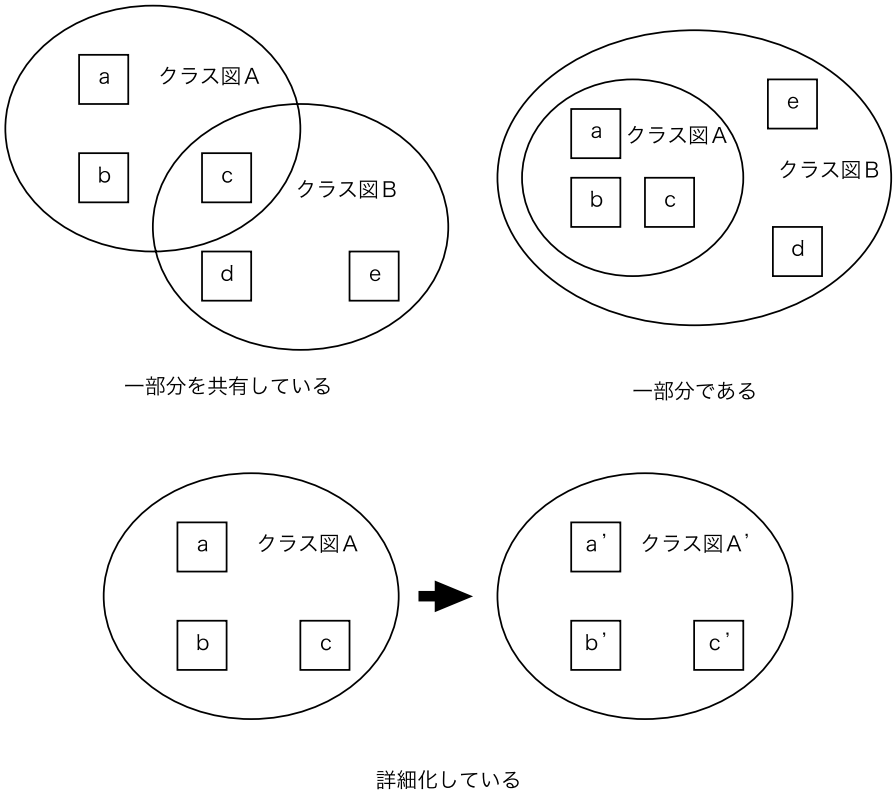


図 3.2: クラス図間の関係

クラス図と状態遷移図の関係

クラス図と状態遷移図の関係を考える。このとき考えられる依存関係は1つである。

1. 状態遷移図は、1つのクラスの状態遷移を表現した図である。

1枚のクラス図が1枚の状態遷移図と依存関係を持つ場合、図面の粒度で関係を定義しても構わないと考える。しかし、この関係は1つのクラス図と複数の状態遷移図面間に存

在することもある。この場合、各クラスがそれぞれ状態遷移図と関係を保持している状態が考えられる。ゆえに、この関係を断定するには、クラス図の内容を知る必要がある。この場合、UML 図面間の関係は、UML 図と図の構成要素間の関係で生成される。

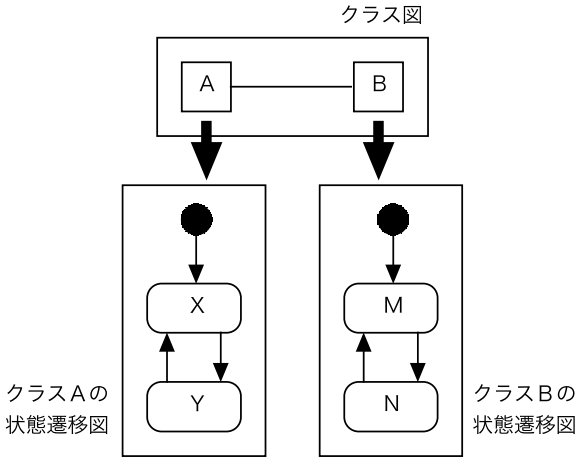


図 3.3: クラス図と状態遷移図の関係

クラス図とコラボレーション図の関係

クラス図とコラボレーション図の関係を考える。このとき考えられる関係は１つである。

1. コラボレーション図は、クラス図の関係をインスタンス化し、メッセージを付加した図である。

これは１対１の図の関係を定義している。しかし、この依存関係はクラス図とコラボレーション図の名前で判断することも可能である。しかし、名前のみで判断するのは危険である。例えば、クラス図Aとそれを具体化したクラス図A ’、クラス図Aの関係をインスタンス化し、メッセージを付加したコラボレーション図Bがある。このとき、クラス図Aとクラス図A ’が同一の名前を持っていた場合、クラス図A ’に用いたクラスをインスタンス化していないにもかかわらず、クラス図A ’とコラボレーション図Bが依存関係を持つことになる。ゆえに、クラス図とコラボレーション図の内容を知る必要がある。

これらより、UML 図面間の依存関係を定義するには、UML 図面の内容を知る必要がある。ゆえに、変更管理モデルを UML 図の構成要素の粒度で定義する。

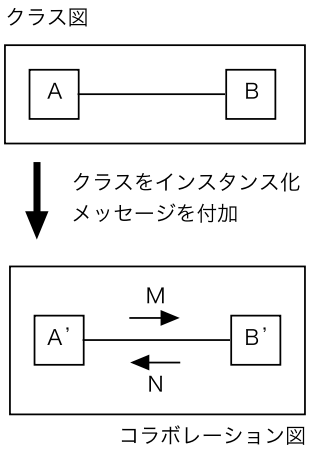


図 3.4: クラス図とコラボレーション図の関係

3.2 UML 図のメタ要素の定義

各 UML 図が表現することを考慮し、変更管理モデルに用いる UML 図のメタ要素を定義する。

まず、関係を示す UML 図を吟味する。これはクラス間の関係を表現するクラス図、オブジェクト間の関係を表現するオブジェクト図、アクターとユースケースの間の関係を示すユースケース図、コンポーネント間の関係を示すコンポーネント図、ハードウェア間の関係を示す配置図である。これらのメタ要素を**関係図**と定義する。各 UML 図で関係を示している UML 図要素を列挙するとクラス、オブジェクト、ユースケース、アクター、ノード、コンポーネントとなる。これらを **Classifier** とする。

つぎに、残りの UML 図を吟味する。残りの UML 図は動的な振舞を表す図であるが、どのような振舞を表現するかで分類する。状態遷移図とアクティビティ図は、例外はあるが、ひとつの Classifier の振舞を表現する図であり、シーケンス図とコラボレーション図は Classifier をインスタンス化したオブジェクト間のコミュニケーションを表現する図である。それぞれ**振舞図**、**状態遷移図**と定義する。

3.3 UML 図要素のメタ要素の定義

ここで関係図、振舞図、相互作用図の構成要素を定義し、UML 図要素との対応を定義する。

表 3.1: UML 図のメタ要素

関係図	Classifier 間の関係を表現する図
	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
振舞図	Classifier の状態を表現する図
	状態遷移図、アクティビティ図
相互作用図	複数の Classifier 間のコミュニケーションを表現する図
	シーケンス図、コラボレーション図

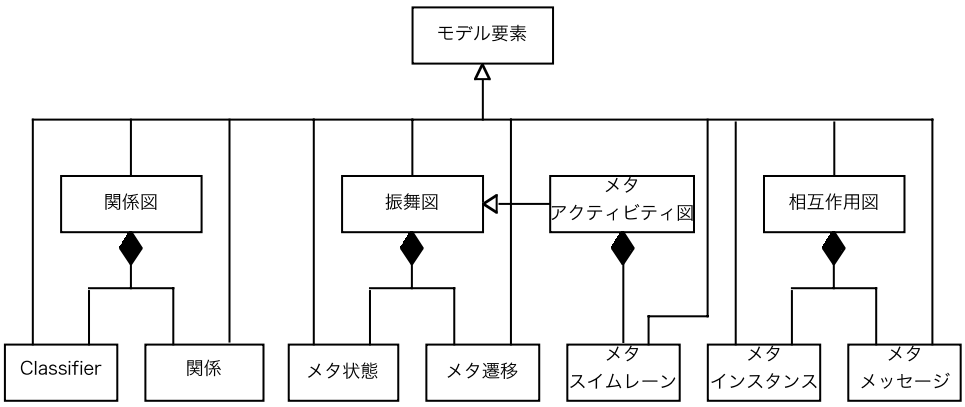


図 3.5: 変更管理モデルの構成要素

3.3.1 関係図の構成要素

関係図は Classifier 間の関係を表現する図である。ゆえに、関係図の構成要素を示す**Classifier** と、その間の関係を示す**関係**のメタ要素を定義する。

前述したように、Classifier をメタ要素とする UML 図要素は、ユースケース図に用いるアクター、ユースケース、クラス図に用いるクラス、オブジェクト図に用いるオブジェクト、コンポーネント図に用いるコンポーネント、配置図に用いるノードである。

関係をメタ要素とする UML 図要素は、ユースケース図に用いるコミュニケーション関連、関係図に分類した全ての図に用いる関連、依存、集約、汎化、オブジェクト図に用いるリンクである。

表 3.2: 関係図の構成要素

UML 図要素	メタ要素
Classifier	アクター、ユースケース、クラス、オブジェクト、コンポーネント、ノード
関係	コミュニケーション関連、関連、依存、集約、汎化、リンク

3.3.2 振舞図の構成要素

振舞図は Classifier の振舞を表現する図である。ゆえに、ある状態を示す**メタ状態**と、状態から状態への遷移を示す**メタ遷移**のメタ要素を定義する。また、振舞図を継承して**メタアクティビティ図**を定義し、その構成要素としてスイムレーンを示す**メタスイムレーン**を定義する。アクティビティ図には、スイムレーンを使用してアクションを行う対象を指定する。この対象は Classifier であるため、変更管理モデルのメタ要素として定義する。

メタ状態をメタ要素とする UML 図要素は、状態遷移図に用いる状態、アクティビティ図に用いるアクション状態である。

メタ遷移をメタ要素とする UML 図要素は、振舞図に分類したすべての図に用いるアクション、遷移、イベントである。

前述したように、メタスイムレーンをメタ要素とする UML 図要素は、アクティビティ図に用いるスイムレーンである。

表 3.3: 振舞図の構成要素

UML 図要素	メタ要素
メタ状態	状態、アクション状態
メタ遷移	アクション、遷移、イベント
メタスイムレーン	スイムレーン

3.3.3 相互作用図の構成要素

相互作用図は Classifier をインスタンス化したオブジェクト間のコミュニケーションを表現する図である。ゆえに、Classifier をインスタンス化したオブジェクトを示す**メタインスタンス**と、オブジェクト間に通信を行うメッセージを示す**メタメッセージ**を定義する。

メタインスタンスをメタ要素とする UML 図要素は、相互作用図をメタ要素とするすべての UML 図に用いるオブジェクトである。

メタメッセージをメタ要素とするUML図要素は、相互作用図をメタ要素とするすべてのUML図に用いるメッセージである。

表 3.4: 相互作用図の構成要素

UML図要素	メタ要素
メタインスタンス	オブジェクト
メタメッセージ	メッセージ

3.4 UML図と変更管理モデルの対応

変更管理モデルのメタ要素として、UML図とUML図の構成要素の２種類のメタ要素を定義した。この２種類は、例えばクラスとその振舞を示す状態遷移図は関係があるため、同等に扱い関係を定義する。そのため、変更管理モデルではUML図とメタ要素の対応が判断しづらい。そこで１枚のUML図を、UML図のメタ要素と構成要素のメタ要素をパッケージ化して表現する。このパッケージを**領域**と定義し、関係図、振舞図、相互作用図の領域を、それぞれ**関係図領域、振舞図領域、相互作用図領域**と定義する。

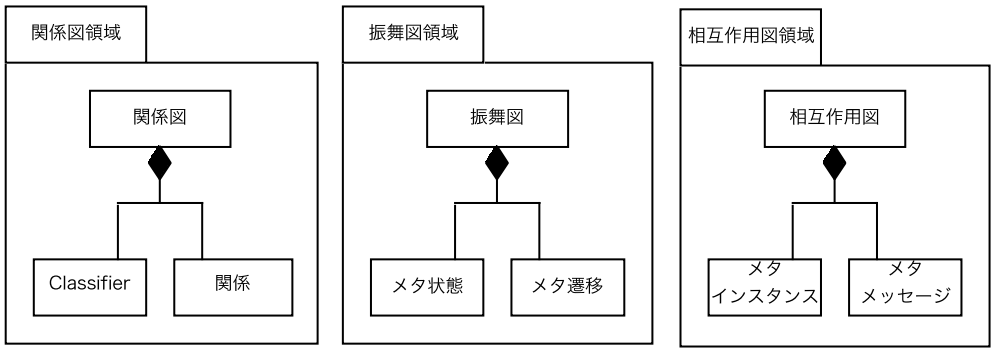


図 3.6: メタ要素の領域

3.5 関係の定義

本節では、メタ要素間に付加する関係を定義する。また、定義した関係が変更管理モデルが目的としている依存関係の自動生成に用いることが可能か吟味する。

本論では、**生存の主従関係、同一関係、関係**を定義する。また、同一関係に**コピー、イ**

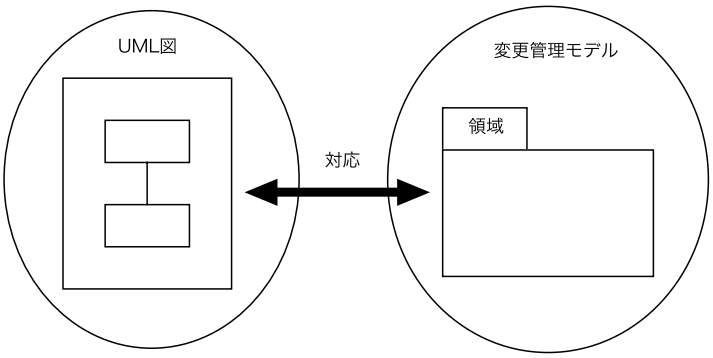


図 3.7: UML 図と変更管理モデルの対応

ンスタンス化、**詳細化**の３つのステレオタイプを定義する。以下、それぞれの関係について詳細に述べる。

3.5.1　生存の主従関係

生存の主従関係は、メタ要素間に生存に関する関係が成立することを示す関係である。モデル表記は、メタ要素間を棒でつないで関係があることを示し、生存の主導権を持つメタ要素の端を黒のひし形で示す。例えば、メタ要素Ａとメタ要素Ｂがあり、生存の主導権をメタ要素Ａがある。このとき、メタ要素Ａが削除されたとき、それに伴い、メタ要素Ｂも削除する必要がある。しかし、メタ要素Ｂを削除しても、メタ要素Ａは削除する必要がない関係を示す。ただし、主導権を持つメタ要素Ａが生成されたのと同時に、メタ要素Ｂは生成される必要はない。

3.5.2　同一関係

同一の関係は、この関係があるメタ要素は同一のものを示しているという関係である。ただし、粒度やクラスとオブジェクトという変形はステレオタイプとし、本論ではコピー、インスタンス化、詳細化の３つの種類を定義する。モデル表記は、メタ要素間を矢印の破線で関係を示し、基になるメタ要素の端に矢印を向ける。例えば、メタ要素Ａとメタ要素Ｂがあり、メタ要素Ａに矢印が向いていたとき、メタ要素Ａが基であることを示す。

コピー

コピーのステレオタイプを持つ同一の関係は、基を示すメタ要素がコピー基を示す。具体的には、メタ要素Ａとメタ要素Ｂの間にコピーのステレオタイプを持つ同一の関係がある場合、メタ要素Ｂはメタ要素Ａのコピーであることを示す。コピーの関係をもつ場合、

両方のメタ要素は同一の名前をもつため、双方向にコピーのステレオタイプを持つ同一の関係が存在することになる。

インスタンス化

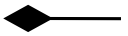
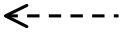
インスタンス化のステレオタイプを持つ同一の関係は、基を示すメタ要素がインスタンス基のクラスであることを示す。具体的には、メタ要素Aとメタ要素Bの間にインスタンス化のステレオタイプをもつ同一の関係がある場合、メタ要素Bはメタ要素Aをインスタンス化したものであることを示す。

詳細化

詳細化のステレオタイプを持つ同一の関係は、先のメタ要素は基のメタ要素を詳細化したものであることを示す。具体的には、メタ要素Aとメタ要素Bの間に詳細化の同一関係がある場合、メタ要素Bはメタ要素Aを詳細化したものあることを示す。

3.5.3 関係

“関係” の関係は、何らかの関係があることを示し、生存に関する主従関係はない。モデル表記は、メタ要素間を直線で接続する。例えば、メタ要素Aとメタ要素Bの間に “関係” の関係がある。このとき、メタ要素Aを削除してもメタ要素Bは削除されない。また、メタ要素Bを削除してもメタ要素Aは削除されない。

関係 A－B	意味
	生存の主従関係
	同一 （コピー、インスタンス）

3.6 自動的に付加可能な関係

変更管理モデルは、依存関係を自動生成する目的を持つ。つまり、システムを構築したとき、システムが判断可能な関係を用いる必要がある。

システムが判断できることとして、文字列の一致がある。これは、２種類の文字列が存在するとき、双方の文字列が同一か判断する手法である。これを明らかに使用できる関係は**コピーの同一**と**インスタンス化の同一**である。コピーの同一関係は、双方のメタ要素が

同一のものであるため、同じ名前を用いている。ゆえに文字列の一致で関係の付加を判断することが可能である。また、インスタンス化の同一関係は、メタ要素の名前の一部が一致する。インスタンス化されたUML図要素、すなわちオブジェクトの名前は「(クラス名) : (オブジェクト名)」で示される。文字列を一致させる対象位置が定義されているため、インスタンス化の同一関係は、文字列の一致で関係の付加を判断することが可能である。

一方、システムとして構築した場合、自動で関係を付加することが不可能と考える関係は詳細化の同一である。なぜなら、詳細化した内容をシステムは認識できないと考えるからである。

これ以外の関係は、両端のメタ要素が異なるものを示すため、名前の一致で判断することができない。しかし、例えば関連するメタ要素が一致するなど、何らかの条件を与えることによって、システムが判断することが可能であると考える。

ゆえに、変更管理モデルにおいてメタ要素間に用いる関係を**コピーの同一、インスタンス化の同一、生存の主従、関係**とする。

3.7 UML図面間の依存関係の生成法

メタ要素間には関係、生存の主従、同一の3種類の関係があり、削除の変更に関して述べると、生存の主従、同一の関係を追跡することで、削除する必要があるメタ要素を抽出することが可能である。メタ要素はUML図面の要素と対応するため、抽出されたメタ要素を、それらに対応するUML図要素に付加することによって、削除するUML図要素の順番を生成することが可能である。

しかし、人間が要求する情報はUML図を変更する順番である。つまり、各々のUML図要素がどのUML図の要素か判断する基準が必要である。そこで、1枚のUML図と対応させたメタ要素の集合を示す領域を用いる。メタ要素間の関係は、領域を越えた関係も存在する。その関係が生存に関する関係ならば、それが領域間の依存関係と解釈可能である。すなわち、これはUML図面間の依存関係である。

このように、メタ要素の関係からUML図面間の依存関係を生成する。

3.8 メタ要素間の関係の定義

UML図要素間、およびUML図要素とUML図の関係に基づき、メタ要素間の関係を定義する。

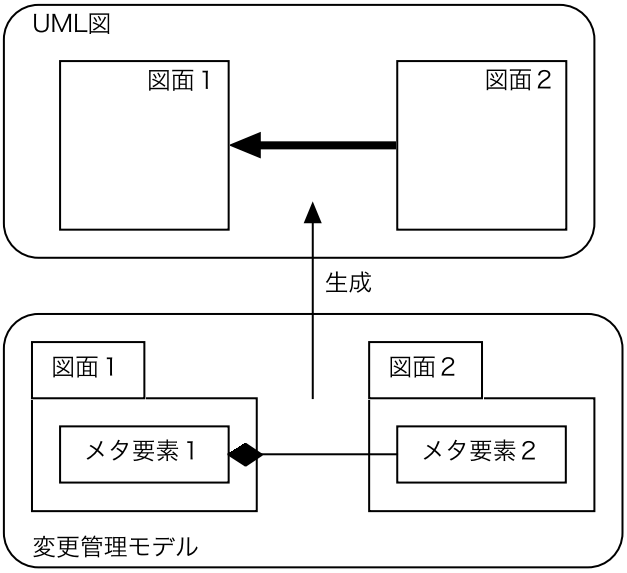


図 3.8: UML 図面間の依存関係の付加

モデル要素の再帰関係

モデル要素に、再帰関係のコピーの同一関係を定義する。具体的には、２つのクラス図で同一のクラスを用いる場合、同一物間の対応を表している。

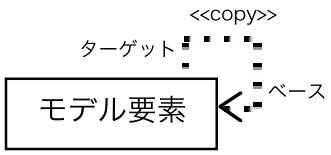


図 3.9: UML 図の変更管理モデル ー モデル要素

つぎに、領域を越えるメタ要素間の関係を定義する。

関係図領域と振舞図領域の関係

関係図領域と振舞図領域の依存関係を生成するメタ要素間の関係は、Classifier と振舞図である。

振舞図の定義より、振舞図が存在するには Classifier が存在しなければならない。また、逆に Classifier にとって、振舞図は振舞を示す１つの手段にすぎないため、Classifier は振舞図が存在しなくてよい。例えば、クラスとその振舞を示す状態遷移図の関係である。ゆえに、振舞図と Classifier の関係は、Classifier を主とする生存の主従関係と定義する。

関係図領域とメタアクティビティ図領域の関係

関係図領域とメタアクティビティ図領域の依存関係を生成するメタ要素間の関係は、Classifier と振舞図、および Classifier とメタスイムレーンの関係である。

Classifier とスイムレーンの関係について述べる。スイムレーンは Classifier に相当するため、Classifier が存在しないとスイムレーンは存在できない。しかし、すべての Classifier はスイムレーンで表現されない。例えば、あるユースケースのアクティビティ図に用いたスイムレーンと、そのユースケースとコミュニケーション関連があるアクターの関係である。ゆえに、スイムレーンと Classifier の関係は、Classifier を主とする生存の主従関係である。

関係図領域と相互作用図領域の関係

関係図領域と相互作用図領域の依存関係を生成するメタ要素間の関係は、Classifier とメタインスタンス、およびメタ関係とメタメッセージの関係である。

まず、Classifier とメタインスタンスの関係について述べる。メタインスタンスは Classifier をインスタンス化したものである。すなわち Classifier を削除した場合、それをインスタンス化したメタインスタンスも削除する必要がある。例えば、クラスとそのオブジェクトとの関係である。また、型と実体という違いはあるが、この 2 つは同一のものを示している。ゆえに、Classifier とメタインスタンスの関係は、Classifier を主とするインスタンス化の同一の関係である。

つぎに、‘関係’ とメタメッセージの関係について述べる。メタメッセージは ‘関係’ をインスタンス化したリンク上で通信する。すなわち、Classifier 間に ‘関係が存在しないと、メッセージ通信を行えない。また、メタメッセージはリンク上をいくつでも通信することができ、メタメッセージを削除しても ‘関係’ を削除する必要がない。例えば、クラス間の関係とそれをインスタンス化したオブジェクト間のメッセージの関係である。ゆえに、メタメッセージと ‘関係’ の関係は、関係を主とする生存の主従関係である。

振舞図領域と相互作用図領域の関係

振舞図領域と相互作用図領域のメタ要素間の関係として、メタメッセージと遷移の関係がある。

メタメッセージは他のメタインスタンスへの通信を示すものである。ゆえに、他の Classifier の状態へ影響を与える遷移、つまりメタスイムレーンをまたぐ遷移とメタメッセージは関係がある。例えば、アクティビティ図でのアクターＡのアクション状態からアクターＢへアクション状態への遷移と、シーケンス図でのアクターＡからアクターＢへのメッセージの関係である。ただし、振舞図と相互作用図は共に動的な振舞を示す手段の図である。そのため、メタメッセージと遷移は主従関係がない関係である。

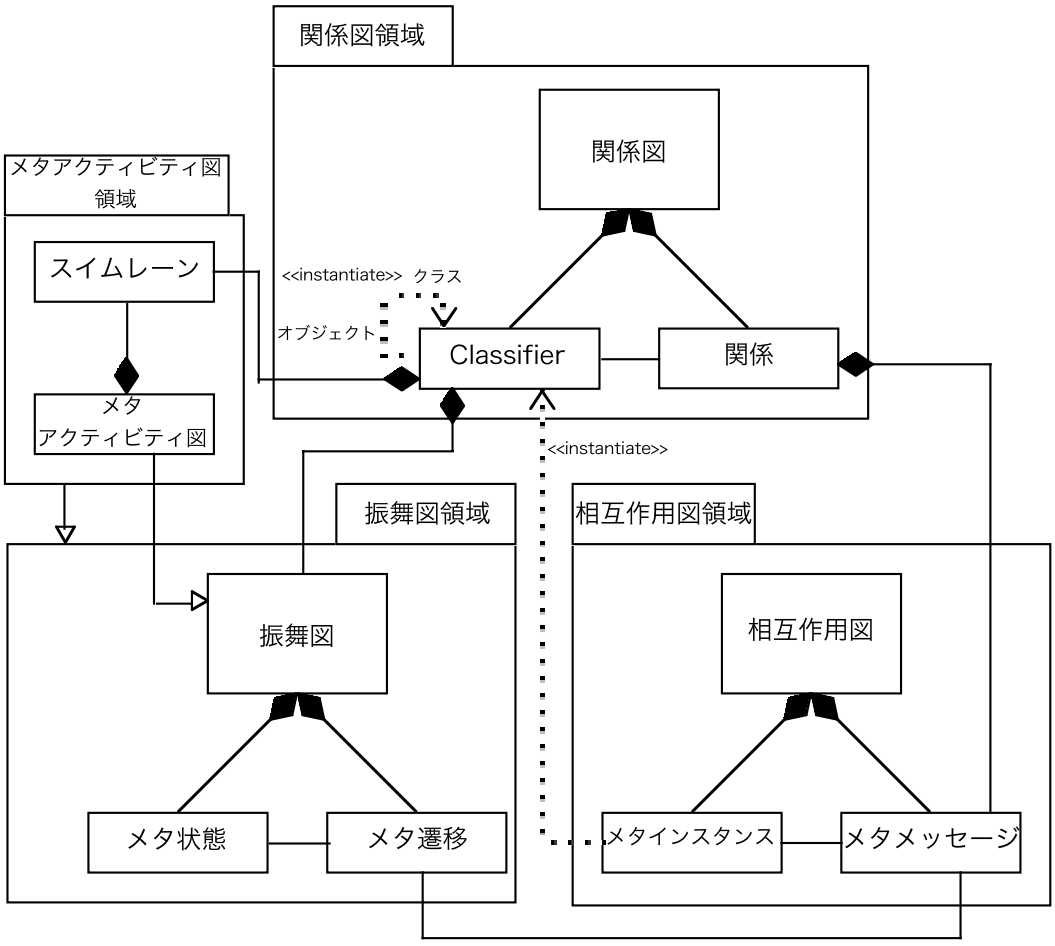


図 3.10: UML 文書の変更管理モデル ー 図面間の関係

第4章　有効性の確認

ここで簡単な具体例を用いて、人間が定義したUML図面間の依存関係と、変更管理モデルを用いて生成されたUML図面間の依存関係を比較し、この手法の有効性を示す。

4.1　具体例

具体例として、自動車の巡航制御・監視システムを設計したUML文書[7]の一部を用いた。対象としたUML文書は、Figure20.2からFigure20.16までの15枚で、以下にそれぞれのUML文書の内容を示す。

2. 巡航制御ユースケースパッケージ（ユースケース図）
3. 監視システムユースケースパッケージ（ユースケース図）
4. 静的オブジェクト（クラス図）
5. システムに使うクラス（クラス図）
6. パッケージ間の関係（クラス図）
7. サブシステムとクラスの関係（クラス図）
8. ‘シャフト回転数の更新’ユースケースの実現（コラボレーション図）
9. ‘距離と速度の決定’ユースケースの実現（コラボレーション図）
10. ‘速度制御’ユースケースの実現（コラボレーション図）
11. ‘Cruise Control’オブジェクトの状態遷移（状態遷移図）
12. ‘キャリブレーション値の調整’ユースケースの実現（コラボレーション図）
13. ‘Calibration Control’オブジェクトの状態遷移（状態遷移図）
14. ‘巡航速度の計算とリセット’ユースケースの実現（コラボレーション図）
15. ‘巡航燃料消費の計算とリセット’ユースケースの実現（コラボレーション図）

16. ‘オイルメンテナンスの調査とリセット’ユースケースの実現（コラボレーション図）

ただし、巡航操縦ユースケースパッケージは‘シャフト回転数の更新’、‘距離と速度の決定’、‘速度制御’、‘キャリブレーション値の調整’のユースケースで構成され、監視システムパッケージは‘巡航速度の計算とリセット’、‘巡航燃料消費の計算とリセット’、‘オイルメンテナンスの調査とリセット’ユースケースで構成される。また、‘Cruise Control’オブジェクトは‘速度制御’ユースケースで用いられ、‘Calibration Control’オブジェクトは‘キャリブレーション値の調整’ユースケースで用いられる。

4.2 変更管理モデルの適用

UML 図の依存関係モデルを用いて、UML 図面群に依存関係を付加した。その結果を以下に示す。

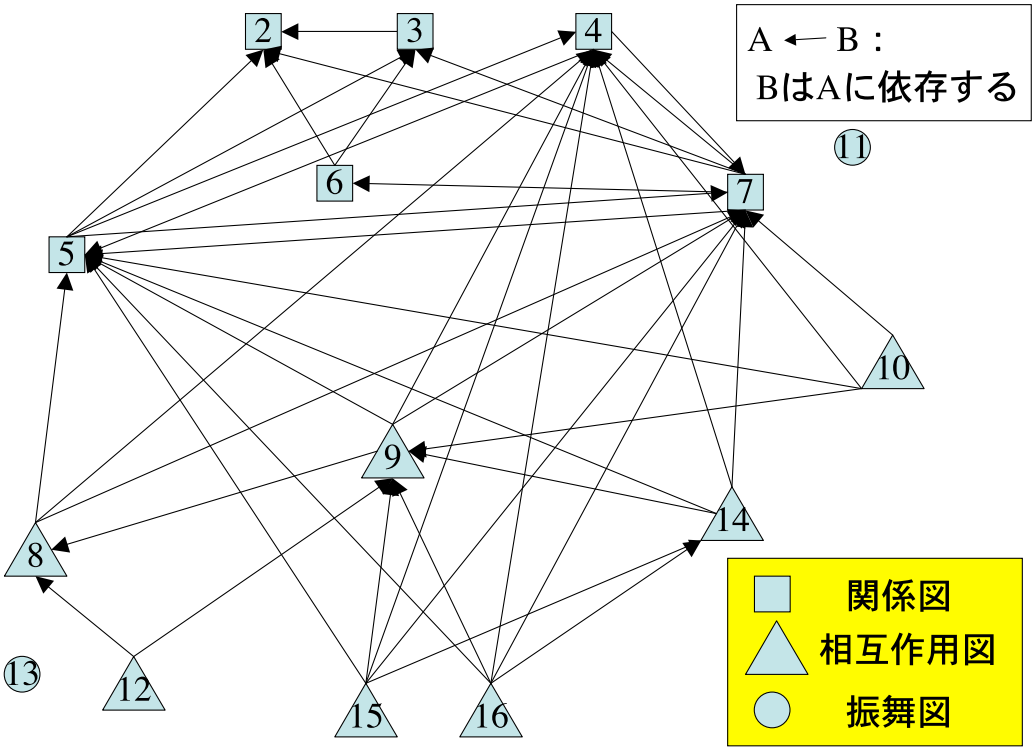


図 4.1: 変更管理モデルを利用して生成した依存関係

4.3 比較

UML 図の依存関係モデルを用いて付加した UML 文書間の依存関係と、人間が定義した UML 図面間の依存関係を比較して有用性を検証する。

ここで、人間が定義した UML 図面間の依存関係を示す。

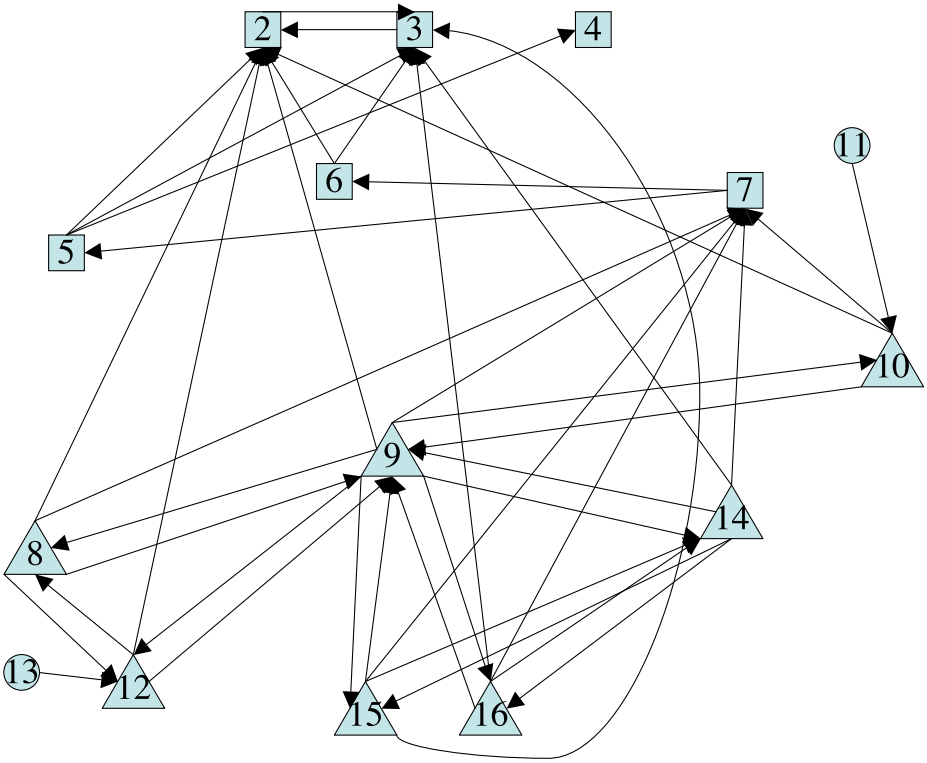


図 4.2: 人間が定義した依存関係

UML 図の変更管理モデルを用いて生成した依存関係と、人間が付加した依存関係の和集合を、UML 図面間に必要な依存関係とする。双方が定義した依存関係は正しいものとする。以下にそれぞれが付加した依存関係数を示す。

表 4.1: 付加した依存関係数

	変更管理モデルを利用して生成した依存関係	人間が定義した依存関係
依存関係数	48	42
一致した依存関係数	33	

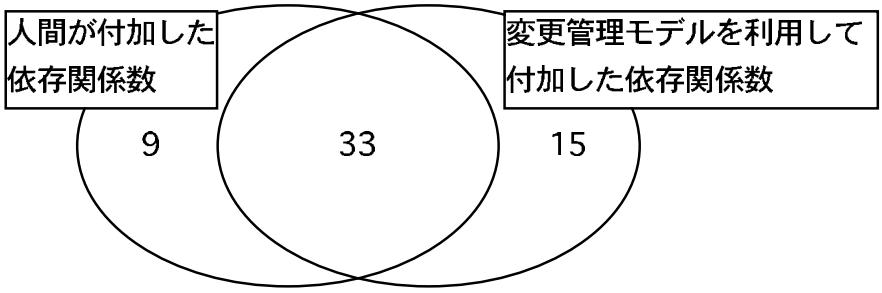


図 4.3: 依存関係数の集合

双方が付加した依存関係数は 33 であった。変更管理モデルを用いて生成できなかった依存関係は 9、変更管理モデルを用いて生成できたが人間が定義しなかった依存関係は 15 あった。どちらか定義した依存関係をそれぞれ吟味する。

4.3.1 変更管理モデルを用いて生成しなかった依存関係

変更管理モデルを用いて生成できなかった依存関係を分類すると、以下の 2 つに分類できる。以降、それぞれを吟味する。

- Classifeir と相互作用図の間に関係を定義していなかったため
- インスタンスと振舞図の間に関係を定義していなかったため

Classifeir と相互作用図の間の関係

具体的には、ユーザはユースケースと問題領域オブジェクト間の依存関係を定義した。しかし、変更管理モデルにはこの依存関係を生成する関係、つまり Classifier と相互作用図面間の関係を定義していない。

変更管理モデルでは、Classifier とオブジェクト間の関係を定義している。具体的に、これより生成する UML 図面間の依存関係は、クラス図とそれで示された関係をインスタンス化し、メッセージを付加したコラボレーション図やシーケンス図である。この変更管理モデルにおいて定義している関係図と相互作用図の関係は「相互作用図は、関係図で定義した Classifier 間のコミュニケーションを示す図」である。このとき Classifier から相互作用図領域への変更に関する依存は、Classifier をインスタンス化したオブジェクトである。

一方、ユーザが付加した依存関係は、変更管理モデルでは Classfier と相互作用図の関係である。このとき、関係図と相互作用図の関係は「相互作用図は、関係図で定義した Classifier の具体的な活動を示す図」となる。このとき Classifier から相互作用図領域への変更に関する依存は、Classifier の活動を示す相互作用図となる。

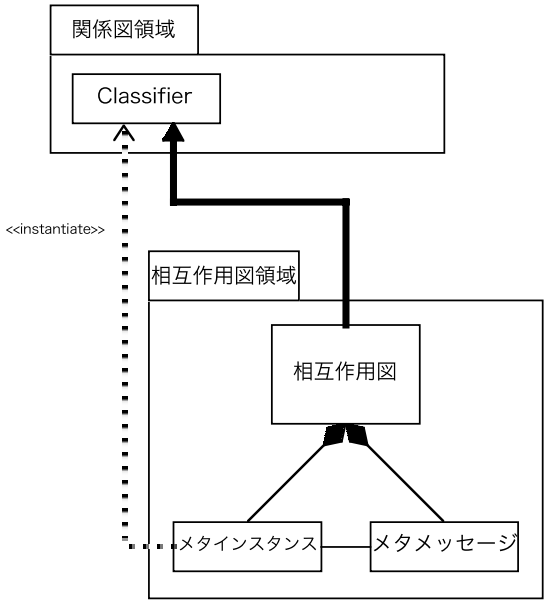


図 4.4: 未定義の関係：Classifier と相互作用図の関係

以上より、Classifier から相互作用図領域への変更に関する依存は２つ存在する。この２つの依存関係を生成する関係を、変更管理モデルに付加する。

Classifier から振舞図領域への関係は、問題ではないので上位メタ要素「Classifier」で定義する。そして「Classifier」を継承したメタ要素を、それぞれ仮に「静的 Classifier」、「動的 Classifier」と名付ける。「静的 Classifier」は、相互作用図領域の「オブジェクト」と同一（コピー）の関係を定義する。「動的 Classifier」は、相互作用図領域の「相互作用図」と生存の主従の関係を定義する。このように変更管理モデルを変更することで、この問題は解決できる。

ただし、修正前の Classifier をメタ要素としていた UML 図要素を、静的 Classifier と動的 Classifier に分類する必要がある。直感的には、ユースケースのメタ要素が動的 Classifier、残りの UML 図要素のメタ要素が静的 Classifier である。しかし、これに関して検討する必要がある。

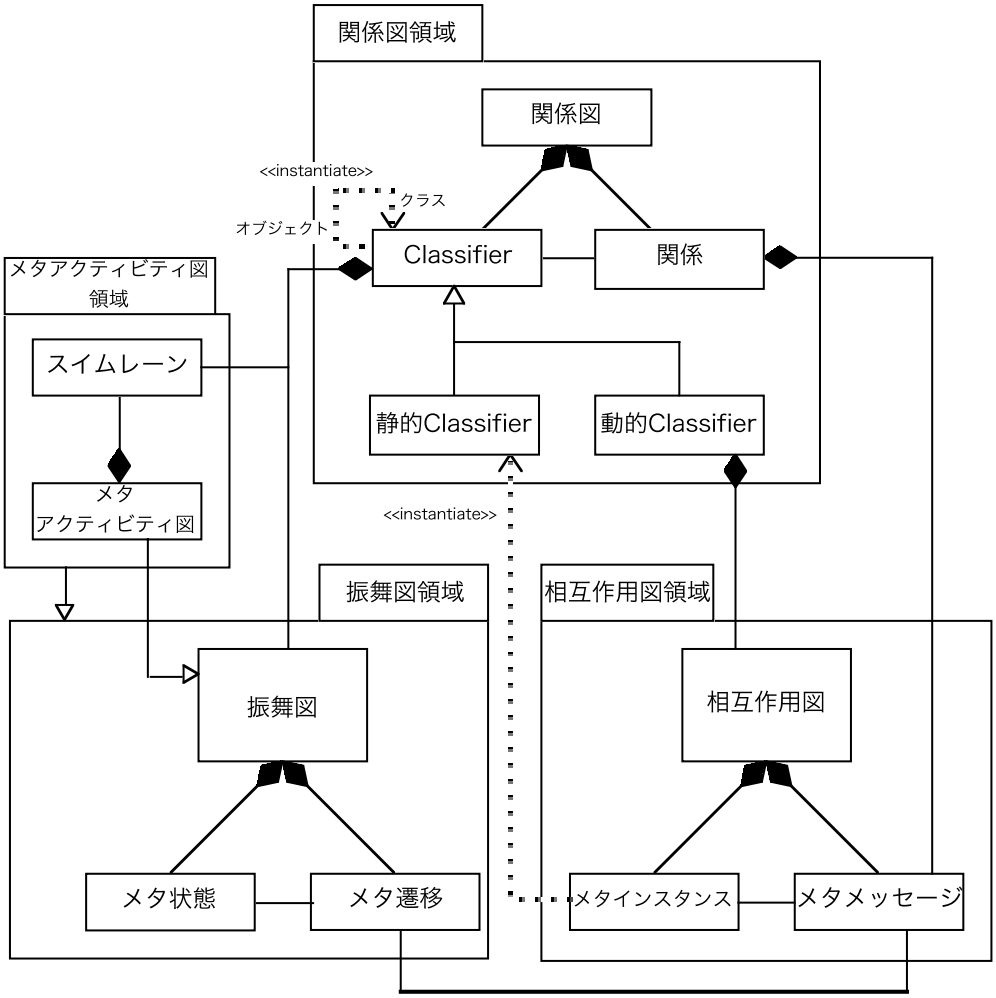


図 4.5: Classifier を詳細化した変更管理モデル

メタインスタンスと振舞図の関係

具体的には、クラスとその振舞を示す状態遷移図が依存関係をもつとき、クラスをインスタンス化したオブジェクトも状態遷移図と依存関係がある。しかし、変更管理モデルにはこの依存関係を生成する関係、つまりメタインスタンスと状態図の関係を定義していない。

これは、変更管理モデルがメタ要素間の関係をどの程度、定義するかによって解決する方法が異なる。

変更管理モデルで定義する関係を基本的な関係だけとし、推論できる依存関係は実装したシステムに生成させるとする。この場合、変更管理モデルに関係を加えず、このままでよい。一方、存在するすべての関係を変更管理モデルに定義するとする。この場合、推論できる関係は、いまはメタインスタンスと状態図の関係だけなので、モデルが煩雑にならないと考える。しかし、今後、推論できる関係が増加することもありうる。ゆえに、変更管理モデルで定義する関係をどの程度にするか、検討中である。

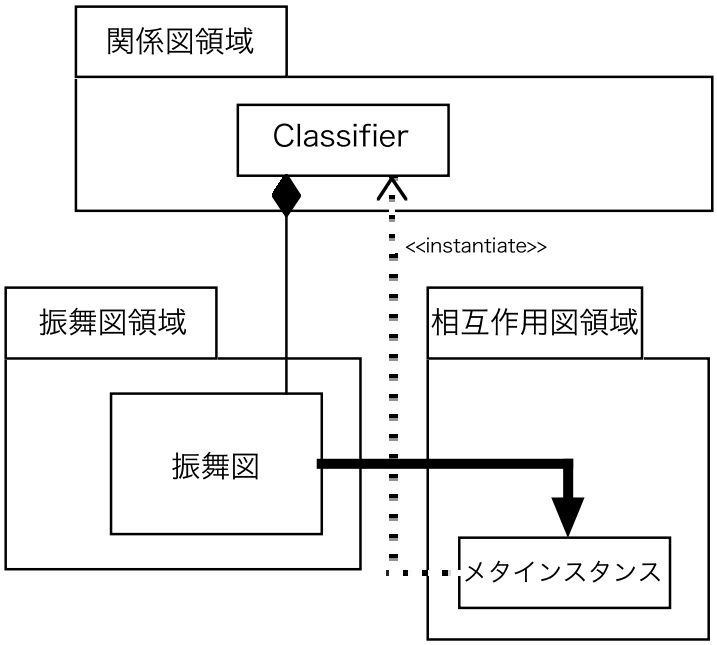


図 4.6: 未定義の関係：メタインスタンスと振舞図の関係

4.3.2　人間は定義せず、変更管理モデルを利用して生成した依存関係

具体的には、複数のクラス図において用いられているクラスが、インスタンス化したオブジェクトとを持つコラボレーション図と関係がある。しかし、そのクラスをもつすべてのクラス図とコラボレーション図に依存関係を定義していなかった。この関係は人間が故意につけなかった依存関係かもしれない。その理由として、同一のクラスを用いる場合、初めにそのクラスを書いたクラス図から変更する、など考えられる。これは、UML 図を書く設計者のポリシーによるものと考えられる。

一方、変更管理モデルでは、クラスのメタ要素である Classifier とオブジェクトのメタ要素であるメタインスタンスの間に関係を定義し、Classifier の尾や要素であるモデル要素に再帰関係でコピーの同一を定義している。これより、クラスとオブジェクトの関係が存在すると、そのクラスとコピーの同一関係を持つクラスは同一の名前を保持するため、同じオブジェクトと関係をもつ。ゆえに、同一のクラスを持つすべて図クラス図と、そのクラスのオブジェクトを保持するコラボレーション図に依存関係を付加した。

これは変更管理モデルを利用して依存関係を付加する手法の利点と考えられる。

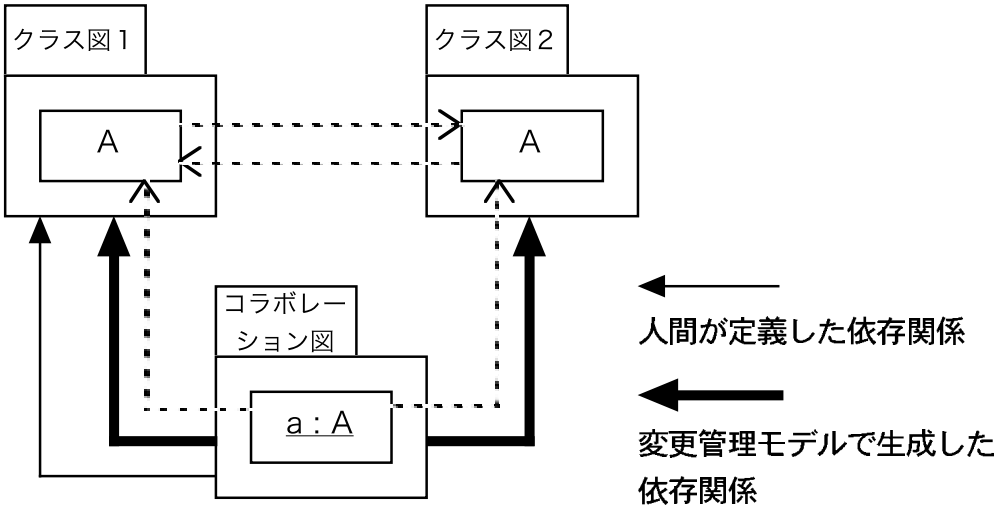


図 4.7: 人間は定義せず、変更管理モデルで生成した依存関係

4.4　評価

変更管理モデルを利用して付加した依存関係と、人間が定義した依存関係がほとんど一致した。一致しなかった部分が生じた展開は、変更管理モデルがまだ完全でないことを示しているが、本研究の方向性の有効性は確認された。

第5章　おわりに

5.1　まとめ

本論文は、UML 図の変更管理モデルを定義し、変更管理モデルを利用して UML 図面間の依存関係を生成する方式を提案した。まず、UML 図と図の構成要素に対応するメタ要素を定義し、メタ要素間に用いる関係としてコピーの同一関係、インスタンス化の同一関係、および生存の主従関係を定義した。細粒度でメタ要素を定義したため、１枚の UML 図と対応するメタ要素の集合を領域として定義し、領域を越える関係を UML 図面間の依存関係として生成する。つぎに、メタ要素間の関係を定義し、UML 図の変更管理モデルを定義した。この手法を具体例に適用した結果、ほとんどの依存関係が一致した。この手法で生成できなかった依存関係は２種類あり、Classifier が粗粒度であることと、推測可能な関係をモデルに定義するのかという検討課題を得た。また、この手法で生成したが人間は定義しなかった依存関係もあり、要素のコピーによる依存関係の定義し忘れ防止を支援するというメリットを得た。変更管理モデルはまだ不完全であるが、本研究の有効性が示された。

5.2　今後の課題

依存関係を「変更する必要がある関係」として定義した。つまり、依存関係を追跡することで、変更作業を示すワークフローであると解釈することもできる。この観点から課題を述べると以下の２つがあげられ、それぞれについて述べる。

- UML 図要素間に依存関係を生成するメタ要素間の関係の十分性
- 共同開発における変更プロセスの生成の支援

UML 図面間に依存関係を生成するメタ要素間の関係の十分性

UML 図面間に依存関係を生成するメタ要素間の関係として、コピーの同一関係、インスタンス化の同一関係、生存の主従関係の３種類を定義した。具体例での有効性の確認において、本論文においては十分であると認識した。しかし、まだ変更管理に使える関係が存在する可能性がある。例えば追加・更新に関する関係が存在する可能性がある。今後、追加・更新の変更も含めて、さまざまな例を用いて、十分性を確認する。

変更プロセスの生成の支援

本論文で、簡単な具体例として１５の UML 図面間の依存関係を付加した。実験結果からも理解できるように、複雑に依存関係は存在する。UML 図面間に張られた依存関係を追跡すると、ほぼすべての UML 図面を追跡することになり有用ではない。なぜなら、図面間にはられた依存関係では、ほぼすべての UML 図を追跡することになり、依存関係を付加した利点を生かすことができない。そのため、変更の内容により追跡する依存関係を限定する手法を考える。その１つに、細粒度での依存関係の付加がある。これにより行える変更プロセスの支援を２つあげる。

ひとつは、UML 図の変更作業内容を考慮した変更に関するワークフローの生成である。UML 図の変更作業は、メタ要素の追加・削除・更新のいずれかに相当する。しかし、それぞれの変更において、追跡する関係は異なると考える。直感的に、メタ要素の削除における変更のワークフローは本論文で定義した２種類の関係を追跡したワークフローを考える。なぜなら生存の主従関係も同一の関係も、定義より明白である。このように、メタ要素の追加・更新においても同様に追跡する関係を検討する。

また、共同開発の場合、複数の変更プロセスが同時並行的に進められ、互いに関連しあう場合が多い。作業の同期制御を行う場合、制御の判断材料として変更内容があると考える。直感的には、前述した変更に関するワークフローで UML 図要素を用いた競合判断し、変更作業の優先順位を決定することなど検討する。

謝辞

本研究を行うにあたり、多大な御助言、ご指導を賜りました北陸先端科学技術大学院大学 落水 浩一郎 教授に深く感謝の意を表します。

本研究を進めるにあたり、 種々の有益な御助言をいただきました、 情報科学センター 藤枝 和宏 博士、落水研究室 藤田 充典 氏をはじめ、研究を行う上で非常に役立つご意見、ご感想を寄せていただいた落水研究室の皆さまに深く感謝いたします。

本論文の審査にあたり、本学 篠田 陽一 教授、権藤 克彦 助教授には、種々の有益な御助言をいただきました。深く感謝いたします。

最後に、学業だけでなく、私生活の面からもご支援してくださいました家族、友人に深く感謝いたします。

参考文献

[1] Object Management Group：“Unified Modeling Language (UML)1.4”, http://www.omg.org/ technology/ documents/formal/uml.htm

[2] Ivar Jacobson, Grady Booch, James Rumbaugh：“UML による統一ソフトウェア開発プロセス”、翔泳社、2000.

[3] H. Gomaa, “Designing Real-Time Applications with the COMET/UML Method”, Proc. Workshop on Formal Design Techniques for Real-Time UML, UML 2000 Conference, York, England, October 2000.

[4] Rational Software Coorporation：“Rational Rose”、http://www.rational.com/

[5] The IIOSS Consortium：“IIOSS”、http://www.iioss.org/

[6] Dragan Milicev：“Automatic Model Transformations Using Extended UML Object Diagrams in Modeling Environments”, IEEE Trans. Software Eng., vol. 28, no. 4, pp.413-431, April. 2002.

[7] Hassan Gomaa：“Designing Concurrent, Distributed, and RealTime Applications with UML”, Addison-Wesley, 2000.

[8] Object Management Group：“XML Metadata Interchange(XMI) 1.2”, http://www.omg.org/technology/documents/formal/xmi.htm

[9] Grady Booch：“ UML ユーザーガイド ”、ピアソン、1999.

[10] 落水 浩一郎：“ソフトウェア分散共同開発のためのチーム構造モデルの定義と調整支援への応用”、情報処理学会、ソフトウェア工学研究会、SE-131、pp.17-24、2001.

[11] H.E Eriksson, Magnus Penker：“UML ガイドブック”、エスアイビー・アクセス, 1998.

[12] 具志堅隆児, 垣花一成, 倉骨彰：“実践的 UML 入門”, ASCII, 2002.

[13] Terry Quatrani：“UML 活用型開発入門” , トッパン, 1998.

[14] Graig Larman：“実践 UML”, ピアソン・エデュケーション, 1998.

本研究に関する発表論文

[1] 小谷 正行、落水 浩一郎：“UML 文書の変更管理モデル”、電気電子通信学会、ソフトウェアサイエンス研究会、SS2002-41、2003.