JAIST Repository
https://dspace.jaist.ac.jp/

| | |
|---|---|
| Title | 整合性を考慮したUMLモデル支援環境の研究 |
| Author(s) | 馬場, 茂雄 |
| Citation | |
| Issue Date | 2001-03 |
| Type | Thesis or Dissertation |
| Text version | author |
| URL | http://hdl.handle.net/10119/1461 |
| Rights | |
| Description | Supervisor:片山 卓也, 情報科学研究科, 修士 |

JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY

Japan Advanced Institute of Science and Technology

修 士 論 文

# 整合性を考慮したUMLモデル支援環境の研究

指導教官　片山 卓也 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

馬場 茂雄

2001 年 3 月

Copyright © 2001 by Shigeo Baba

# 目 次

# 図 目 次

# 第1章　はじめに

## 1.1　研究の背景と目的

システムの複雑化，大規模化に伴い，ソフトウェア開発における有効なものとしてオブジェクト指向方法論が注目されている．オブジェクト指向方法論は，構造と操作を一体化したオブジェクトを用いて実世界の概念をモデル化する方法である．これまで，オブジェクト指向方法論として，Booch 法，OMT 法，OOSE 法などの多くの方法論が提唱されてきた．これらの方法論は，類似した概念を多く含んでいたにもかかわらず，異なった記法が用いられていた．Rational Software の Grady Booch と James Rumbaugh は，1994 年に方法論を統合した” Unified Method” の作成を開始し，後にこの作業に Ivar Jacobson が加わった．しかし，方法論の統合は困難であることに気づき，方法論の統合に先立って，オブジェクト指向方法論の記法のみを統一することを目指した．その成果物として，“UML(Unified Modeling Language)” が誕生した．1997 年，Rational Software 社は OMG Analysis および Design Task Force に UML バージョン 1.0 を提出し，その後，Rational 社はこのバージョンの UML に修正を加え，UML バージョン 1.1 として OMG 標準の案を提出した．これを OMG(Object Management Group) はオブジェクト指向の表記法として標準化した．よって，UML は現在，オブジェクト指向開発における，分析，設計のためのモデリング言語として広く定着している．理解性を高めるため，UML にはシステムを異なる側面から表現した 9 種類のダイアグラムが用意されている．これらのダイアグラムは，それぞれ独立性が高いものではあるが，完全に独立しておらず，モデル作成の際に異なる種類のダイアグラム間，同種の複数のダイアグラム間において矛盾が生じる可能性がある．オブジェクト指向方法論は，複雑なシステム，および大規模なシステムの開発に採用されることが多い性質のため，ダイアグラム間の矛盾のチェックを人間の目で行なうことは困難な作業である．以上のような理由のから，ダイアグラム間の整合性のチェックはツールによる支援が必須である．これまでにもモデルの整合性をチェックする機構を備えた CASE ツールは存在していたが，それらは異なる種類のダイアグラムの整合性のチェック，モデルに対する制約のチェックまではサポートしてはいない．UML は記法のみを定義したものであり，使用する方法論によって，またはソフトウェアを開発する組織によって，独自にモデルに制約を与える状況は頻繁に発生する．

そこで，本研究では UML のシンタックスのみに着目し，モデルに対する制約および矛盾をチェックするための支援環境を提案する．

## 1.2　論文の構成

本稿では，UML のシンタックス上の整合性をチェックするためのアプローチを提案する．本論文の構成は以下の通りである．

2 章 ソフトウェア開発の現状について説明し，モデルの整合性チェックの必要性を述べる．その後，本研究で提案する UML 支援環境の概要と不整合検出のアプローチについて，その指針を示す．

3 章 本研究において，モデルの記述言語として対象にしている UML について，その概要について述べる．また，UML の標準の制約記述言語である OCL についてもその概要を説明する．

4 章 本研究で提案する環境で扱う関係データベース技術について，そのシステムの構成や理論，SQL などについて説明する．

5 章 モデルを関係データベースで管理するには，そのテーブルの仕様を定めなくてはいけない．本章では，モデルのテーブル化の指針を述べた後，すべてのモデル要素の，テーブルの属性名のリストを列記する．

6 章 OCL で記述された UML への制約を SQL に変換するためのフレームワークを述べる．

7 章 最後にモデル要素間の対応に関する不整合検出の適用例について，モデルとテーブルを提示し，OCL 式から変換した SQL 式で不整合検出が行なえることを示す．

# 第2章　UML支援環境の概要

## 2.1　ソフトウェア開発の現状

ソフトウェア開発が行なわれる際，それらの活動は何らかのプロセスに基づいて行なわれる．それは有名なプロセスを採用するかもしれないし，また，無名なプロセスが採用されるかもしれない．もしくは，開発に携わる技術者の経験に基づくプロセスであることも多く，それらを複合的に使用することも珍しくない．

プロセスとは，特定の目標を達成するために誰が，いつ，何をどのように実行するかを規定するものである [2]．品質のよいソフトウェアを効率的に開発するガイドラインを提供しているプロセスほど効果的なプロセスということができる．

オブジェクト指向開発で使用されるプロセスのほとんどが，その各工程の成果物としてモデルを作成する．モデルとは，システムを抽象化したものであり，対象にするシステムを一定の視点および一定の抽象レベルで表したものである．

どんな条件にも適用できるプロセスを開発することは困難である．よって，開発条件によって，最適なプロセスを選択しなければいけない．プロセスを選択する場合，組織的な条件，ドメインの条件，ライフサイクルの条件，技術的な条件などを考慮に入れる必要がある．

ソフトウェアの開発保守工程全般を，何らかのモデルによって抽象化したソフトウェアプロセスモデル (software process model) がある．その代表的なものとして，上流工程から下流工程への流れをモデル化した図 2.1 に示すウォータフォールモデルがある．

この他にも，スパイラルモデル，噴水モデルなど，様々な開発モデルが提唱されているが，大まかなプロセスの流れはウォータフォールモデルと同じである．

開発プロセスにも多少依存するが，各開発段階の成果物として作成されるモデルを図 2.2 に示す.

最近のオブジェクト指向開発では，ユースケース駆動のプロセスが流行している．代表的なプロセスとして，Rational Software 社の Ivar Jacobson，Grady Booch，James Rumbaugh が提唱している RUP(Rational Unified Process) がある．

ユースケース駆動のプロセスでは，開発者は，顧客の要求をユースケースモデルでユースケースとして捉えることから始める．次に分析を行ない，ユースケースを満たすシステムを設計する．このときには，まず，分析モデルを作成し，次に設計モデルと配置モデルを作成する．その後で，システムの実装モデルを作成する．実装モデルにはすべてのコードが含まれている．

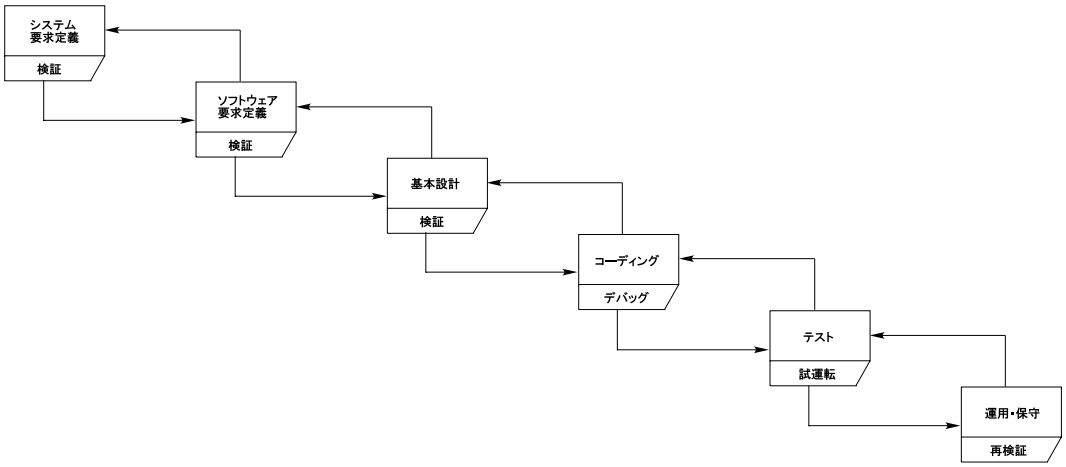


図 2.1: ウォーターフォールモデル

以下に，主に UML でモデルを作成する 3 つの工程について簡略に説明する．

- 要求分析
  ユーザの希望をヒアリングし，まとめ，その結果から理想的で実現可能なモデルを作成する．この段階での成果物はユースケースモデルである．UML ダイアグラムでは，主にユースケース図とアクティビティ図でモデルを表現する．この段階では，システムの実現方法は考慮しない．

- 分析
  要求分析で作成したユースケースモデルをもとに，実世界に近い形でシステムの静的構造，相互作用，振る舞いなどをモデル化する．その際，技術的なものや実装上の詳細などはモデルに含めない．分析者は，作成するソフトウェアのドメインに関する知識をユーザやドメインに関係する人達から獲得する必要がある．この段階での成果物は分析モデルである．分析モデルは要求の仕様を詳細に定義するものである．設計モデルの原案としての役割を担う．

  ここでのモデリング作業において，特にモデル間の整合性について，チェックする必要がある．モデルが矛盾が生じている状態で次の設計モデルを作成しても当然，矛盾が生じているモデルが作成されてしまう．一般に設計モデルよりも分析モデルの方が不整合なモデルになる可能性が高い．この段階でモデルの不整合を許してしまうと後で大きなコストを背負ってこの段階からやり直すことになる．よって，この段階での整合性チェックは重要である．また，ユーザインタフェースのプロトタイピングなどもここで行なうことが多い．分析段階で主に使用される UML ダイアグラムは，クラス図，シーケンス図，コラボレーション図，ステートチャート図，アクティビティ図である．

- 設計

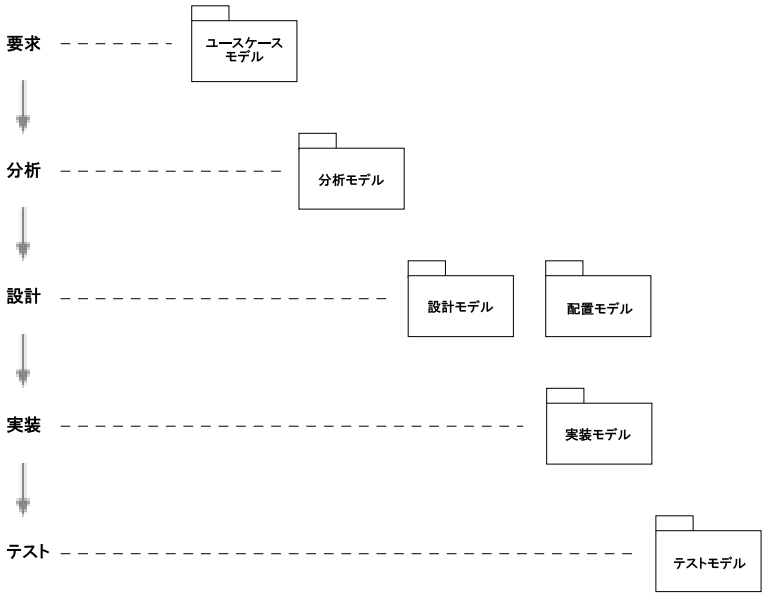


図 2.2: 各ソフトウェア開発段階で作成されるモデル

設計モデルでは，設計モデルのサブシステムと実装モデルのコンポーネントはそのまま対応している．この段階で初めてシステムのアーキテクチャを決定する．分析モデルと設計モデルの違いは，分析モデルが実装の構想を図示したものというよりも概念モデルと捉える方がよい．分析モデルと設計モデルの間にはシームレスな対応関係（対応追跡関係）がある．設計は分析の結果に対して，より技術的なものや実装上の詳細なども加味して，分析し，モデルを作成する．設計モデルでは，できる限り実装時のネーミングを使用するのが望ましい．設計モデルは，分析段階と同様のダイアグラムが使用される．

大規模で複雑なシステムを開発する際に，上記の開発過程でのモデリング作成時の整合性チェックは，人手を駆使して行なう場合，大変重要な作業であるにも関わらず，単調で退屈な作業であり，見落としやすく，さらに，時間的コスト，経済的コストを増大させる．整合性をチェックする作業は，ツールによる支援により，大きく効率を改善させる．

次節で本研究での整合性チェックを計算機によって支援するためのアプローチを示す．

## 2.2　整合性チェックのためのアプローチ

本研究で提案する環境では，図 2.3 のように，UML で使用されたすべてのダイアグラムを，単一の関係データベースシステムで管理する．通常，各ダイアグラ

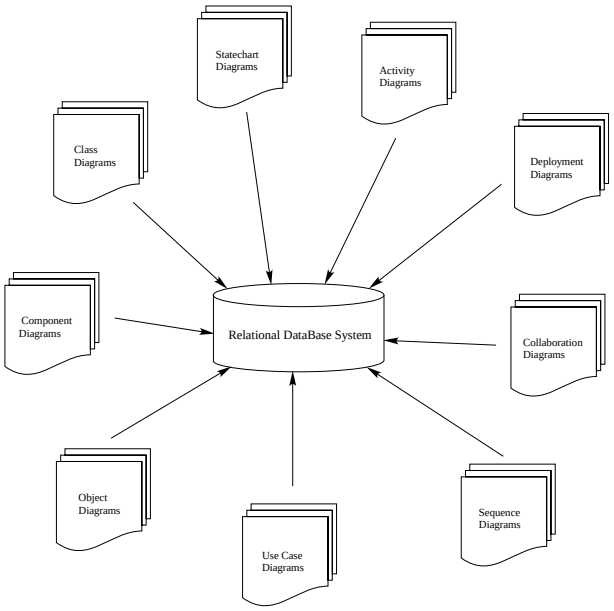


図 2.3: UML ダイアグラムの関係データベース一括管理

ムは，複数のモデルによって使用される．

関係データベースシステムを利用することで以下のようなメリットがある．

- 関係データベースは広く普及しており，導入が比較的容易である．
- 既存のシステムを利用するので開発する手間を大幅に削減できる．
- 一般に普及している技術であり，実績のあるシステムを利用できるため，信頼性が高い．

関係データベースシステム内では，モデルごと，もしくは，ダイアグラムごとに管理するのではなく，モデル要素 (Model Element) ごとに管理する．

モデル要素とは，UML メタモデルにて正式に定義されている．UML メタモデルについては，後の章で説明する．

モデル要素ごとに管理するとは，厳密に言えば，モデル要素ごとにテーブルを作成し，モデル要素のインスタンスごとにテーブルを作成することである．モデル要素のインスタンスとは，例えば，“Message” というモデル要素のインスタンスは，シーケンス図やコラボレーション図のダイアグラムを使用して作成されたモデル上の各々の “Message” を指す．

モデル要素ごとにテーブルを作成することで，UML の論理的な構造に基づいて作成でき，矛盾なくモデルをテーブルに納めることができる．また，後述する OCL(Object Constraint Language) は，UML の標準の制約記述言語であり，本環境でも OCL を使用するが，UML の論理的な構造がテーブルでも保たれているこ

とは，特に大幅な意味の変換をしなくてもシンタックスの変換のみで関係データベースへのアクセスができることを意味する．

整合性をチェックするには，どういった状態であることがシステムが整合している状態なのか，もしくは，どういった状態であることがシステムが不整合な状態なのかを規定しなくてはいけない．UML を使用する上での最低限の規則については，OMG Unified Modeling Language Specification [1] の Semantics の章に Well-FormednessRules として，OCL で記述されている．

例えば，Final State というモデル要素の Well-FormednessRule は，以下のように定められている．

Final State

A final state cannot have any outgoing transitions.

```
self.outgoing -> size = 0
```

また，Interface というモデル要素には，次のような Well-FormednessRule が定義されている．

Interface

[1] An Interface can only contain Operations.

```
self.allFeatures->forAll(f |
      f.oclIsKindOf(Operation) or f.oclIsKindOf(Reception))
```

[2] An Interface cannot contain any ModelElements.

```
self.allContents->isEmpty
```

[3] All Features defined in an Interface are public.

```
self.allFeatures->forAll ( f | f.visibility = #public )
```

これらの規則は，上でも述べたようにUMLを使用してモデリングをする際に最低限守るべき規則であり，Well-FormednessRulesには，例えば，あるモデル要素と同一のダイアグラムに現れないモデル要素との暗黙的な制約については，ほとんど記述されていない．これらの暗黙的な制約については，開発方法論にも依存するかもしれないし，モデリングを行なっている組織の独自の制約規則かもしれない．

組織の独自の制約やUMLの暗黙の制約とは例えば，以下のようなものが考えられる．

- あるモデル要素の特定の名前の禁止．
- クラス名とオブジェクト名の同一名の禁止．
- 次の状態へ遷移することができない状態の禁止．
- “Message” と同名の”Event” が存在しなくてはならない．
- 関連のないクラス間において，そのクラスのオブジェクト間ではメッセージの送受信は行なわない．

ここで，考えられる制約をすべて列挙することは，困難であり，また，論文の趣旨から逸れるのでこれくらいにとどめておく．

少し，解説を加えておくと,「あるモデル要素の特定の名前の禁止」は，例えば，A社では，“Computer” というクラス名は抽象的で混乱を招くので使ってはいけないという取り決めをしてモデルを作成する場合である．

「クラス名とオブジェクト名の同一名の禁止」は，クラスのインスタンスであるオブジェクトにあるクラスと同じ名前が使われていた場合，これもまた，混乱を招きかねない．よって，クラス名とオブジェクト名の同一名は使用してはいけないというような取り決めを行なう場合である．

「次の状態へ遷移することができない状態の禁止」は，実際には，そういうシステムも存在するが，例えば，B社でこれから作成するソフトウェアは，状態の遷移しないシステムは存在しないという前提のもとで開発を行なう場合である．

4つ目と5つ目の制約も，UMLの仕様には定義されていないが，モデルを記述する上での暗黙の一般的な制約である．

これまでに世に送り出されたモデリングツールは，ツールベンダーが定めた制約のみしかチェックすることしかできず，ユーザはそれを受け入れるしかなく，方法論や，組織の独自の制約をチェックするような機能は備えていなかった．しかし，本研究で提案する環境では，上で示したWell-FormednessRulesのような制約の記述を，ユーザが必要に応じてOCLで記述することによって，整合性チェックの項目をカスタマイズできる．

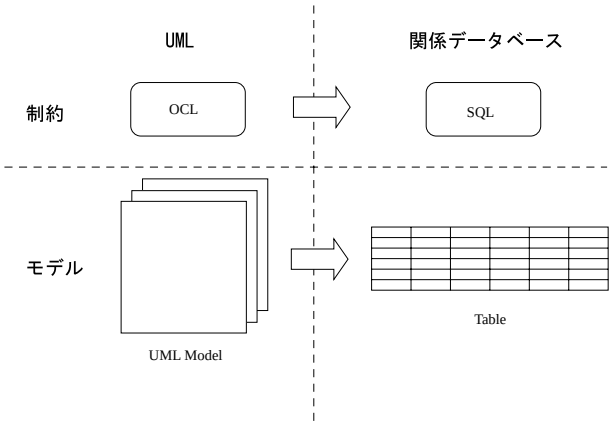


図 2.4: UML 領域から関係データベース領域へのマッピング

これは，UML の構文規則を構築する作業と等価であり，UML に対して制約を与えることと同じである．UML に対して制約を与えるという作業は，UML のメタモデルに対して不変条件を設定する作業に他ならない．

本環境では，ユーザが OCL で記述した UML への制約をほぼ等価な意味をもつ結果を出力する整合性判定のための SQL クエリーと，その不整合箇所を出力する SQL クエリーの 2 種類のクエリーに変換される．つまり，本環境では，図 2.4 のように UML の領域を関係データベースの領域にマッピングを行なうのである．関係データベースの世界にマッピングすることによって，関係データベース技術の利点を享受することができる．変換された SQL クエリー群は，ユーザがモデリング作業中に必要に応じて整合性チェックを行なう際に実行される．

まず，モデルに関する情報が納められているテーブルに対し，制約を満たしているかどうかを判定する SQL クエリーを発行し，その結果，“True” と判定された場合は，その結果を処理し，ユーザに結果を報告し，整合性チェックを終了する．もし，“False” と判定された場合は，さらに不整合箇所を出力する SQL クエリーを発行し，その制約を満たしていない箇所を抽出する．その結果を処理し，ユーザに報告する．

以下に上記の整合性チェックの手順を簡略にまとめたものを示す．

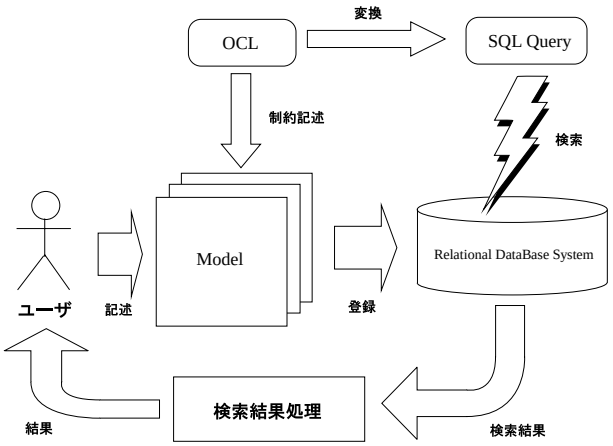


図 2.5: 整合性チェックシステムの概要

1. UML に対する制約条件を OCL で記述する．
2. OCL で記述された制約条件を整合性を判定するための SQL クエリーと不整合箇所を抽出するための SQL クエリーに変換する．
3. ユーザによってモデルが作成される．その際，モデルの情報は関係データベースに登録され，管理される．
4. ユーザが整合性チェックを行なう時，まず整合性を判定するための SQL クエリーを発行する．
5. 整合性を判定するための SQL クエリーを発行した結果，“True” であれば，結果を処理し，ユーザに示して終了する．
6. 整合性を判定するための SQL クエリーを発行した結果，“False” であれば，不整合箇所を出力するための SQL クエリーを発行する．
7. 不整合箇所を出力するための SQL クエリーを発行した結果を処理し，ユーザに示して終了する．

図 2.5 に整合性チェックシステムの概要を示す．

システムのアーキテクチャとしては，UML への制約は，制約定義ファイルにすべて記述しておき，この制約定義ファイルを OCL-SQL コンバータで SQL クエリーの集合に変換し，整合チェッククエリーファイルに保存する．整合性チェックを行なう際は，整合チェッククエリーファイルの SQL クエリー群を順次実行する．その後，制約定義ファイルに制約を追加するごとに OCL-SQL コンバータで SQL クエリーに変換し，整合チェッククエリーファイルに保存する． OCL から SQL へ

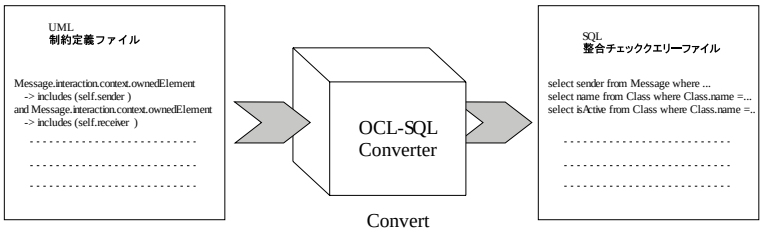


図 2.6: OCL-SQL Converter

の変換についての説明や，整合性チェックの適用例などについては，後述する．

# 第3章　UML

## 3.1　UMLの目標

UMLの開発の主な目標は以下の通りである [1].

- ユーザがわかりやすいモデルを開発/交換できるように，既成のビジュアルなモデリング言語を提供する.
- 主概念を拡張するための拡張メカニズムと特殊化メカニズムを提供する.
- 特定のプログラミング言語や開発プロセスに依存しない.
- モデリング言語を理解するための形式的定義を提供する.
- ツール市場の成長を促進する.
- コラボレーション，フレームワーク，コンポーネントなどのハイレベルな開発概念をサポートする.
- 最良の技術を統合する.

上記のように，UMLは計算機で支援することを意識して設計されている．しかし，特定の開発プロセス，プログラミング言語への依存を避け，さらには，拡張メカニズムと特殊化メカニズムを備えるなど，柔軟に対応できる設計がなされているため，UMLの機能を十分に活かすことができるツールを開発することをより難しくしている.

## 3.2　UMLの層構造

UMLのアーキテクチャはOMGのMOF(Meta-Object Facility)に基づいて，4つの層から成り立っている．このUMLの層構造を図3.1に示す．最下層には，ユーザーオブジェクト層 (user objects layer)があり，この層は例やドメインに特化したインスタンスを扱う層である．ユーザオブジェクト層の一つ上の層はモデル層 (model layer)である．モデル層は，分析，設計段階での成果物としてのUMLモデルを扱う層である．モデル層の上位の次の層は，メタモデル層 (meta model layer)と呼ばれている層である．この層は，UMLを定義している層であり，通常，単な

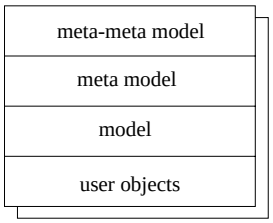


図 3.1: UML の層構造

る UML ユーザーはこの層を意識する必要はないがツール開発者はこの層を意識する必要がある．本研究でも UML の整合性をチェックする際はこの層で作業する．また，モデルに対して制約を加える場合もメタモデル層で作業を行なう．最上位層は，メタメタモデル層 (metameta model layer) である．この層は，メタモデルを定義している層である．つまり，この層で UML の構成要素を定義している．

## 3.3　メタモデル

本節では，本研究の作業に特に重要な意味を持っている層であるメタモデルについて説明する．メタモデル層のトップレベルは，基盤 (Foundation)，動的要素 (Behavioral Elements)，モデル管理 (Model Management) の 3 つのパッケージで構成されている．パッケージとは，ダイアグラム中の構成要素をグループ化したものである．

- 基盤パッケージ (Foundation Package)
  基盤パッケージには以下の要素が含まれている．
  - コア (Core)
  - データ型 (Datqa Types)

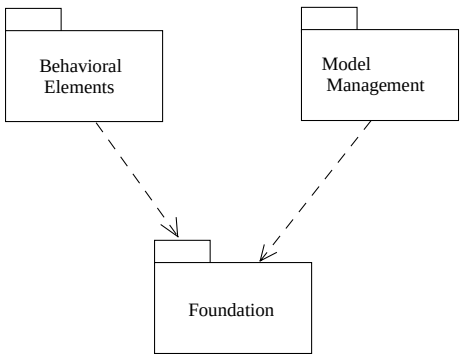


図 3.2: トップレベルパッケージ

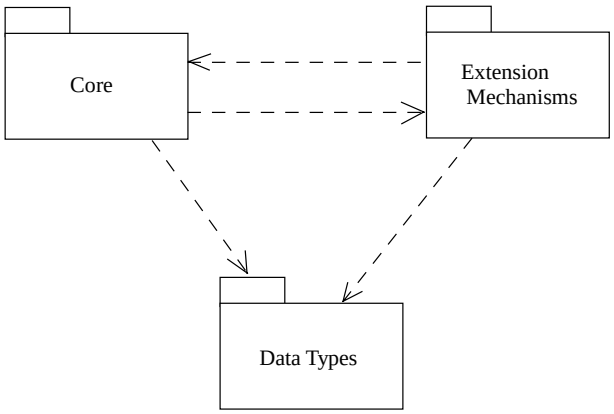


図 3.3: 基盤パッケージの構成要素の依存関係

- 拡張メカニズム (Extension Mechanisms)

これらの依存関係を図 3.3 に示す.

コアパッケージは，UML に必要な抽象メタクラスと具象メタクラスを提供する．インスタンスを作成できるものが具象メタクラス，インスタンスを作成できないものは抽象メタクラスと呼ばれる．例えば，抽象メタクラスには，モデル要素 (ModelElement)，汎化要素 (GeneralizableElement)，分類子 (Classifer) などを挙げることができる．また，具象メタクラスには，クラス (Class)，インタフェース (Interface)，関連 (Association)，データ型 (DataType) などを挙げることができる．

データ型パッケージでは，UML に integer，string，time などの列挙型を含む基本データ型を提供する．

拡張メカニズムパッケージは UML に拡張方法を提供する．

- 動的要素パッケージ (Behavioral Elements Package)
  動的要素パッケージには以下の要素が含まれている．
  - コラボレーション (Collaborations)
  - ユースケース (Use Cases)
  - 状態マシン (State Machines)
  - アクティビティグラフ (Activity Graphs)
  - 共通の振る舞い (Common Behavior)

  これらの依存関係を図 3.4 に示す．共通の振る舞いパッケージは動的要素に要求される核となる概念を提供する．コラボレーションパッケージは特定のタスクを成し遂げるために強調して働くモデル要素の振る舞いを規定してい

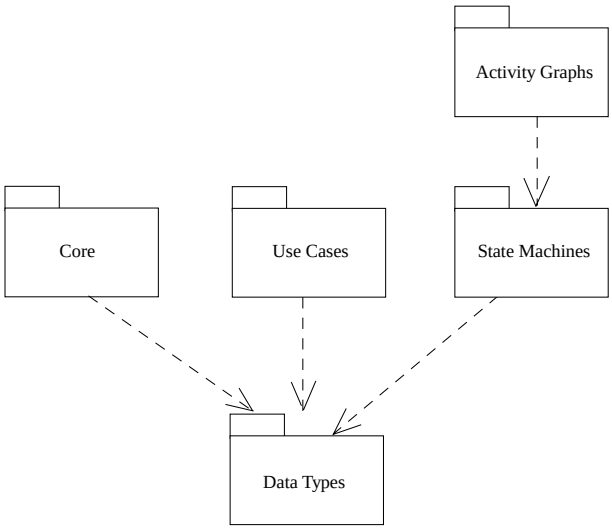


図 3.4: 動的要素パッケージの構成要素の依存関係

る．ユースケースパッケージは，アクターとユースケースの振る舞いを規定する．状態マシンパッケージは，遷移システムの最終状態へのシーケンスの振る舞いを定義している．アクティビティグラフパッケージは，プロセスを表すのに使われる状態マシンの特定のケースを定義している．

- モデル管理パッケージ (Model Management)
  モデル管理パッケージは，どのようにモデル要素がモデルやパッケージ，サブシステム中で組織されるかを規定するものである．

## 3.4　UML のビュー

UML には，5 種類のビューと 9 種類のダイアグラムが用意されている．ビューとは，システムをモデル化する際に着目するある側面のことであり，それぞれのビューは，単数もしくは，複数のダイアグラムで構成される．以下にそれぞれのビューについて簡略に説明する．

- ユースケースビュー
  アクターがシステムに要求すること，もしくは，システムがアクターに提供するサービスや機能について記述する．アクターとは，ユースケース図においてユーザもしくは，外部システムがユースケースとが相互作用する場合に演じる役割のことである．

- 論理ビュー
  システムが提供しなくてはならない機能を静的側面と動的側面とで記述する．

表 3.1: UML のビューで使用するダイアグラム

| ビュー | ダイアグラム |
|---|---|
| ユースケース・ビュー | ユースケース図 |
| 論理ビュー | クラス図，オブジェクト図，状態図，シーケンス図，コラボレーション図，アクティビティ図 |
| コンポーネント・ビュー | コンポーネント図 |
| 並行性ビュー | 状態図，シーケンス図，コラボレーション図，アクティビティ図，コンポーネント図，配置図 |
| 配置ビュー | 配置図 |

- コンポーネントビュー
  ソフトウェアコンポーネント（ソースコードやバイナリコードなど）間の依存関係を記述する．
- 並行性ビュー
  並列処理システムの通信や同期・非同期処理の問題を記述する．
- 配置ビュー
  物理的な装置の配置を記述する．

表 3.1 で各ビューで使用されるダイアグラムを示す．

特定のダイアグラムが複数のビューで使用されることからもわかるが，各ビュー内の情報は，ビューごとに独立してはいない．よって，ビュー間で不整合が発生する可能性がある．

## 3.5　ダイアグラム

ダイアグラムは，システムのある側面や特定の部分を表現するために使用され,．様々なモデル要素が組合さって構成される．システム開発において，通常，複数の種類のダイアグラムを記述する．また，同種のダイアグラムについても適宜，複数記述することができる．UML では，システムを表現するために以下の 9 種類のダイアグラムが用意されている．

- アクティビティ図 (Activity Diagram)
  アクティビティ図は，アクティビティの連続的な流れを表すためのダイアグラムである．アクティビティとは，何らかの振る舞いを表している状態のことである．
- オブジェクト図 (Object Diagram)
  オブジェクト図は，システムのある時点でのオブジェクトのスナップショッ

トとして使用される．表記法はクラス図とほぼ同じである．関連はインスタンスレベルではリンク (link) と呼ばれる．

- クラス図 (Class Diagram)
  クラス図は，システムのクラスの静的構造を表す．クラスとは，同一のデータ（属性），手続き（操作），関係および意味を共有するオブジェクトの集合である．オブジェクトは，それが属するクラスにより生成，破壊される．クラスとクラスの間には，関連，汎化，集約，依存などの関係が存在する．

- コラボレーション図 (Collaboration Diagram)
  コラボレーション図は，メッセージを送受信するオブジェクトの構造的な構成を協調した相互作用図である．

- コンポーネント図 (Component Diagram)
  コンポーネントとはシステム内の物理的なコードモジュールのことで，それらの依存関係が示される．

- シーケンス図 (Sequence Diagram)
  シーケンス図は，メッセージを送受信の時間順序を協調した相互作用図である．オブジェクトから下への垂直線を時間軸とし，その線に沿ってオブジェクト間で送受信されるメッセージの流れと順序を表現する．

- ステートチャート図 (Statechart Diagram)
  ステートチャート図は，クラスのオブジェクトが取りうるすべての状態と，状態から状態への変化を表すダイアグラムである．状態の変化のことを遷移とよび，その遷移にイベントとアクションを関連づける．イベントとは，オブジェクトの状態を変化させる事象のことで，アクションとは，状態遷移が発生した際に実行される手続きのことである．

- 配置図 (Deployment Diagram)
  配置図は，システム中のハードウェア，または，ソフトウェアの物理的なアーキテクチャを表すためのダイアグラムである．

- ユースケース図 (Use Case Diagram)
  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表すダイアグラムである．ユースケースとは，システムが提供する機能の記述であり，アクターとは，モデリングするシステムの外部に存在して，システムに対して何らかの影響を与える抽象実体である．

## 3.6　OCL(Object Constraint Language)

### 3.6.1　OCL の概要

OCL はモデルに対する制約を記述する言語である．制約は自然言語でも記述されるが，より曖昧性を排除するために IBM 社で開発された言語が OCL である．OCL は，強い数学的なバックグラウンドがなくとも，使用できるよう配慮して設計されている．また，OMG Unified Modeling Language Specification [1] の UML Semantics では，Well-formedness rules は OCL で記述されており，UML において標準の制約記述言語となっている．OCL は IBM 社と ObjecTime 社によって UML の構成要素として OMG に提案され，UML1.1 から導入された．特徴としては，OCL 式が評価される際に副作用を伴わないこと，OCL は型付きの言語であること，プログラミング言語ではないので，プログラミングロジックやフロー制御を記述できないなどである．OCL によって UML のメタモデル定義も以前に比べてより曖昧さが排除され整合的なものになった．

### 3.6.2　OCL の使用目的

OCL は主に以下のように使用される．

- クラス図の不変条件を記述する．
- ステレオタイプの不変性を記述する．
- 操作とメソッドの事前，事後条件を記述する．
- ガードを記述する．
- ナビゲーション言語として使用する．
- 操作の制約を記述する．

ガードとは，動的モデルにおいて，発火の際，満たさなければならない条件のことである．ナビゲーションは，関連する要素のプロパティを参照することを指している．

### 3.6.3　OCL の事前定義の型

OCL には, 事前定義の基本型として，Boolean 型，Integer 型，Real 型，String 型がある．また，コレクション型として，Collection 型，Set 型，Bag 型，Sequence 型がある．その他，OclType 型，OclAny 型，OclExpression 型が以上の型を補間する．以下に，基本型以外のコレクション型とその他の型について簡略に説明する．

- コレクション型
  - Collection 型
    Collection 型は，OCL におけるすべてのコレクションの抽象スーパータイプである．コレクション内のオブジェクトを要素と呼ぶ.
  - Set 型
    Set 型は，Collection 型のサブタイプである．重複した要素を含むことはできない．含まれる要素は順不同である.
  - Bag 型
    Bag 型は，Collection 型のサブタイプである．重複した要素を含むことができる．含まれる要素は順不同である.
  - Sequence 型
    Sequence 型は，Collection のサブタイプで，重複して要素を含むことができる．含まれる要素には順序がある.
- その他
  - OclType 型
    OclType 型は，モデルで定義されたすべての型と OCL の型のスーパータイプである.
  - OclAny 型
    OclAny 型は，モデルにおけるすべての型とコレクション型以外の事前定義の OCL 型のスーパータイプである.
  - OclExpression 型
    OCL 式は，すべて OclExpression 型として扱われる.

### 3.6.4　プロパティの参照

OCL 式においてプロパティを参照する場合，以下のように “.” に続けて参照するプロパティ名を記述します.

```
context [ Type Name ] inv:
self.[ Property Name ]
```

“[ Type Name ]” には，型名が入る．“[ Property Name ]” には，プロパティ名が入る．プロパティとは，その型が持つ属性や操作，ナビゲーションを行なう際

の関連終端ロール名を指す．また，OCL によって定義されているプロパティもある．ナビゲーションを行なう際は，参照する要素の関連終端ロール名をプロパティとして指定する．ナビゲーションを行なった先の要素でさらにナビゲーションを行なうことも可能である．ナビゲーションを 2 度以上続けるときは，さらに “.” を付加し，その後にナビゲーション先の関連終端ロール名を記述すればよい．以降，ナビゲーションとして使用する関連終端ロール名のことをナビゲーション・プロパティと呼ぶことにする．

コレクション型の要素に対して，OCL は，各種の操作を用意している．これらの操作は，“->” の後に操作名を記述する．例えば，“C” を Collection 型のインスタンスとし，“C” の要素数を調べるには以下のように記述する．

```
C -> size
```

“size” は，Collection 型のインスタンスの要素数を返す操作である．

# 第4章　関係データベース技術

## 4.1　データベースシステム

データベースとは，計算機に基づいた記録保持システムのことである．データベースを利用することによって以下のような利点を享受することができる．

- データの冗長性の除去
- データの一致性の確保
- データの共有化
- データ構造の標準化
- データの機密保護の容易性
- データの一貫性の確保
- 大量データの効率的管理
- 高機能言語による応用プログラム作成の容易化

データベースシステムは，DBMS，データベース，データベース言語，データモデルなどのデータベースを実現するために必要なものすべてを指し示すものである．データベースシステムを構成する要素として，データ，ハードウェア，ソフトウェア，利用者の 4 つの要素から成り立つ．

データベースシステムのモデルを図 4.1 に示す．

大量のデータを組織的に格納し管理する上では，種類の同じデータを一つの型としてグループ化し，それらが持つ共通の構造やその相互関連に注目することが有効である．このような抽象化に基づいてデータベース中のデータの構造，形式，関連，さらに各種整合性制約などを記述したものを，一般にスキーマ (schema) と呼ぶ．データの抽象化は，内部レベル，外部レベル，概念レベルの 3 つのレベルに分けられる．内部レベルは物理的なデータの格納のレベルに対応し，その構成を規定したものは内部スキーマ (internal schema) と呼ばれる．概念レベルはデータベース全体を論理的に記述したレベルに対応し，そのスキーマは概念スキーマと呼ばれる．外部レベルは個々のアプリケーションごとのデータベースに対する視点に対応し，そのスキーマは外部スキーマと呼ばれる．リレーショナルデータ

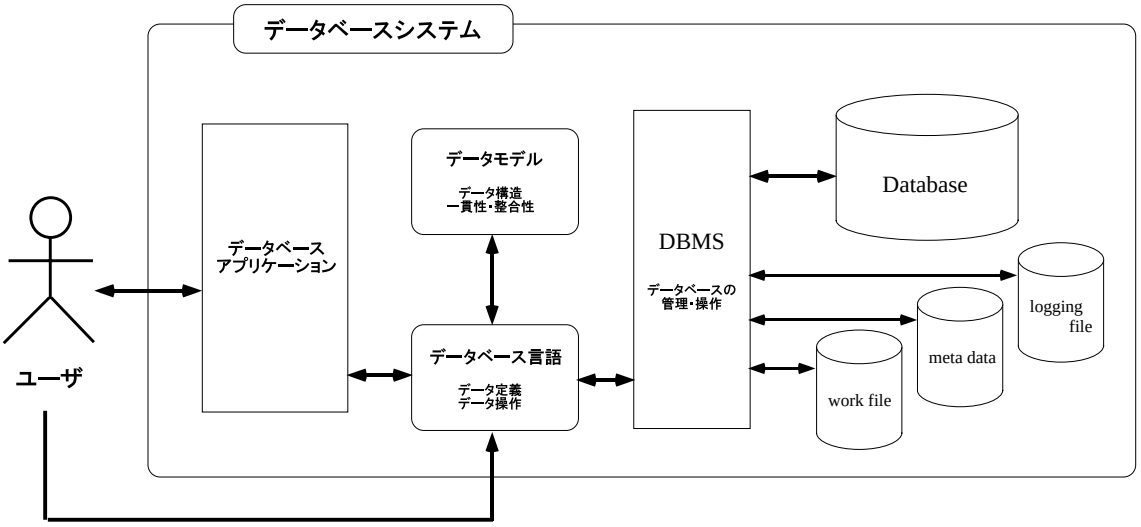


図 4.1: データベースシステム

モデルでは，外部レベルの記述のことはビューと呼ばれる．これら，概念スキーマ，外部スキーマ，内部スキーマの 3 つのスキーマアーキテクチャは，1978 年に ANSI/SPARC(American National Standards Institute/Standards Planning and Requirements Commitee) が提案したモデルであり，3 層スキーマモデルと呼ばれている．また，ISO(International Organization for Standardization) でも標準化されている．ANSI/SPARC の 3 層スキーマモデルを図 4.2 に示す.

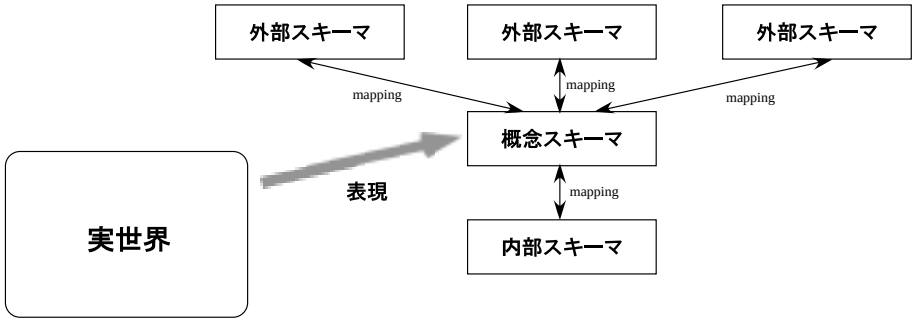


図 4.2: ANSI/SPARC の 3 層スキーマモデル

ANSI/SPARC の 3 層スキーマモデル以外にもスキーマとサブスキーマから成る 2 層スキーマというアーキテクチャも存在する．スキーマでは，データベース全体の論理構造と物理構造を定義する．サブスキーマでは，ユーザごとの視点に対応する．このアーキテクチャの弱点はスキーマに論理構造と物理構造の両方を定義してしまうことにある．

## 4.2　関係

関係データモデルは，1970 年に提案され，その単純性，数学的基盤の明解さ，それ以前のネットワークデータモデル，階層データモデルと比べた物理的レベルからの独立性の高さなどを特徴とする．リレーショナルデータモデルではデータベースを関係の集まりとしてモデル化している．関係の定義を定義 1 に示す．

**定義 1** 集合 $D_1, D_2, \cdots, D_n$（必ずしも相異なる必要はない）の集まりが与えられたとき，$R$ が順序付きの $n$ 項組 $< d_1, d_2, \cdots, d_n >$ の集合で，$d_1$ が $D_1$ に属し，$d_2$ が $D_2$ に属し，$\cdots, d_n$ が $D_n$ に属する場合，その $R$ をそれら $n$ 個の集合の上の関係 (relation) であるという．集合 $D_1, D_2, \cdots, D_n$ は $R$ の定義域 (domain) である．値 $n$ は $R$ の次数 (degree) である．

関係データベースのテーブルの各部の名称を図 4.3 に示す．

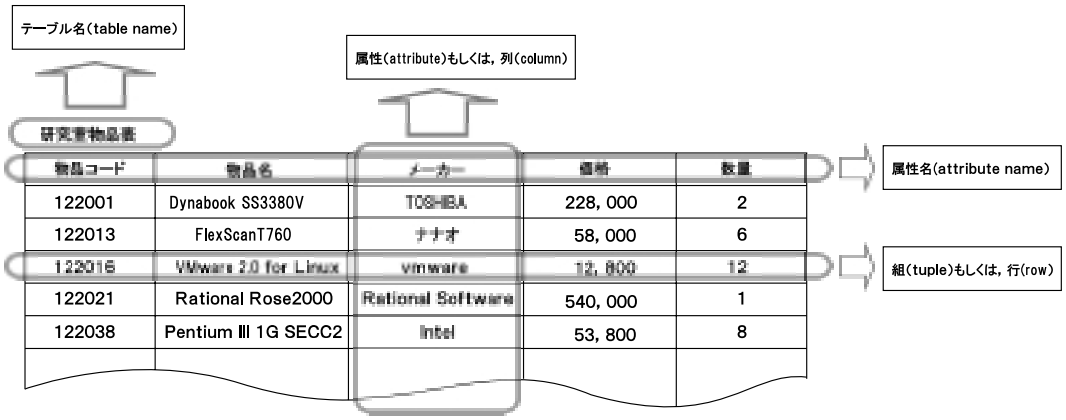


図 4.3: テーブルの各部の名称

関係データベースのインスタンスは組の集合により表現される．

## 4.3　関係代数

関係の操作を行なう方法として集合を対象とする演算を用いることが挙げられる．和，差，積，直積などの集合演算と関係操作のための演算を用いて関係の操作を行なう方法が関係代数である．以下に関係代数について説明する．

- 集合演算
  - 和集合演算 (union)
    和は 2 つの関係 $R(A_1, \cdots, A_n)$ および，$(S_1, \cdots, S_n)$ の和集合をとる二

項演算であり，その結果の関係 $R \cup S$ は，以下のように与えられる．

$$R \cup S = \{\ t \mid t \in R \vee t \in S\ \}$$

ただし，$R$ と $S$ の次数は同じでなければならない．なお，和演算の結果の関係が持つ属性名は $R$ と同一である．

- 差集合演算 (defference)
  差は 2 つの関係 $R(A_1, \cdots, A_n)$ および，$(S_1, \cdots, S_n)$ の差集合をとる二項演算であり，その結果の関係 $R - S$ は，以下のように与えられる．

$$R - S = \{\ t \mid t \in R \wedge \neg\, t \in S\ \}$$

  ただし，$R$ と $S$ の次数は同じでなければならない．なお，差演算の結果の関係が持つ属性名は $R$ と同一である．

- 積演算 (intersection)
  2 つの関係 $R(A_1, \cdots, A_n)$ および，$(S_1, \cdots, S_n)$ が与えられたとき，積 $R \cap S$ は以下のように与えられる．

$$R \cap S = \{\ t \mid t \in R \wedge t \in S\}$$

  ただし，$R$ と $S$ の次数は同じでなければならない．積演算は以下のように差でもって表現可能である．

$$R \cap S = R - (R - S)$$

- 直積演算 (cartesian product)
  2 つの関係 $R(A_1, \cdots, A_n)$ および，$(S_1, \cdots, S_m)$ が与えられたとき，直積 $R \times S$ は以下の関係を導出する二項演算である．

$$R \times S = \{\ t * u \mid t \in R \wedge u \in S\ \}$$

  ただし，$t * u$ は連結した組を表す．

- 関係演算
  - 選択演算 (selection)
    選択は，関係 $R(A_1, \ldots, A_n)$ がもつ組のうち，指定した条件を満たすものだけを残し，他を削除する単項演算である．選択条件 $F$ を用いた選択 $\sigma_F(R)$ は以下の関係を導出する演算である．

$$\sigma_F(R) = \{\ t \mid t \in R \wedge P_F(t)\ \}$$

    ただし，$P_F(t)$ は組 $t$ が選択条件 $F$ を満足するとき真となる述語とする．なお，選択演算の結果の関係が持つ属性名は $R$ と同一とする．

– 射影演算 (projection)
射影は関係 $R(A_1, \cdots, A_n)$ が持つ属性のうち，指定した属性だけを残し他を削除する単項演算である．$\{ A'_1, \cdots, A'_m \} \subseteq \{ A_1, \cdots, A_n \}$ のとき，射影 $\pi_{A_1,\cdots,A_m}(R)$ は以下の関係を導出する．

$$\pi_{A_1,\cdots,A_m}(R) = \{ t [ A'_1, \cdots, A'_m ] \mid t \in R \}$$

ただし，$t [ A'_1, \cdots, A'_m ]$ は組 $t$ から属性 $A'_1, \cdots, A'_m$ の値だけを残して他の成分を削除した組である．

– 結合演算 (join)
結合条件 $F$ を R の属性 $A$ と $S$ の属性 $B_j$ の比較演算子 $\theta$ による比較条件 $A, \theta, B_j$ とするとき，結合 $R \bowtie_F S$ は以下の関係を導出する二項演算である．

$$R \bowtie_F S = \{ t * u \mid t \in R \wedge u \in S \wedge P_F(t, u) \}$$

ただし，$t * u$ は組 $t$ と $u$ を連結した組を表し，$P_F(t, u)$ は $t$ と $u$ が結合条件 $F$ を満足するとき真となる述語とする．結合演算は以下のように，直積と選択の組合せとして表現できる．

$$R \bowtie_F S = \sigma_F(R \times S)$$

– 商演算 (division)
2 つの関係 $R(A_1, \cdots, A_n)$ および $S(A_m, \cdots, A_n)$（ただし，$1 < m \leq n$ で同じ名前の属性のドメインは同一であるとする）に対し，商 $R \div S$ は以下のように定義される．

$$R \div S = \pi_{A_1,\ldots,A_{m-1}}(R) - ((\pi_{A_1,\cdots,A_{m-1}}(R) \times S) - R)$$

## 4.4　DBMS(DataBase Management System)

DBMS は，データベースにアクセスするユーザが，快適に利用できるようなサービスを行なっているシステムである．DBMS の主な機能は以下の通りである．

- データベースの管理
データベースの定義と操作を行なう．

- トランザクション管理
データベースの操作履歴を管理する．

- 同時実行制御
同時に複数のユーザがデータベースを操作しても整合性が保たれるように管理する．

- 機密保護管理
  データベースに対するアクセスを管理する．例えば，アクセスが許されていないユーザはデータの参照や変更を行なえないように管理する．

- 障害回復管理
  障害発生時に，可能な限りデータの損失を最小限に抑えて復旧を行なう機能である．

## 4.5　SQL

SQL は，IBM で開発された Structured Query Language(構造化問い合わせ言語) の頭文字が語源である．その後，ISO(International Organization for Standardization) で標準化され，SQL は，略称ではなく固有名詞となった．

SQL には、大きく分けてデータ定義言語（Data Definition Language，DDL）とデータ操作言語 (Data Manipulation Language，DML) がある．

DDL で表名や列名、データ型などの定義，つまりデータベース自体の定義を行い，表に対して，DML でデータの追加，修正，削除，検索などの操作を行う．

### 4.5.1　データ型

SQL は，次のスカラーデータ型をサポートしている．

| | |
|---|---|
| CHARACTER($n$) | $n$ 文字で構成される固定長の文字列 ($n > 0$) |
| CHARACTER VARYING($n$) | $n$ 文字までで構成される可変長の文字列 ($n > 0$) |
| BIT($n$) | $n$ ビットで構成される固定長の文字列 |
| BIT VARYING($n$) | $n$ ビットまでで構成される可変長の文字列 |
| NUMERIC($p, q$) | $p$ 桁の 10 進数．$q$ 桁の少数値を伴う ($0 < -q < -p, p > 0$) |
| DECIMAL($p, q$) | $m$ 桁の 10 進数．q 桁の少数値を伴う ($0 < -q < -m, p > 0, m$) |
| INTEGER | 10 進またはバイナリ形式の符合付き整数 |
| SMALLINT | 10 進またはバイナリ形式の符合付き整数 |
| FLOAT($p$) | 浮動少数値 $N$，バイナリ仮数 $m$ の符号付きバイナリ指数 $e$ で表される |

上記の文字列の長さ ($n$)，精度 ($p$ と $m$)，位取り (q) は，すべて符号なし 10 進整数で指定しなければならない．上記のデータ型は以下のような省略形や別表記を使用することができる．

| | | |
|---|---|---|
| CHARCTER(1) | - | CHARACTER |
| CHARACTER | - | CHAR |
| CHARACTER VARYING | - | VARCHAR |
| INTEGER | - | INT |
| NUMERIC$(p, 0)$ | - | NUMERIC$(p)$ |
| NUMERIC$(p)$ | - | NUMERIC($p$ は実装時に定義される) |
| DECIMAL | - | DEC |
| DECIMAL$(p, 0)$ | - | DECIMAL$(p)$ |
| DECIMAL$(p)$ | - | DECIMAL($p$ は実装時に定義される) |
| FLOAT$(p)$ | - | FLOAT($p$ は実装時に定義される) |
| FLOAT$(s)$ | - | REAL($s$ は実装時に定義される) |
| FLOAT$(d)$ | - | DOUBLE PRECISION($d$ は実装時に定義される) |

### 4.5.2　集約関数

SQL には，5 種類の集約関数が組み込まれている．これらの関数は特定の集約 (aggregate) を操作する．集約とは，あるテーブルのある列のスカラー値の集合である．以下に 5 種類の集約関数についての戻り値を示す．

| | |
|---|---|
| COUNT | 列のスカラー値の数 |
| SUM | 列のスカラー値の合計 |
| AVG | 列のスカラー値の平均 |
| MAX | 列のスカラー値の最大値 |
| MIN | 列のスカラー値の最小値 |

COUNT(*)，MAX，MIN 以外の集約関数の前には，DISTINCT キーワードを付けることができる．このキーワードが付くときは，関数を適用する前に，引数から重複した値が取り除かれる．DISTINCT の他に ALL というキーワードがある．ALL は重複した値もすべて関数に適用する．集約関数の前に DISTINCT か ALL のキーワードが置かれない場合はデフォルトとして ALL とみなされる．

### 4.5.3　条件式

条件式も SQL の重要な概念の一つである．条件式は，テーブル式や行やグループの真偽を評価するために，WHERE 句や，ON 句，HAVING 句などで使用される．以下に条件式の構文を表記する．

条件式
  ::= 条件項
    | 条件式 OR 条件項

```
  条件項
    ::= 条件因子
      | 条件項 AND 条件因子

    条件評価
      ::= 条件一次子 [ IS [ NOT ] { TRUE | FALSE }]

単純条件
  ::= 比較条件
    | BETWEEN 条件
    | LIKE 条件
    | IN 条件
    | MATCH 条件
    | 限定条件
    | 存在条件
    | UNIQUE 条件

  比較条件
    ::= 行構築子 比較演算子 行構築子

  比較演算子
    ::= | < | <= | > | >= | <>

  BETWEEN 条件
    ::= 行構築子 [ NOT ] BETWEEN 行構築子 AND 行構築子

  LIKE 条件
    ::= 文字列式 [ NOT ] LIKE パターン [ ESCAPE エスケープ ]

  IN 条件
    ::= 行構築子 [ NOT ] IN ( テーブル式 )
      | スカラー式 [ NOT ] IN ( スカラー式カンマリスト )

  MATCH 条件
    ::= 行構築子 MATCH [ UNIQUE ] ( テーブル式 )

  限定条件
    ::= 行構築子 MATCH [ UNIQUE ] ( テーブル式 )
```

EXISTS 条件
  ::= EXISTS ( テーブル式 )

UNIQUE 条件
  ::= UNIQUE ( テーブル式 )

条件一次子は，単純条件，または，かっこで囲まれた条件式のことである．以下で条件式の各ステートメントについて簡略に説明する．

- 比較条件
  = , < , <= , > , >= , <> の比較演算子によって条件式を定義する．
- BETWEEN 条件
  BETWEEN 条件は，単なる省略形である．

  ```
  y BETWEEN x AND z
  ```

  この条件は，次の構文と同じである．

  ```
  x <= y AND y <= z
  ```

- LIKE 条件
  LIKE 条件は，文字列の簡単なパターン照合のためのステートメントである．
- IN 条件
  A IN (C,D,...)
  上記の A がリスト内の任意の値と一致する場合，TRUE を返す．
- MATCH 条件
  行構築子の評価結果を R とする．テーブル式によって抽出されたテーブルを T とする．T の中に R が存在するときに TRUE を返す．
- 限定条件
  SOME と ANY は等価な意味をもつ．SOME と ANY は，任意のテーブル式に対して比較条件を満たすならば TRUE を返す．

  ALL は，すべてのテーブル式に対して比較条件を満たすならば TRUE を返す．
- EXIST 条件
  テーブル式によって行が抽出された場合は TRUE を返す．
- UNIQUE 条件
  テーブル式によって同一の行が抽出されない場合は TRUE を返す．

### 4.5.4　データ定義

- CREATE SCHEMA
  CREATE SCHEMA は，スキーマを定義する．構文を以下に示す.

```
CREATE SCHEMA [ スキーマ ] [ AUTHORIZATION ユーザ ]
                        [ DEFAULT CHARACTER SET 文字セット ]
                        [ スキーマ要素のリスト ]
```

- DROP SCHEMA
  DROP SCHEMA は，既存のスキーマを破棄する．構文を以下に示す.

```
DROP SCHEMA スキーマ { RESTRICT | CASCADE }
```

- CREATE TABLE
  CREATE TABLE はベーステーブルを定義する．構文を以下に示す.

  ベーステーブル定義
  ::= CREATE TABLE ベーステーブル
  　　( ベーステーブル要素カンマリスト )

  構文中の “ベーステーブル” は，新しいベーステーブルの名前である．各 “ベーステーブル要素” は，列定義か，あるいはベーステーブルの制約定義となる.

  列定義の構文を次に示す.

  列定義
  ::= 列 { データ型 | ドメイン }
  　　[ デフォルト定義 ]
  　　[ 列制約定義リスト ]

  オプションのデフォルト定義は以下の形式で表す.

```
DEFAULT { 定数 | ニラディック関数 | NULL }
```

- DROP TABLE
  DROP TABLE は，既存のテーブルを破棄する．構文を以下に示す.

```
DROP TABLE ベーステーブル { RESTRICT | CASCADE }
```

- CREATE VIEW
  ビューとは，名前付けされた仮想テーブルのことである．CREATE VIEW は，ビューを定義する．構文を以下に示す．

  ビュー定義
  ```
  ::= CREATE VIEW ビュー [ ( 列カンマリスト ) ]
          AS テーブル式
                  [ WITH [ CASCADED | LOCAL ] CHECK OPTION ]
  ```

  構文中に現れる “ビュー” は，新しいビューの名前である．

- DROP VIEW
  DROP VIEW は，既存のビューを破棄する．構文を以下に示す．

  ```
  DROP VIEW ビュー { RESTRICT | CASCADE }
  ```

- GRANT
  GRANT は，データベースに対するアクセス権を定義する．構文を次に示す．

  ```
  GRANT 特権 ON オブジェクト TO ユーザ [ WITH GRANT OPTION ]
  ```

  GRANT の構文要素の説明は今回は避けることにする．DDL の代表的なステートメントとしての紹介にとどめておく．

### 4.5.5　データ操作

SQL の UNION，EXCEPT，および INTERSECT は，集合論の結び，差分，交差に基づいている．結び，差分，交差のオペランドを表す 2 つのテーブルは同じ次数でなくてはならず，該当する列は互換性のあるデータ型で定義されなければならない．

UNION と EXCEPT は「非結合テーブル式」であり，INTERSECT は,「非結合テーブル項」で使用する．構文を次に示す．

非結合テーブル式
```
::= 非結合テーブル項
   | テーブル式 { UNION | EXCEPT } [ ALL ]
                    [ CORRESPONDING [ BY ( 列カンマリスト) ] ]
                          テーブル項
```

非結合テーブル項

```
::= 非結合テーブル一次子
  | テーブル項 INTERSECT [ ALL ]
                    [ CORRESPONDING [ BY ( 列カンマリスト ) ] ]
                        テーブル一次子
```

- SELECT
  SELECT は，データベースから特定のデータを照会する．次のような形式を取る．

  ```
  SELECT [ ALL | DISTINCT ] 選択項目カンマリスト
  ```

  ALL と DISTINCT のどちらも指定されなかった場合は，ALL とみなされる．

- INSERT
  INSERT は，表に行を追加する．次のような形式を取る．

  ```
  INSERT INTO テーブル [ ( 列カンマリスト ) ] ソース
  ```

- UPDATE
  UPDATE は，行の内容を更新する．次のような形式を取る．

  ```
  UPDATE テーブル SET 代入カンマリスト [ WHERE 条件式 ]
  ```

- DELETE
  行を削除する．次のような形式を取る．

  ```
  DELETE FROM テーブル [ WHERE 条件式 ]
  ```

- FROM 句
  FROM 句は，次のような形式を取る．

  ```
  FROM テーブル参照のカンマリスト
  ```

  FROM 句は，テーブルのカンマリストに記述されているテーブルのデカルト積である新しいテーブルを生成する．

- WHERE 句
  WHERE 句は，次のような形式を取る．

  ```
  WHERE 条件式
  ```

WHERE 句は，条件式を満たさない行は，すべてテーブルより取り除かれる．

- GROUP BY 句
  GROUP BY 句は, 次のような形式を取る．

  ```
  GROUP BY 列カンマリスト
  ```

  列カンマリストで指定された列名の値でグループ化される．

- HAVING 句
  HAVING 句は，次のような形式を取る．

  ```
  HAVING 条件式
  ```

  FROM 句，WHERE 句，GROUP BY 句から生成されたグループテーブルから条件式を満たさなかったすべてのグループを取り除いたテーブルを生成する．

- ORDER BY 句
  ORDER BY 句は，次のような形式を取る．

  ```
  ORDER BY 順番項目のカンマリスト
  ```

  各順番項目は，次の形式を取る．

  ```
  { 列 | 符号なし整数値 } [ ASC | DESC ]
  ```

  ORDER BY 句は，表の指定した項目を昇順または，降順に並べ替える．ASC は昇順を意味し，DESC は降順を意味する．ASC または，DESC を指定しなかった場合，ASC が指定されたものとみなされる．

# 第5章　モデルのテーブル化

## 5.1　テーブル化の方針

モデルを関係データベースとして，その情報を管理するに当たり，テーブルを定義しなければならない．前にも述べたように，本環境では，モデル要素ごとにテーブルを作成する．モデル要素は，メタモデル中のメタクラスを指す．UMLで定義されているモデル要素を以下に示す．なお，モデル要素はパッケージごとに分類して表記している．

- Foundation Package
  - Core Package
    Abstraction, Association, AssociationClass, AssociationEnd, Attribute, BehavioralFeature, Binding, Class, Classifier, Comment, Component, Constraint, DataType, Dependency, Element, ElementOwnership, ElementResidence, Feature, Flow, GeneralizableElement, Generalization, Interface, Method, ModelElement, Namespace, Node, Operation, Tagged Values, Parameter, Permission, PresentationElement, Relationship, StructuralFeature, TemplateParameter, Usage
  - Extension Mechanisms Package
    Constraint, ModelElement, Stereotype, TaggedValue
- Behavioral Elements Package
  - Common Behavior Package
    Action, ActionSequence, Argument, AttributeLink, CallAction, ComponentInstance, CreateAction, DestroyAction, DataValue, Exception, Instance, Link, LinkEnd, LinkObject, NodeInstance, Object, Reception, ReturnAction, SendAction, Signal, Stimulus, TerminateAction, UninterpretedAction
  - Collaborations Package
    AssociationEndRole, AssociationRole, ClassifierRole, Collaboration, Interaction, Message

  - Use Cases Package
    Actor，Extend，ExtensionPoint，Include，UseCase，UseCaseInstance
  - State Machines Package
    CallEvent，ChangeEvent，CompositeState，Event，FinalState，Guard，SignalEvent，SimpleState，State，StateMachine，StateVertex，StubState，SubmachineState，SynchState，TimeEvent，Transition
  - Activity Graphs Package
    ActivityGraph，ActionState，CallState，ObjectFlowState，Partition，SubactivityState

- General Mechanisms Package

  - Model Management Package
    Model，Package，Subsystem

テーブル属性は，モデル要素のプロパティとする．モデル要素のプロパティは，メタクラスの属性集合とナビゲーション・プロパティ集合の和とする．メタクラスの属性については，OMG Unified Modeling Language Specification Specification [1] の UML Semanticsで定められている．

上記のような方針でテーブルを作成すると，テーブルの各構成要素とモデル要素との対応は以下のようになる．

| | | |
|---|---|---|
| テーブル名 | → | モデル要素名 |
| 属性名 | → | モデル要素のプロパティ名 |
| 属性 | → | 特定のプロパティの値の集合 |
| 組 | → | 特定のモデル要素のインスタンスに関する情報の集合 |

モデル要素のプロパティは，メタモデルの汎化関係によって継承されるものとする．

前に，すべてのメタクラスに対してテーブルを作成すると述べたが，抽象メタクラスはインスタンスを持たないのでテーブルを作成することは，無意味であるように思えるかもしれない．しかし，抽象メタクラスに対して記述した制約の有効範囲は，そのメタクラスから継承しているすべての子クラスに及ぶ．そこで，抽象メタクラスのテーブルは，その子の具象メタクラスから抽象メタクラスが所有するテーブル属性のみを射影し，子のメタクラスのすべての組の和演算を行ない作成する．つまり，抽象メタクラスのテーブルは実体としては存在せず，テーブ

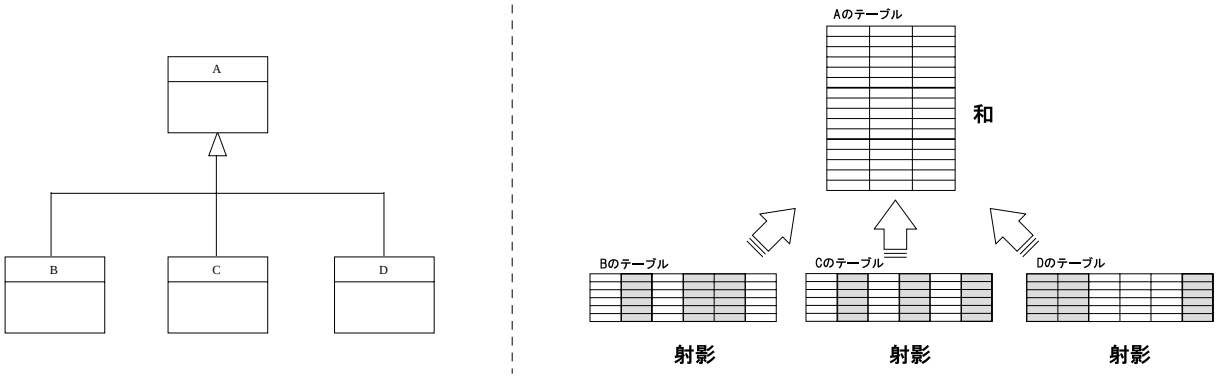


図 5.1: 抽象メタクラスのテーブル生成

ルの実体を持つ具象メタクラスの集合から生み出された仮想的なビューテーブルとして存在する．図 5.1 に抽象メタクラスのテーブル生成のイメージを示す．

上で挙げたモデル要素のうち，抽象メタクラスを次に示す．以下のリストもパッケージごとに分類している．

- Foundation Package
  - Core Package
    BehavioralFeature, Classifier, Element, Feature, GeneralizableElement, ModelElement, Namespace, StructuralFeature
  - Extension Mechanisms Package
    ModelElement
- Behavioral Elements Package
  - Common Behavior Package
    Action, Instance
  - State Machines Package
    State, StateVertex
- General Mechanisms Package
  - Model Management Package
    (Non Abstract Metaclass)

上記の抽象メタクラス以外のクラスは具象メタクラスである．具象クラスにもプロパティを継承する子メタクラスを持つメタクラスが存在する．これについても，プロパティを継承している子のメタクラスから親メタクラスから継承したテー

ブル属性のみを射影し，子のメタクラスのすべての組の和を親メタクラス自身のテーブルとの和としてテーブルを生成する．

ここで，テーブル属性をメタモデルから抽出する例を示す．例として，以下では，モデル要素の “Operation” についてテーブルの属性を抽出する．

例を扱うにあたり，ここで UML のすべてのメタモデルを示す．この例ではすべてのメタモデルを必要するわけではないが，後で示すすべてのモデル要素についてのテーブル属性もこれらのメタモデルに基づいて抽出されるのでここで示しておく．図 5.2 は Core Pakage - Backbone，図 5.3 は Core Package - Relationships，図 5.4 は Core Package - Dependencies，図 5.5 は Core Package - Classifiers，図 5.6 は Core Package - Auxiliary elements，図 5.7 は Extension Mechanisums，図 5.8 は Common Behavior - Signals，図 5.9 は Common Behavior - Actions，図 5.10 は Common Behavior - Instances and Links，図 5.11 は Collaborations，図 5.12 は Use Cases，図 5.13 は State Machine - Main，図 5.14 は State Machine - Events，図 5.15 は Activity Graphs，図 5.16 は Model Management のメタモデルである．

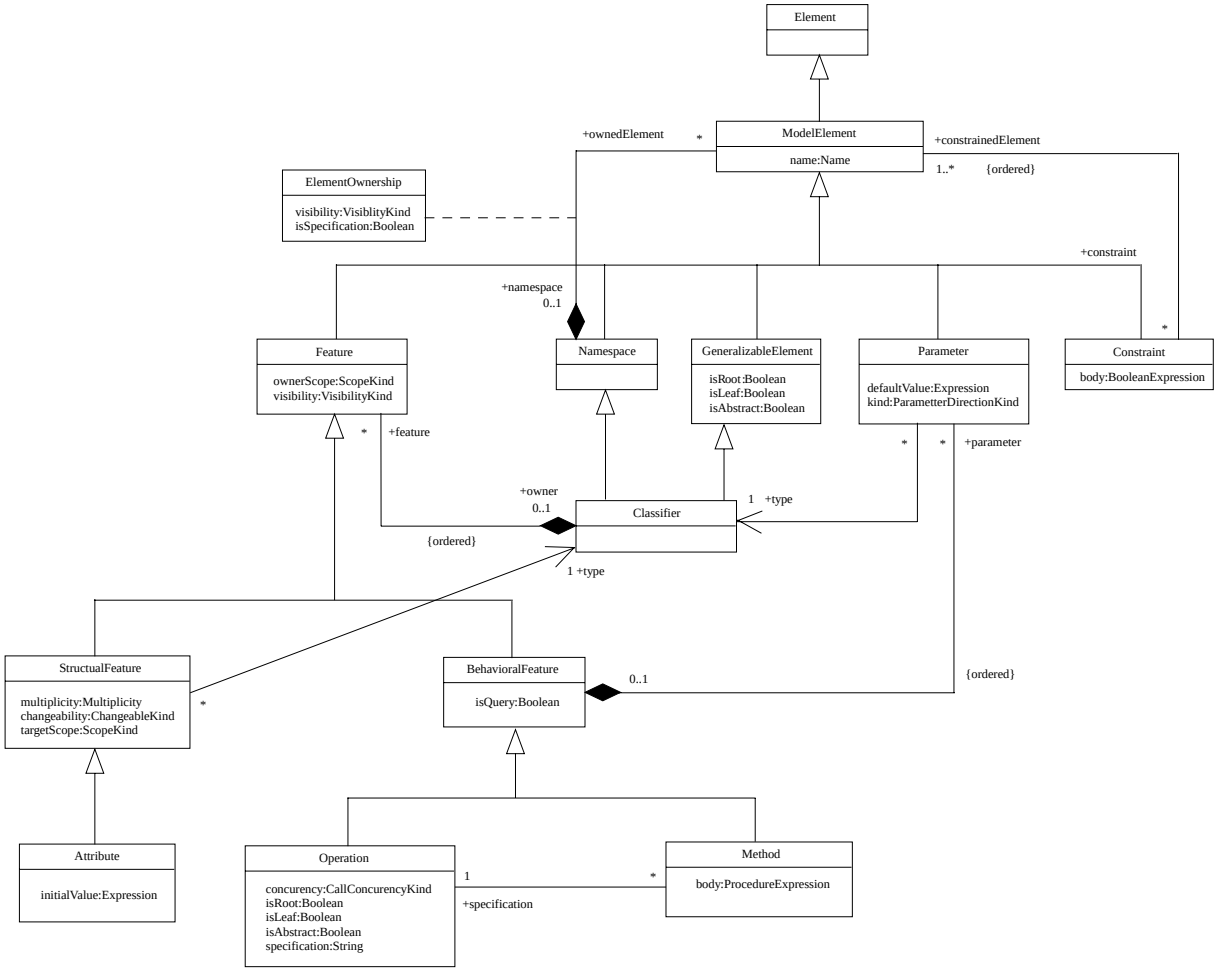


図 5.2: Core Package - Backbone

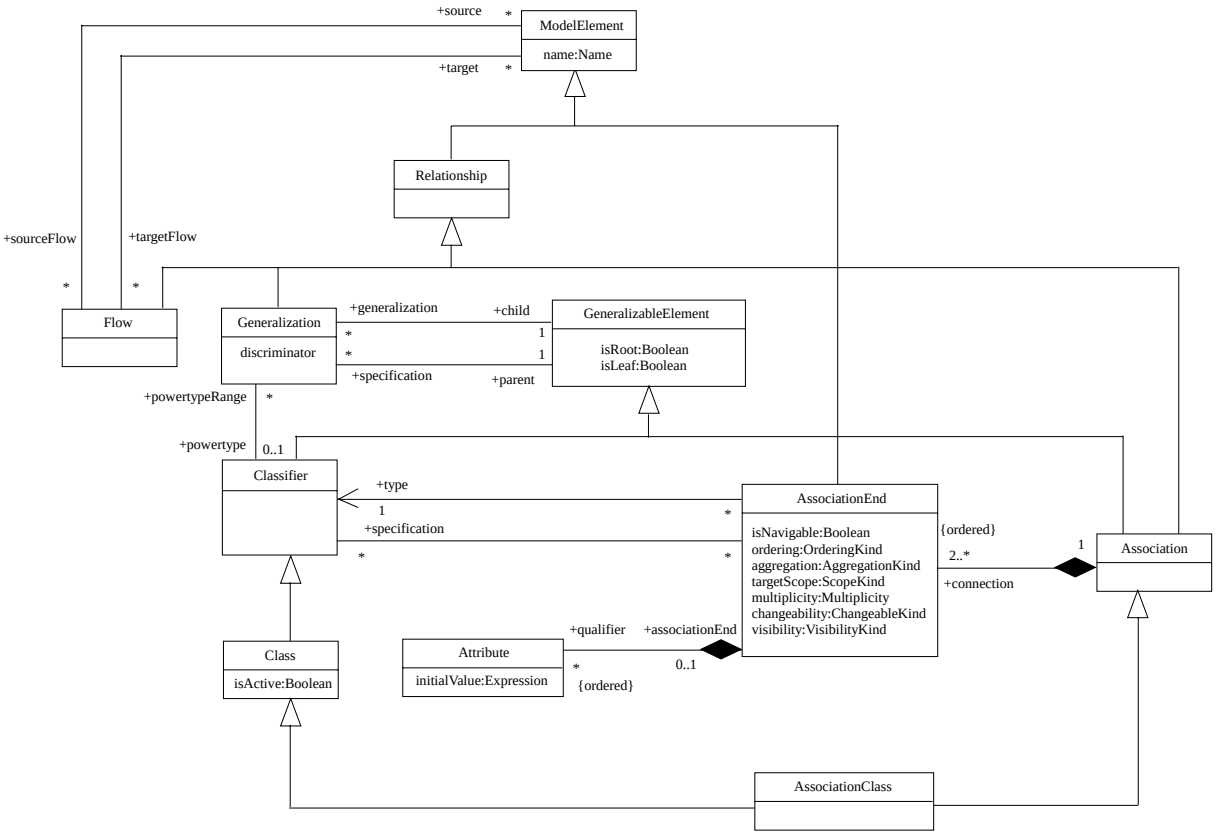


図 5.3: Core Package - Relationships

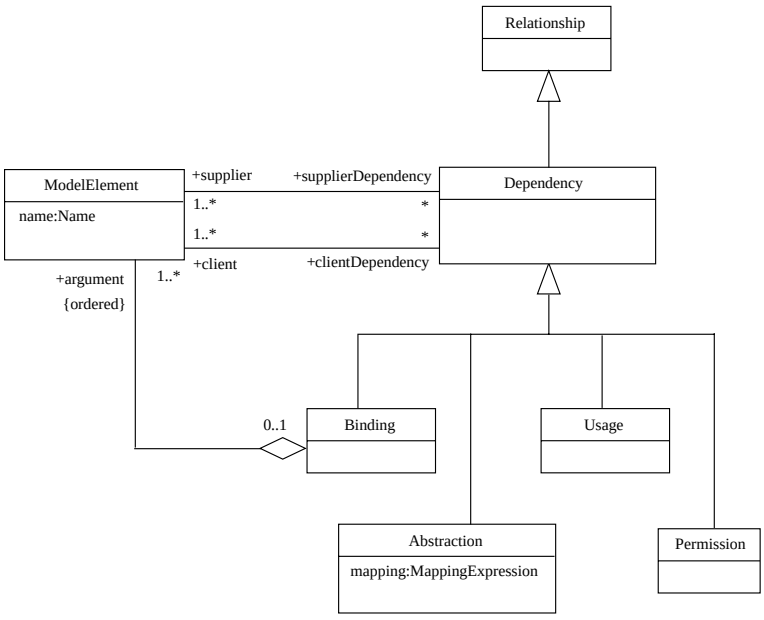


図 5.4: Core Package - Dependencies

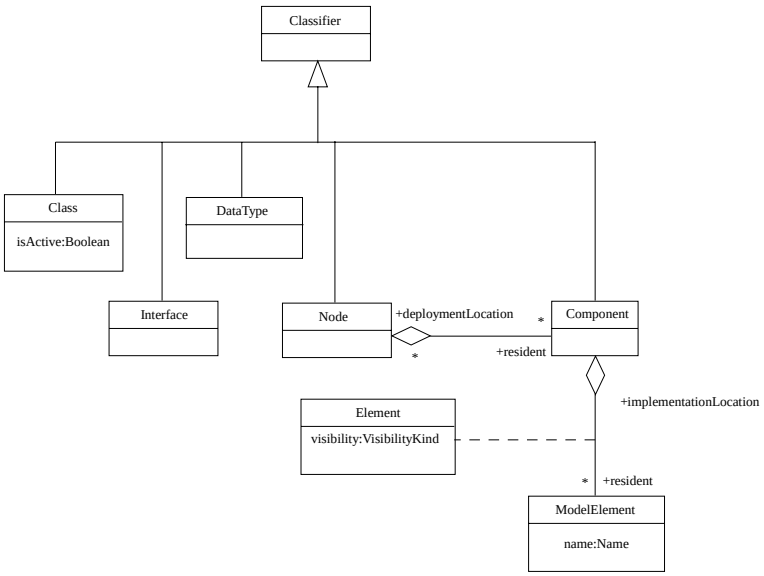


図 5.5: Core Package - Classifiers

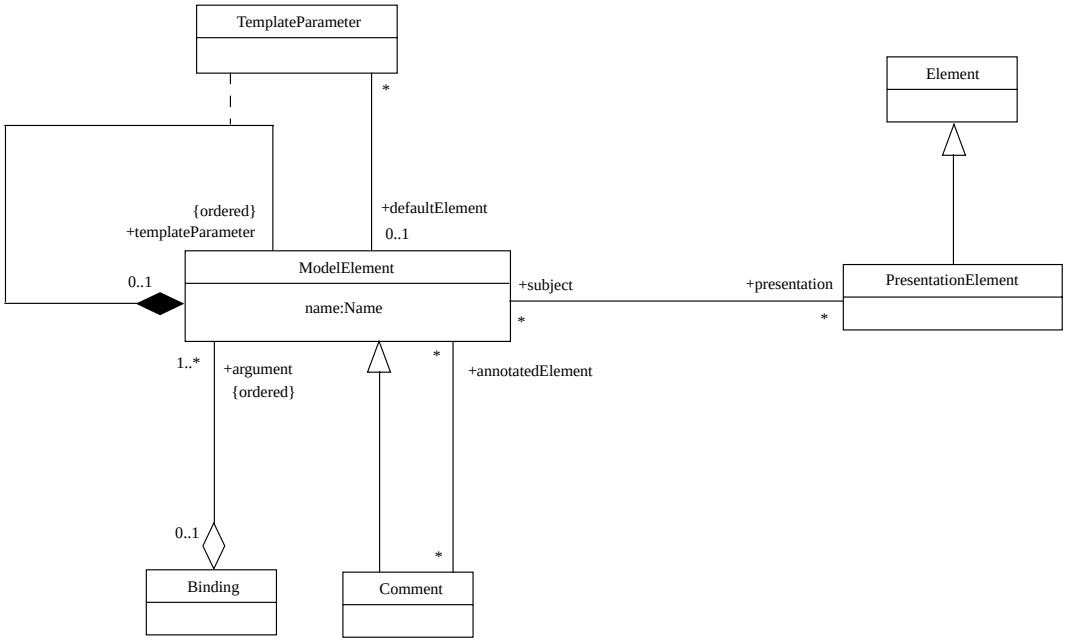


図 5.6: Core Package - Auxiliary elements

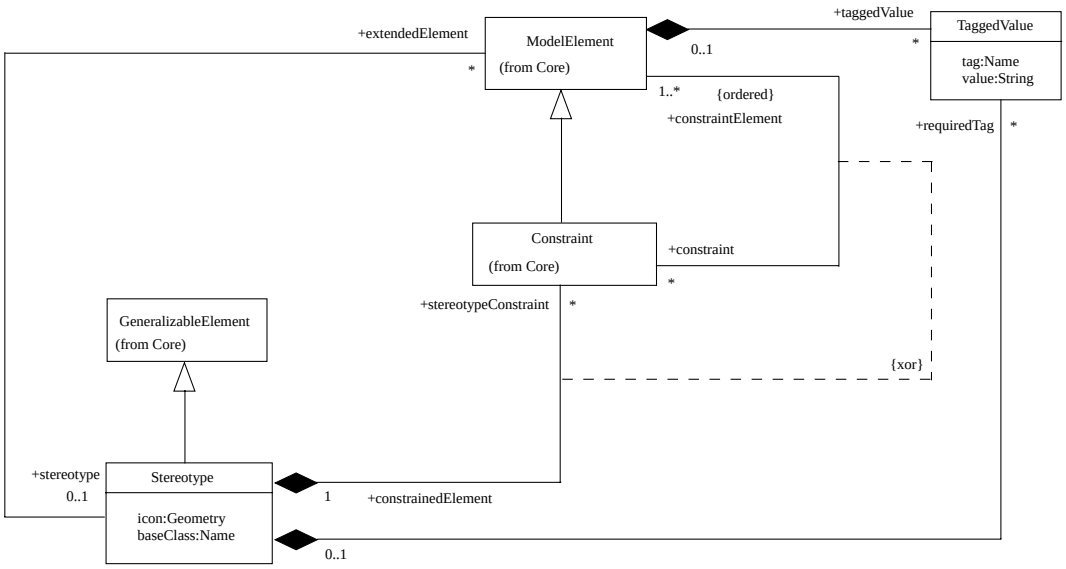


図 5.7: Extension Mechanisms

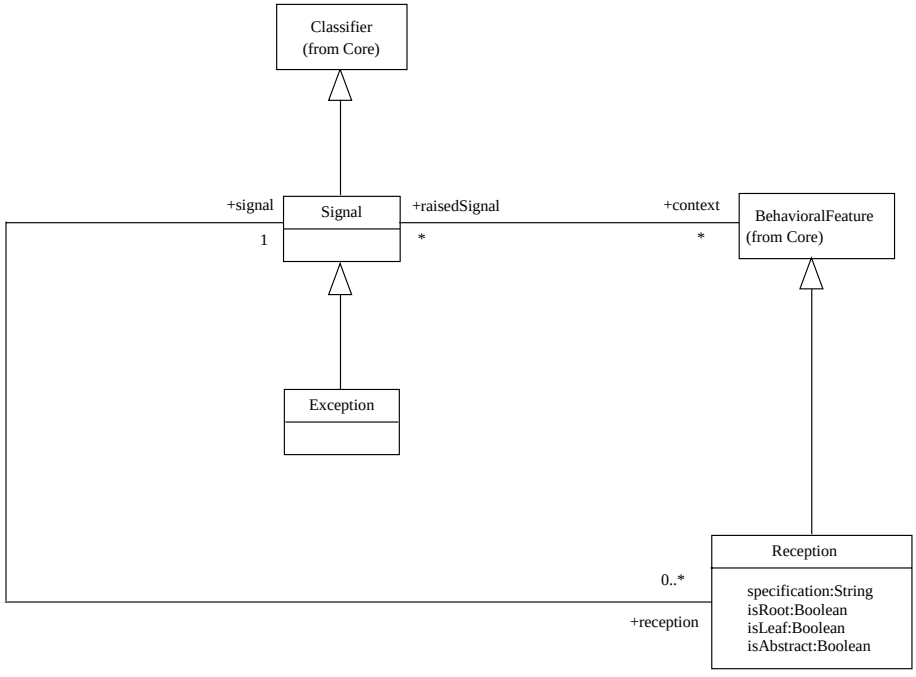


図 5.8: Common Behavior - Signals

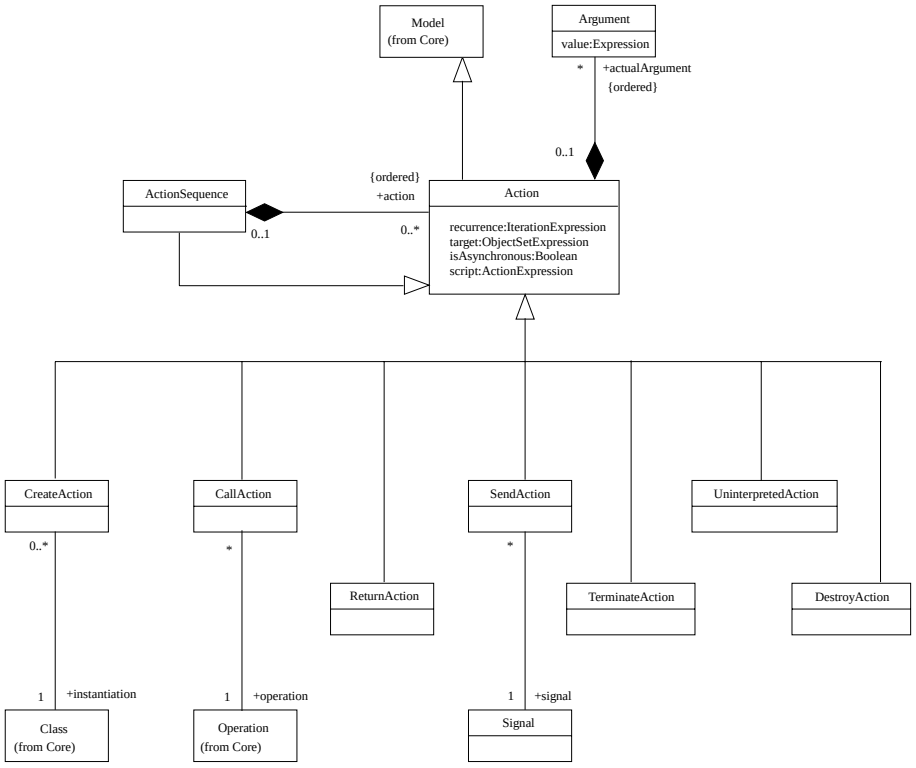


図 5.9: Common Behavior - Actions

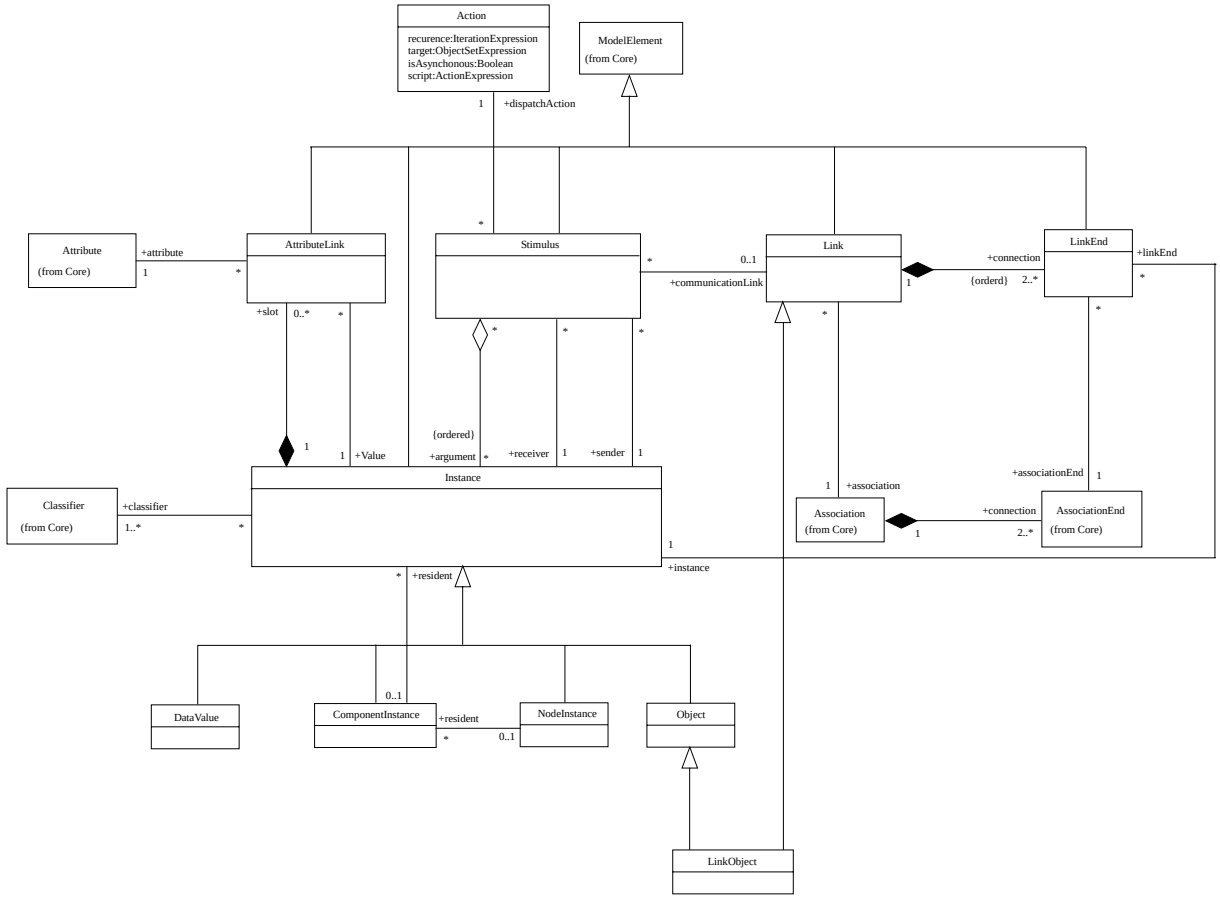


図 5.10: Common Behavior - Instances and Links

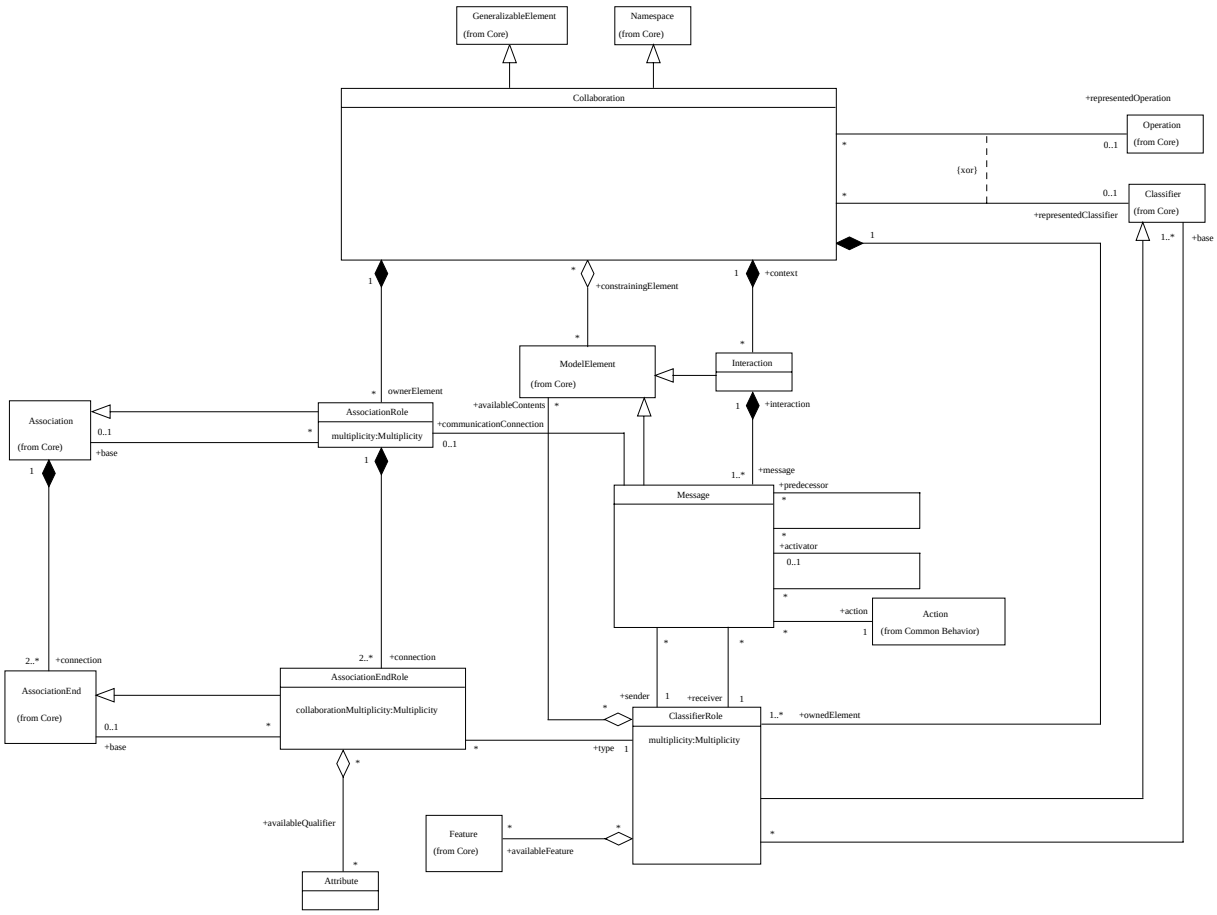


図 5.11: Collaborations

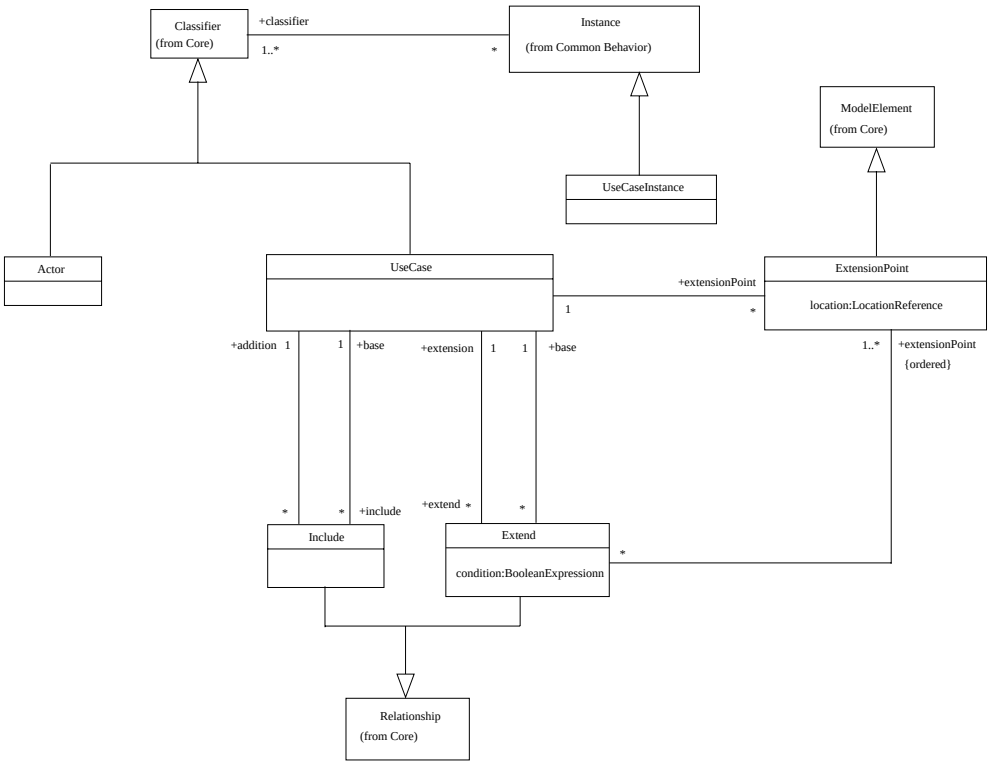


図 5.12: Use Cases

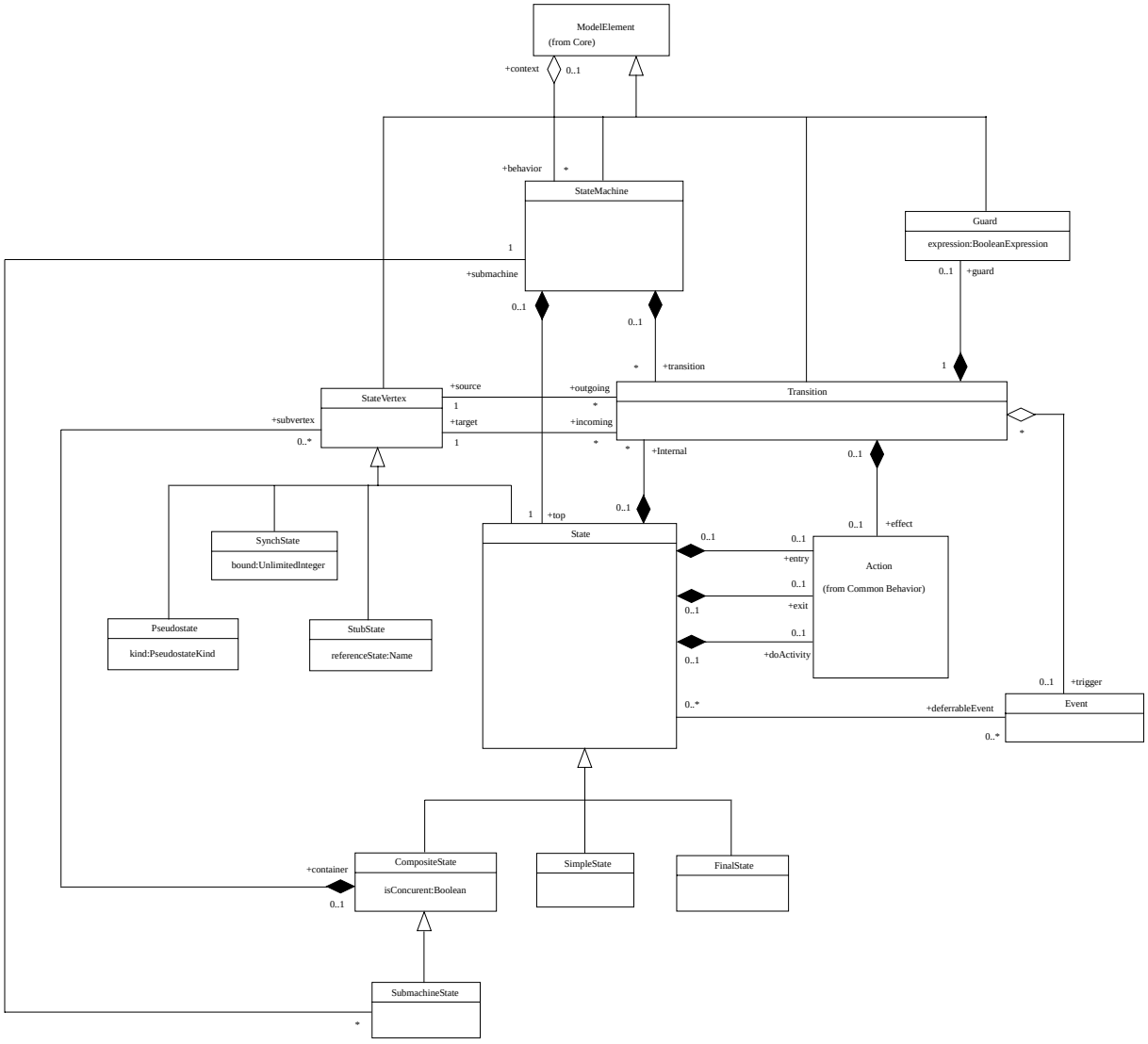


図 5.13: State Machines - Main

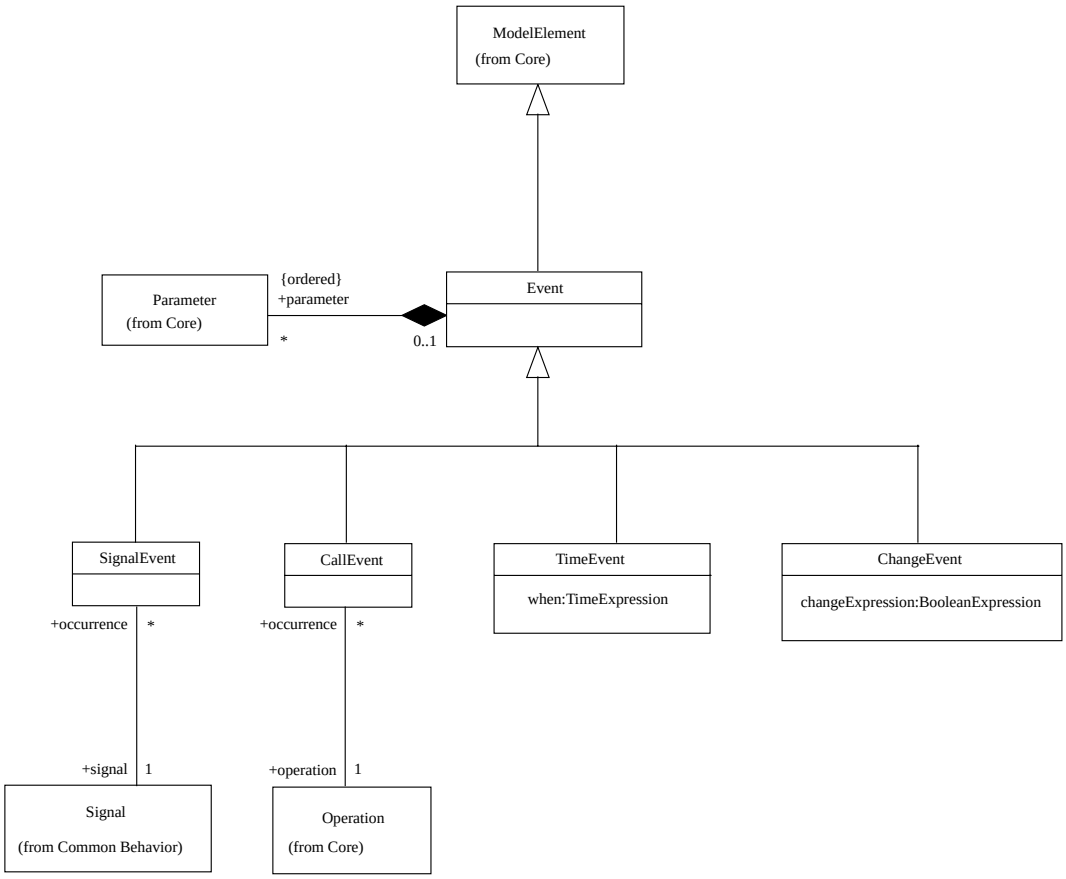


図 5.14: State Machines - Events

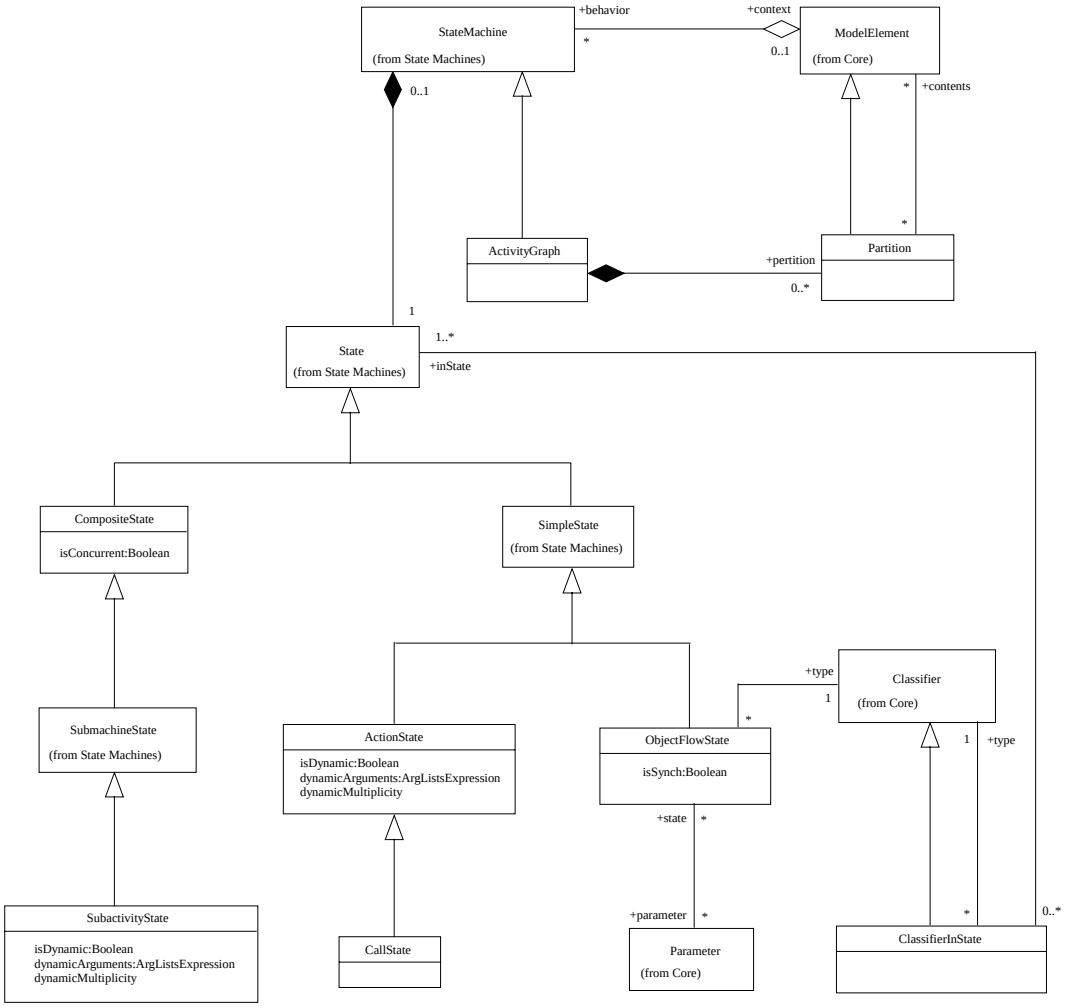


図 5.15: Activity Graphs

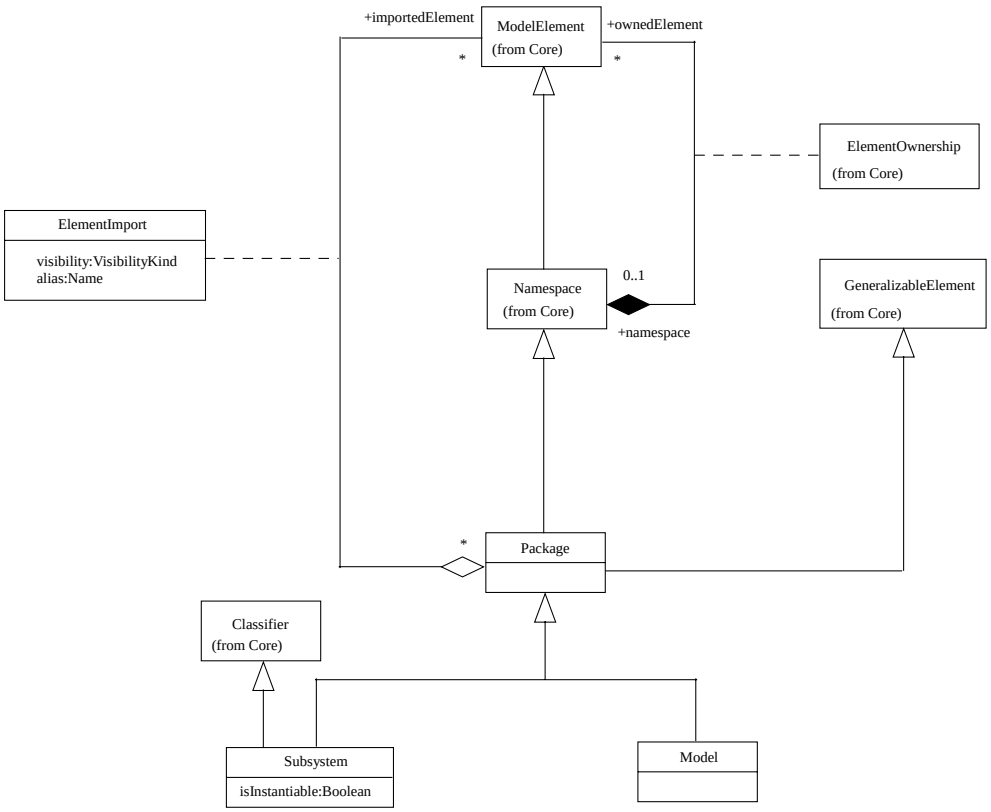


図 5.16: Model Management

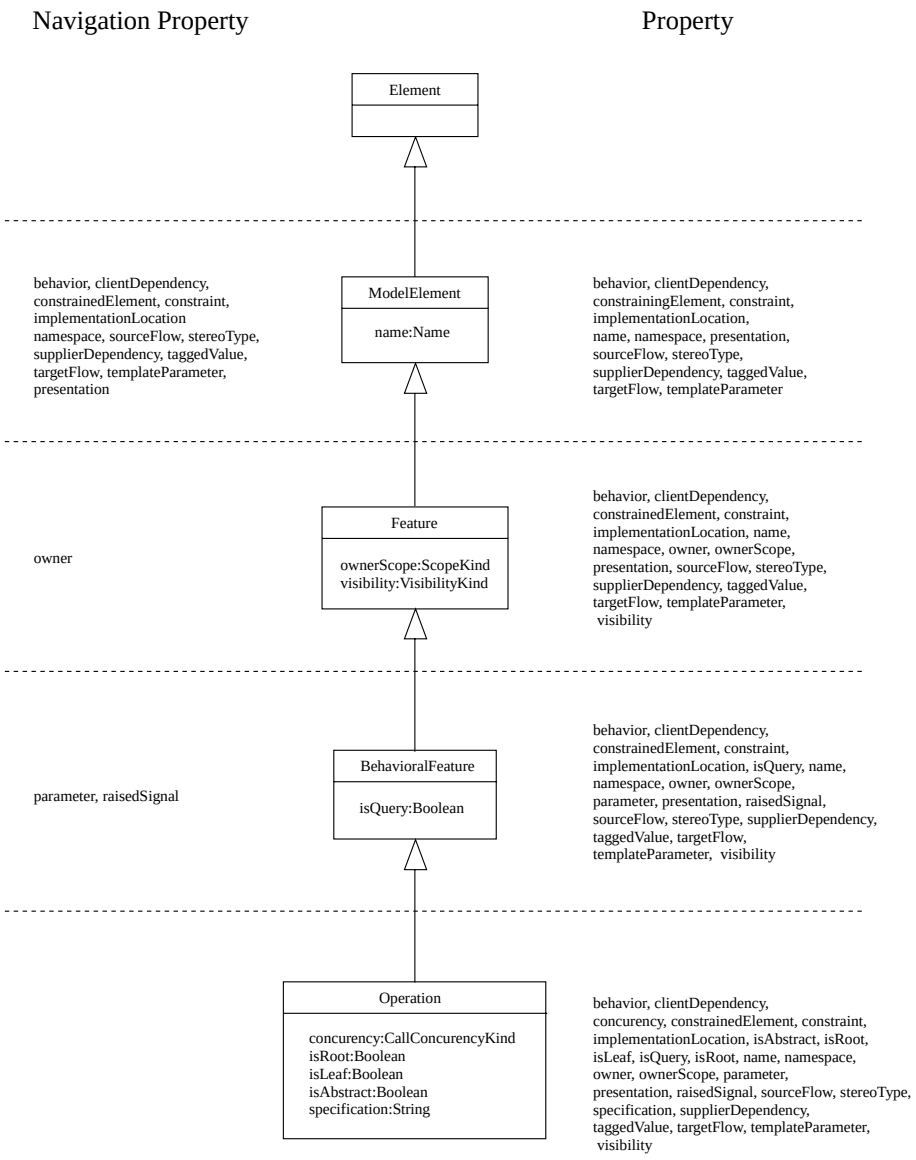


図 5.17: “Operation” までのプロパティの継承

“Operation” は，図 5.2 に示されるメタモデルの Core Package - Backbone より，“Element”，“ModelElement”，“Feature”，“BehavioralFeature” と汎化関係にあり，プロパティを継承していることがわかる．

親メタクラスから順にプロパティを抽出すると，“Element” は，属性および，ナビゲーション・プロパティは持っていない．“ModelElement” は，属性として，name を持っており，図 5.2 の Core Package - Backbone，図**??**の Core Package - Relationship，図 5.4 の Core Package - Dependencies，図 5.5 の Core Package - Classifiers，図 5.6Core Package - Auxiliary elements，図 5.11 の Collaborations，図 5.13 の State Machines - Main，図 5.15 の Activity Graphs，図 5.16 の Model Management よりナ

ビゲーション・プロパティは，behavior，clientDependency，constrainingElement，constraint，implementationLocation，namespace，sourceFlow，stereoType，supplierDependency，taggedValue，targetFlow，templateParameter，presentation を持っている．“Feature” の属性として，ownerScope と visibility を持っており，図 5.2 の Core Pacage - Backbone よりナビゲーション・プロパティとして，owner を持っている．“BehavioralFeature” の属性は，isQuery である．図 5.2 の Core Pacage - Backbone と図 5.8 の Common Behavior - Signals よりナビゲーション・プロパティは，parameter と raisedSignal である．そして，“Operation” の属性は，concurendy，isRoot，isLeaf，isAbstract，specification である．ナビゲート可能なモデル要素はない．

“Operation” は，これらすべてのプロパティを継承する．よって，“Operation” のテーブルの属性名は，behavior，clientDependency，concurency，constrainingElement，constraint，implementationLocation，isAbstract，isRoot，isLeaf，isQuery，isRoot，name，namespace，owner，ownerScopename，parameter，raisedSignal，sourceFlow，stereoType，specification，supplierDependency，taggedValue，targetFlow，templateParameter，presentation，visibility となる．

図 5.17 に “Operation” までのプロパティの継承のモデルを示す．

次節の「テーブルの属性名」では，同様の手法ですべてのモデル要素についてテーブルの属性名を抽出する．

## 5.2　テーブルの属性名

前節で示した方法で，各モデル要素のテーブル属性を抽出したものを以下に示す．各モデル要素は，パッケージごとに分類されており，各項目の構成は，“*” の直後の頭文字が大文字のタームは，モデル要素名を表している．その次行からテーブル属性のリストを “, ” で区切って表記している．各モデル要素とパラメータの意味は，OMG Unified Modeling Language Specification Specification [1] の UML Semantics に記述されている．

- Foundation Package
  - Core Package
    - Abstraction
      behavior, clientDependency, client, constrainedElement, constraint, implementationLocation, mapping, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, supplier, taggedValue, targetFlow, templateParameter, visibility
    - Association
      behavior, clientDependency, connection, constrainedElement, con-

straint, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* AssociationClass
  behavior, clientDependency, connection, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isActive, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* AssociationEnd
  aggregation, behavior, changeability, clientDependency, constrainedElement, constraint, implementationLocation, isNavigable, multiplicity, namespace, name, ordering, presentation, qualifier, sourceFlow, specification, stereoType, supplierDependency, taggedValue, targetFlow, targetScope, templateParameter, type, visibility
* Attribute
  associationEnd, behavior, changeability, clientDependency, constrainedElements, constraint, implementationLocation, initialValue, multiplicity, namespace, name, ownerScope, owner, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, targetScope, templateParameter, type, visibility
* BehavioralFeature
  behavior, clientDependency, containingElement, constraint, implementationLocation, isQuery, namespace, name, ownerScope, owner, parameter, presentation, raisedSignal, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Binding
  argument, behavior, clientDependency, client, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, supplier, taggedValue, targetFlow, templateParameter, visibility
* Class
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isActive, isLeaf, isRoot, namespace, name, ownedElement, partici-

pant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* Classifier
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Comment
  annotatedElement, behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Component
  behavior, clientDependency, constrainedElement, constraint, deploymentLoction, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, resident, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Constraint
  behavior, body, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* DataType
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Dependency
  behavior, clientDependency, client, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, supplier, taggedValue, tar-

getFlow, templateParameter, visibility

* Element
  (No Property)
* ElementOwnership
  isSpecification, visibility
* ElementResidence
  visibility
* Feature
  behavior clientDependency, constrainedElements, constraint, implementationLocation, namespace, name, ownerScope, owner, presentation, sourceFlow, stereoType, suplierDependency, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Flow
  behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* GeneralizableElement
  behavior clientDependency, constraint, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, presentation, specialization, supplierDependency, templateParameter
* Gneralization
  behavior, child, clientDependency, constrainedElement, constraint, discriminator, implementationLocation, namespace, name, parent, powertype, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Interface
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Method
  behavior, body, clientDependency, constraint, containingElement,

implementationLocation, isQuery, namespace, name, ownerScope, owner, parameter, presentation, raisedSignal, sourceFlow, specification, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* ModelElement
  behavior clientDependency, constrainedElement constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
* Namespace
  behavior clientDependency, constrainedElement constraint, implementationLocation, namespace, name, ownedElement, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Node
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, resident, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Operation
  behavior clientDependency, concurrency, constrainedElement, constraint, implementationLocation, isAbstract, isLeaf, isQuery, isRoot, namespace, name, ownerScope, owner, parameter, presentation, raisedSignal, sourceFlow, stereoType, specification, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Parameter
  behavior, clientDependency, constrainedElement, constraint, defaultValue, implementationLocation, kind, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, type visibility
* Permission
  behavior, clientDependency, client, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, supplier, taggedValue, targetFlow, templateParameter, visibility

        * PresentationElement
          behavior clientDependency, constraint, implementationLocation, namespace, name, presentation, supplierDependency, templateParameter
        * Relationship
          behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
        * StructuralFeature
          behavior, clientDependency, constrainedElements, constraint, implementationLocation, namespace, name, ownerScope, owner, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
        * TemplateParameter
          defaultElement
        * Usage
          behavior, clientDependency, client, constrainedElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, supplier, taggedValue, targetFlow, templateParameter, visibility
    - Extension Mechanisms
        * Constraint
          behavior body, clientDependency, constrainedElement constraint, implementationLocation, namespace, name, presentation, supplierDependency, templateParameter
        * Stereotype
          baseClass, behavior, clientDependency, constraint, extendedElement, generalization, icon, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, presentation, requiredTag, specialization, stereotypeConstraint, supplierDependency, templateParameter
        * TaggedValue
          modelElement, stereotype, tag, value

- Behavioral Elements Package
    - Common Behavior Package

* Action
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* ActionSequence
  action, actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* Argument
  value,
* AttributeLink
  attribute, behavior, clientDependency, constrainingElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, value visibility
* CallAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, operation, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* ComponentInstance
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, resident, slot, supplierDependency, templateParameter,
* CreateAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, instantiation, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, tagged-

Value, targetFlow, target, templateParameter, visibility

* DestroyAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* DataValue
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, slot, supplierDependency, templateParameter,
* Exception
  behavior, clientDependency, constrainedElement, constraint, context, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, reception, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Instance
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, slot, supplierDependency, templateParameter,
* Link
  association, behavior, clientDependency, connectio, constrainingElement, constraint, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
* LinkEnd
  associationEnd, behavior, clientDependency, constrainingElement, constraint, implementationLocation, instance, namespace, name, presentation, qualifierValue, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
* LinkObject
  association, behavior, classifier, clientDependency, connectio, constrainingElement, constraint, implementationLocation, linkEnd, namespace, name, presentation, slot, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter,

visibility

* NodeInstance
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, resident, slot, supplierDependency, templateParameter,
* Object
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, slot, supplierDependency, templateParameter,
* Reception
  behavior, clientDependency, constraint, implementationLocation, isAbstract, isLeaf, isQuery, isRoot, namespace, name, ownerScope, owner, parameter, presentation, signal, specification, supplierDependency, templateParameter, visibility
* ReturnAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* SendAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, signal, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* Signal
  behavior, clientDependency, constrainedElement, constraint, context, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, reception, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
* Stimulus
  argument, behavior, clientDependency, communicationLink, constrainingElement, constraint, dispatchAction, implementationLo-

cation, namespace, name, presentation, receiver, sender, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility

* TerminateAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility
* UninterpretedAction
  actualArgument, behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isAsynchronous, isLeaf, isRoot, namespace, name, ownedElement, presentation, recurrence, script, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, visibility

– Collaborations Package

* AssociationEndRole
  aggregation, availableQualifier, base, behavior, changeability, clientDependency, collaborationMultiplicity, connection, constrainedElement, constraint, generalization, implementationLocation, isAbstract, isLeaf, isNavigable, isRoot, multiplicity, namespace, name, ordering, presentation, qualifier, sourceFlow, specialization, specification, stereoType, supplierDependency, taggedValue, targetFlow, targetScope, templateParameter, type, visibility
* AssociationRole
  base, behavior, clientDependency, connection, constraint, generalization, implementationLocation, isAbstract, isLeaf, isRoot, multiplicity, namespace, name, presentation, specialization, supplierDependency, templateParameter
* ClassifierRole
  availableContents, availableFeature, base, behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, multiplicity, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

    - Collaboration
      behavior, clientDependency, constrainedElement, constraint, generalization, implementationLocation, interaction, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, presentation, representedClassifier, representedOperation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
    - Interaction
      behavior, clientDependency, constrainedElement, constraint, context, implementationLocation, message namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
    - Message
      action, activator, behavior, clientDependency, communicationConnection, constrainedElement constraint, implementationLocation, interaction, namespace, name, predecessor, presentation, receiver, sender, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
- Use Cases Package
    - Actor
      behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
    - Extend
      base, behavior, clientDependency, condition, constraint, extensionPoint, extension, implementationLocation, namespace, name, presentation, supplierDependency, templateParameter
    - ExtensionPoint
      behavior, clientDependency, constrainedElement, constraint, implementationLocation, location, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
    - Include
      addition, base, behavior, clientDependency, constrainedElement, constraint, constraint, implementationLocation, namespace, name,

presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility

* UseCase
  behavior, clientDependency, constrainedElement, constraint, extend, extensionPoint, feature, generalization, implementationLocation, include, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter visibility
* UseCaseInstance
  behavior, classifier, clientDependency, constraint, implementationLocation, linkEnd, namespace, name, presentation, slot, supplierDependency, templateParameter,

– State Machines Package

* CallEvent
  behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, operation, parameter, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* ChangeEvent
  behavior, changeExpression, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, parameter, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* CompositeState
  behavior, clientDependency, constrainedElement, constraint, container, container, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, isConcurrent, isRegion, namespace, name, outgoing, presentation, sourceFlow, stereoType, subvertex, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Event
  behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, parameter, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* FinalState
  behavior, clientDependency, constrainedElement, constraint, con-

tainer, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* Guard
  behavior, clientDependency, constrainedElement, constraint, expression, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* PseudoState
  behavior, clientDependency, constrainedElement, constraint, container, implementationLocation, incoming, kind, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* SignalEvent
  behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, parameter, presentation, signal, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* SimpleState
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* State
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* StateMachine
  behavior, clientDependency, constrainedElement, constraint, context, implementationLocation, namespace, name, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, top, transition visibility
* StateVertex
  behavior, clientDependency, constrainedElement, constraint, con-

tainer, implementationLocation, incoming, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* StubState
  behavior, clientDependency, constrainedElement, constraint, container, implementationLocation, incoming, namespace, name, outgoing, presentation, referenceState, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* SubmachineState
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, isConcurrent, isRegion, namespace, name, outgoing, presentation, sourceFlow, stereoType, submachine, subvertex, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* SynchState
  behavior, bound, clientDependency, constrainedElement, constraint, container, implementationLocation, incoming, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* TimeEvent
  behavior, clientDependency, constrainedElement, constraint, implementationLocation, namespace, name, parameter, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility when
* Transition
  behavior, clientDependency, constrainedElement, constraint, effect, guard, implementationLocation, namespace, name, presentation, sourceFlow, source, stereoType, supplierDependency, taggedValue, targetFlow, target, templateParameter, trigger visibility

– Activity Graphs Package

* ActivityGraph
  behavior, clientDependency, constrainedElement, constraint, context, implementationLocation, namespace, name, partition, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, top, transition visibility
* ActionState

behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, dynamicArguments, dynamicMultiplicity, entry, exit, implementationLocation, incoming, internalTransition, isDynamic, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

* CallState
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, dynamicArguments, dynamicMultiplicity, entry, exit, implementationLocation, incoming, internalTransition, isDynamic, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* ClassifierInState
  behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, inState, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, type visibility
* ObjectFlowState
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, entry, exit, implementationLocation, incoming, internalTransition, isSynch, namespace, name, outgoing, presentation, sourceFlow, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* Partition
  behavior, clientDependency, constrainedElement, constraint, contents, implementationLocation, namespace, name, presentation, sourceFlow, stereotype, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
* SubactivityState
  behavior, clientDependency, constrainedElement, constraint, container, deferrableEvent, doActivity, dynamicArguments, dynamicMultiplicity, entry, exit, implementationLocation, incoming, internalTransition, isConcurrent, isDynamic, isRegion, namespace, name, outgoing, presentation, sourceFlow, stereoType, submachine, subvertex, supplierDependency, taggedValue, targetFlow,

templateParameter, visibility

- General Mechanisms Package
  - Model Management Package
    - Model
      behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
    - Package
      behavior, clientDependency, constrainedElement constraint, generalization, implementationLocation, importedElement, isAbstract, isLeaf, isRoot, namespace, name, ownedElement, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility
    - Subsystem
      behavior, clientDependency, constrainedElement, constraint, feature, generalization, implementationLocation, importedElement, isAbstract, isInstantiable, isLeaf, isRoot, namespace, name, ownedElement, participant, powertypeRange, presentation, sourceFlow, specialization, stereoType, supplierDependency, taggedValue, targetFlow, templateParameter, visibility

# 第6章　OCLからSQLへの対応

前にも述べたように，OCLで記述された制約は，SQLに変換される際，図6.1のように，整合性を判定するためのSQLクエリーと不整合箇所を特定するためのSQLクエリーの2種類のSQLクエリーが生成される．OCLで記述する整合条件は，制約を満たしているかどうかの真偽値を返すのみであるが，モデリングを行なうユーザにとって，最も欲しい不整合に関する情報は，モデルのどこに不整合が生じているのかという情報である．生成されたSQLクエリーは，まず，整合性を判定するためのSQLクエリーから発行され，その結果の意味が “False” であれば，不整合箇所を特定するためのSQLクエリーを発行し，不整合箇所を抽出する．本章では，OCLからSQLへの変換の際の，その変換方法，および，意味の対応について述べる．

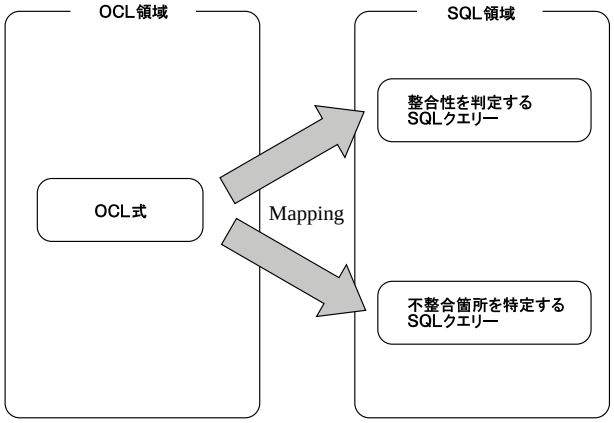


図 6.1: OCL式からSQLクエリーへのマッピング

## 6.1　整合性判定クエリー

不変条件として定義されたOCL式が返す値の意味は，“True” か “False” のどちらかである．UMLに制約として記述したOCL式は，UMLメタモデルに対する不変条件であることは前にも述べた．

モデルが整合している状態とは，定められたすべてのOCL式の戻り値の意味が “True” であることである．OCLからSQLへマッピングした後も，SQLクエリー

のトップレベルの Expression は，制約を満たしていれば “True” という意味を返し，制約を満たしていなければ “False” の意味を返すものとする．

しかし，SQL の仕様では，トップレベルで “True”，もしくは，“False” を SQL クエリーの戻り値として返すようにはなっておらず，SQL クエリーはすべてテーブルとして結果を返す．そこで，OCL の “True” の意味と “False” の意味を何らかのテーブルに意味を対応づけなくてはいけない．

本環境では，単一の組と属性を持つテーブルに “True” という値を持っていれば，“True” という意味を持っているものとし，“False” という値を持っていれば，“False” という意味を持つものと定める．

これは，以下のような SQL クエリーで実現できる．

```
select [ Condition Expression ] as Boolean
```

select の後の “[ Condition Expression ]” には，“True”，もしくは，“False” を戻り値とするような SQL 式を記述する．“[ Condition Expression ]” には，部分的にはサブクエリーを利用してもサブクエリーそのものをここに適用することはできない．サブクエリーもあくまでクエリーであるため，値ではなくテーブルを返すものである．“as Boolean” とすることで真偽値が入る属性のテーブル属性名を “Boolean” とすることができる．ここでは，テーブル属性名の未定義を避けるためにある特定の名前付けを行なった．

上記の SQL クエリーは，図 6.2 のようなテーブルを返す．

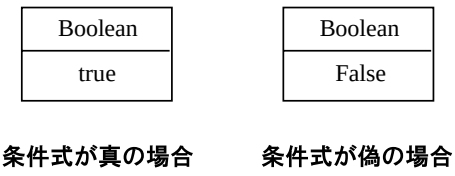


図 6.2: 真偽を意味するテーブル

さらに，“[ inconsistent element output subquery ]” が，不整合箇所を抽出するためのサブクエリーであるとする．“[ inconsistent element output subquery ]” は，モデルが制約条件を満たしているならば，組を持たないテーブルを返す．モデルが制約条件を満たしていないならば，その箇所を組として持つテーブルを返す．よって，テーブルの組の有無を調べることで整合，不整合の判定ができる．“[ inconsistent element output subquery ]” が制約条件を満たしているならば，“True” のテーブルを返し，制約条件を満たしていないならば，“False” のテーブル

を返す SQL クエリーは以下のクエリーで実現できる．

```
select not exists [ inconsistent element output subquery ]
   as Boolean
```

## 6.2　不整合箇所抽出クエリー

次に，不整合箇所を特定するための SQL クエリーについて説明する．不整合箇所を特定するための SQL クエリーは，前節の整合性判定クエリーのサブクエリーである，“[ inconsistency element output subquery ]” の部分をメインのクエリーとしたものである．よって，クエリーを発行する順序としては，整合性判定クエリーが先であるが，OCL から SQL へコンバートされる際には，不整合箇所抽出クエリーを先に生成しなくてはならない．

不整合箇所を特定するための SQL クエリーは，以下のような構文で実現される．

```
select [ Table Name ].name from [ Table Name ]
   where [ Table Name ].name
      not in [ consistent element output subquery ]
```

“[ Table Name ]” には，テーブル名，すなわち特定のモデル要素名を当てはめる．OCL で不変条件を記述する場合，以下のような形式で記述する．

```
context [ Model Element ] inv:
   [ OCL Expression ]
```

“[ Model Element ]” には，OCL の型名，すなわちその OCL 式のコンテキストとしてモデル要素名を指定する．SQL クエリーの “[ Table Name ]” には，OCL のコンテキスト名である “[ Model Element ]” を当てはめる．上記の SQL クエリーの 3 箇所の “[ Table Name ]” には同じモデル要素名を記述する．

“where” 句中の “not in” について少し説明をする．述語 “in” は，以下のような形式をとる．

```
A in (C, D, ... )
```

A が括弧の中のリスト内の任意の値である，“C, D, ...” と一致する場合，“True” を返す．もし，リスト中のいずれかの値が “Null” である場合は，“Unknown” を返す．つまり，“A not in (C, D, ...)” は，“A” がリストに存在しないのであれば，“True” を返し，“A” がリスト中に存在するのであれば “False” を返す．

よって，“[ sonsistent element output subquery ]” は，制約条件を満たしている要素のリストを組とするテーブルを出力するサブクエリーであるとすると，“[ Table Name ].name not in [ consistent element output subquery ]” は，制約条件を満たしているモデル要素集合に “[ Table Name ].name” が存在するならば，“False” を返し，制約条件を満たしているモデル要素集合に “[ Table Name ].name” が存在しないならば “True” を返す．

以上のことから，この SQL クエリーは，制約条件を満たしていない “[ Table Name ].name” の集合を組とするテーブルを出力するものであることがわかる．

OCL の条件の記述に対応する部分は，上の不整合抽出クエリー中の “[ consistent element output subquery ]” のサブクエリー中に記述する．この部分は，上でも述べたように，制約を満たしている要素を抽出するクエリーである．“[ consistent element output subquery ]” は，以下の形式である．

```
select [ Table Name ].name
  from [ Table Name List ]
  where [ Condition Expression ]
```

“[ Table Name ]” には，これまでと同様に OCL の Context のモデル要素が当てはめられる．“[ Table Name List ]” には，次の “where” 句内で使用されたテーブルをすべて，カンマリストとして記述する．そして，“where” 句以下に SQL に変換した制約条件を記述する．

## 6.3　プロパティの参照

OCL 式のプロパティの参照の SQL クエリーへの対応について説明する．OCL 式のプロパティの参照は，2 通りに区別する必要がある．

一つは，モデル要素自身に定められているプロパティの参照と，もう一つは，OCL の仕様に定義されているプロパティの参照である．OCL の仕様に定義されて

いるプロパティを以下に示す.

```
name, attributes, associationEnds, operations, supertypes,
allSupertypes, allInstances, oclType, oclIsKindOf(),
oclIsTypeOf(), oclAsType(), evaluationType, abs, floor, max, min,
div, mod, size, concat, toUpper, toLower, substring
```

ただし，“name” はテーブルの属性名と競合するので，OCL 定義のプロパティとして扱わず，モデル要素自身に定められているプロパティとして扱う．これらの OCL 定義プロパティの意味は，“OMG Unified Modeling Language Specification” [1] の “Object Constraint Language Specification” の章に記述されている．

まず，OCL 定義プロパティであるかどうかの判定について述べる．プロパティ参照の構文を一般的な形式で表現すると以下のようになる．

```
   [ Model Element ].[ Property 1 ].[ Property 2 ].
… .[ Property N ]
```

上記の式は，$N$ 番目のプロパティがモデル要素の属性であれば，$N-1$ 回のナビゲーションを行なっており，$N$ 番目のプロパティがナビゲーション・プロパティであるなら $N$ 回のナビゲーションを行なっている．いずれにせよ，$N-1$ 回のナビゲーションを経て，$N$ 番目のプロパティを参照している．ただし，$N \geq 1$ である．“`[ Model Element ]`” には，モデル要素名を当てはめる．この，最後のプロパティである “`[ Property N ]`” の部分を上で列記した OCL 定義プロパティとマッチするかどうかで判定する．

OCL 定義プロパティの参照は，関係データベースのメタなデータを参照しなければならないなど，SQL クエリーの範囲を超えるものもある．よって，各 OCL 定義プロパティは独自の SQL クエリーを定義するか，もしくは，外部プログラムによって実現する．

単に，あるモデル要素自身のプロパティを参照するだけの場合は，以下のような構文である．

```
   [ Model Element ].[ Property ]
```

本研究で提案する環境では，UML のメタクラスごとにテーブルを作成するので，上記の構文中の “[ Model Element ]” は，テーブルでは，テーブル名に相当し，“[ Property ]” は，テーブルの属性名に対応する．

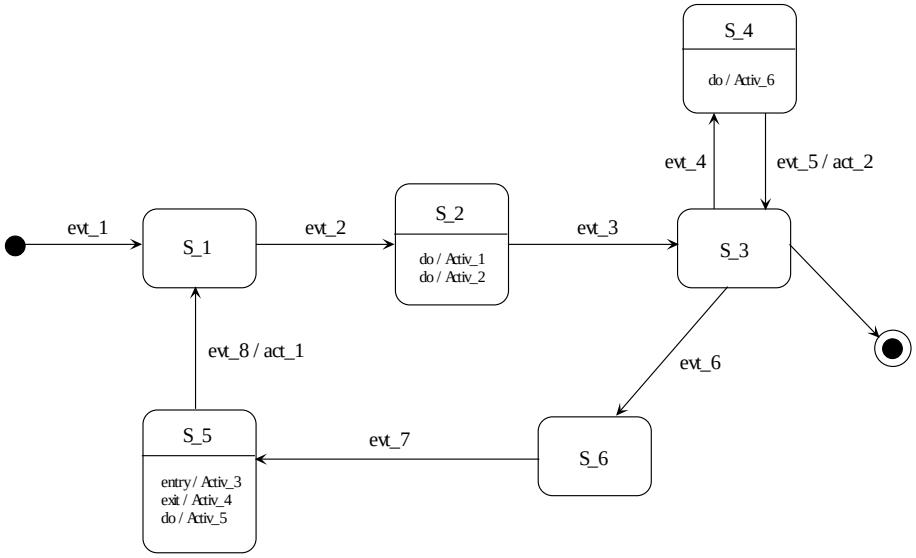


図 6.3: ステートチャート図

例えば，あるシステムのモデルのステートチャートとして，図 6.3 のモデルが与えられているものとしてモデル中の状態の Do アクティビティを参照するとすれば，OCL での記述は以下のようになる．

```
State.doActivity
```

図 6.3 のモデル要素である “State” のテーブルの一部を表 6.1 に示す.「 モデルのテーブル化 」でも示したように，実際には モデル要素 “State” には，表 6.1 のテーブルに記述したテーブル属性名よりも多くのテーブル属性名が存在する．ここでは，簡潔に表すためすべてのテーブル属性名を表示することを避けた．以降，本稿で扱うモデル要素のテーブルについても同様に扱う．

以下に，“State” の “doActivity” を参照するための SQL クエリーを示す.

```
select doActivity from State
```

表 6.1: State テーブル

| name | doActivity | entry | exit | … |
|---|---|---|---|---|
| s_1 | | | | … |
| s_2 | activ_1 | | | … |
| s_2 | activ_2 | | | … |
| s_3 | | | | … |
| s_4 | activ_6 | | | … |
| s_5 | activ_5 | activ_3 | activ_4 | … |
| s_6 | | | | … |
| … | … | … | … | … |

表 6.2: doActivity プロパティの参照クエリーの結果

| doActivity |
|---|
| activ_1 |
| activ_2 |
| activ_5 |
| activ_6 |

$N = 1$ のときは，容易にプロパティ参照の OCL 式を SQL クエリーに変換できることがわかる．上記の SQL クエリーの結果，表 6.3 のテーブルが返される．

よって，OCL 式の “`State.doActivity`” に対して，比較や操作などを行なう場合，activ_1，activ_2，activ_5，activ_6 の 4 つの Do アクティビティが対象になる．

次にナビゲート先のプロパティを参照について説明する．

例えば，図 6.4 のように，モデル要素である “Message” を送信したオブジェクトの “Classifier” を参照する場合，OCL では以下のように指定する．

```
Message.sender.base
```

図 6.5 のシーケンス図を例として説明する．まず，“[Model Element]” の項に “Message” が挿入されているので，“Message” テーブルを参照する．“Message” テーブルを表 6.3 に示す．表 6.3 には，M1，M2，M3，M4，M5 のメッセージが登録されている．

テーブル属性名 “sender” を参照すると，“cr_1”, “cr_2”, “cr_3” の 3 種類のモデ

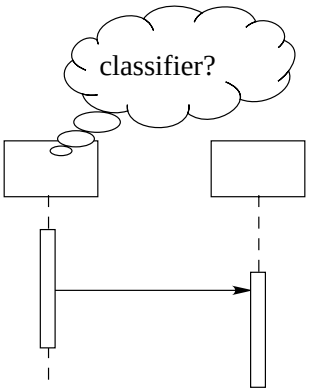


図 6.4: Message の送信オブジェクトの classifier の参照

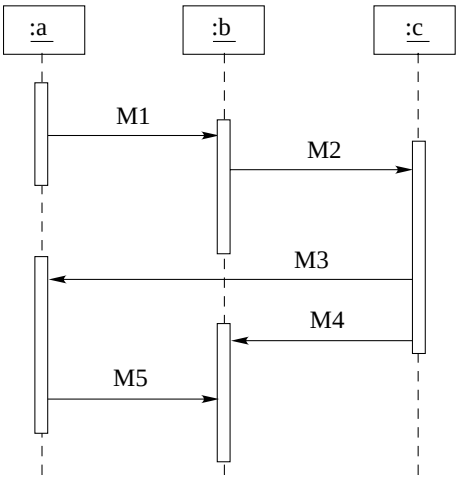


図 6.5: シーケンス図

ル要素名が登録されている．このモデル要素名は，どのモデル要素であるのかを調べるには，図 5.11 の Collaborations のメタモデルを参照すればわかる．

図 5.11 の Collaborations のメタモデル中のメタクラスである “Message” の “sender” をナビゲートすると，“ClassifierRole” を参照できることがわかる．従って，表 6.3 の “Message” テーブルの “sender” に登録されている要素名は，“ClassifierRole” なのである．次に，“ClassifierRole” テーブルの要素である “cr_1”，“cr_2”，“cr_3” のテーブル属性名 “base” を参照すればよい．“ClassifierRole” テーブルを表 6.4 に示す．

“ClassifierRole” の プロパティである “base” は，図 5.11 からわかるように “Classifier” へのナビゲーションである．よって，表 6.4 の “ClassifierRole” テーブルの “base” プロパティは，“Classifier” の要素名を指していることがわかる．表 6.4 の “ClassifierRole” テーブルを参照すると，“cr_1”，“cr_2”， “cr_3” の “base” プロパ

表 6.3: Message テーブル

| name | sender | receiver | ⋯ |
|---|---|---|---|
| M1 | cr_1 | cr_2 | ⋯ |
| M2 | cr_2 | cr_3 | ⋯ |
| M3 | cr_3 | cr_1 | ⋯ |
| M4 | cr_3 | cr_2 | ⋯ |
| M5 | cr_1 | cr_2 | ⋯ |
| ⋯ | ⋯ | ⋯ | ⋯ |

表 6.4: ClassifierRole テーブル

| name | base | ⋯ |
|---|---|---|
| cr_1 | cls_a | ⋯ |
| cr_2 | cls_b | ⋯ |
| cr_3 | cls_c | ⋯ |
| cr_4 | cls_d | ⋯ |
| cr_5 | cls_e | ⋯ |
| cr_6 | cls_f | ⋯ |
| cr_7 | cls_g | ⋯ |
| ⋯ | ⋯ | ⋯ |

ティは，順に，“cls_a”，“cls_b”，“cls_c” であることがわかる．

結果として，“cls_a”，“cls_b”，“cls_c” の 3 つの要素を抽出するための SQL クエリーは以下のようになる．

```
select ClassifierRole.base
  from Message, ClassifierRole
  where Message.sender = ClassifierRole.name
```

ナビゲーションによりさらに多くのテーブルを参照する場合は，“where” 句以降の条件式を “and” によって接続する．

# 第7章　不整合検出手法の適用例

本章では，これまで述べてきたアプローチを，簡略な例に適用して実際に不整合検出ができることを示す．以下の自然言語で示した制約を例として不整合の検出を試みる．

> メッセージはすべて，メッセージを受けとるオブジェクトのステートチャート図において，イベントとして存在しなくてはならない．また，メッセージはすべて，メッセージを送信したオブジェクトのステートチャート図において，アクションとして，存在しなくてはならない．

図 7.1 にステートチャート図 SC_1，図 7.2 にステートチャート図 SC_2，図 7.3 にシーケンス図 SEQ_1 を示す．オブジェクト “o_1” は，クラス “cls_1” のインスタンスとし，“o_2” は，クラス “cls_2” のインスタンスとする．

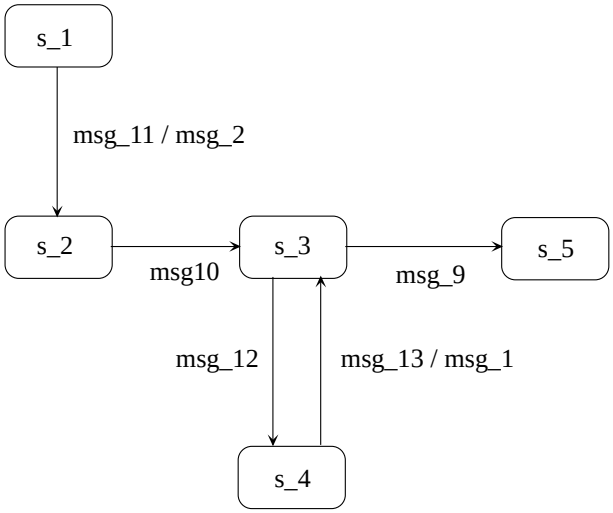


図 7.1: オブジェクト “o_1” のステートチャート図 SC_1

モデルには現れないが，“Transition” 自身にも個々を識別するための名前がつけ

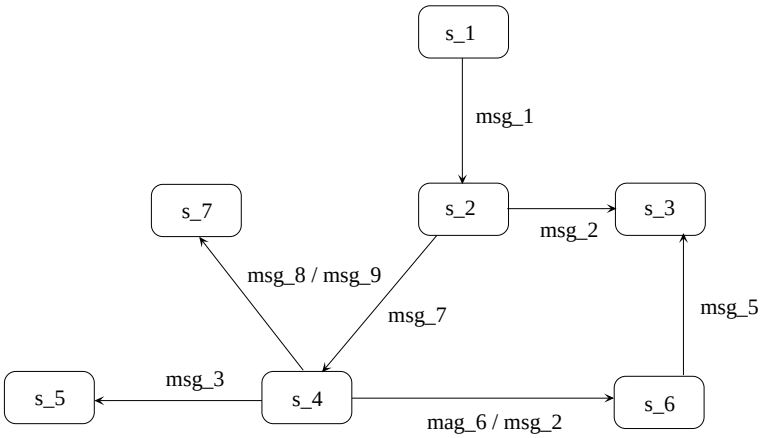


図 7.2: オブジェクト “o_2” のステートチャート図 SC_2

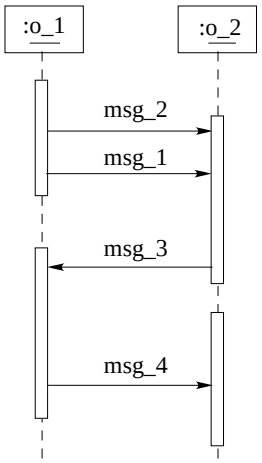


図 7.3: シーケンス図 SEQ_1

られている．図 7.4 に SC_1 の “Transition” 名を，図 7.4 に SC_2 の “Transition” 名を付記したモデルを表す．

以下に図 7.1，図 7.2，図 7.3 のモデルを関係データベースのテーブル化したものを示す．ただし，不整合検出に必要なテーブルのみを表記している．表 7.1 に “Message” テーブルの一部と，表 7.2 に “ClassifierRole” テーブルの一部と，表 7.3 に “Classifier” テーブルの一部と，表 7.4 に “StateMachine” テーブルの一部と，表 7.5 に “Transition” テーブルの一部を示す．

上で示した自然言語で記述した制約の “メッセージはすべて，メッセージを受けとるオブジェクトのステートチャート図において，イベントとして存在しなくてはならない．” の部分を OCL 式で表現すると以下のようになる．

表 7.1: “Message” テーブルの一部

| name | sender | receiver | ··· |
|---|---|---|---|
| msg_1 | cr_1 | cr_2 | ··· |
| msg_2 | cr_1 | cr_2 | ··· |
| msg_3 | cr_2 | cr_1 | ··· |
| msg_4 | cr_1 | cr_2 | ··· |
| ··· | ··· | ··· | ··· |

表 7.2: “ClassifierRole” テーブルの一部

| name | base | ··· |
|---|---|---|
| cr_1 | cls_1 | ··· |
| cr_2 | cls_2 | ··· |
| ··· | ··· | ··· |

表 7.3: “Classifier” テーブルの一部

| name | behavior | ··· |
|---|---|---|
| cls_1 | sc_1 | ··· |
| cls_2 | sc_2 | ··· |
| ··· | ··· | ··· |

表 7.4: “StateMachine” テーブルの一部

| name | context | transition | ⋯ |
|---|---|---|---|
| sc_1 | cls_1 | tr_1 | ⋯ |
| sc_1 | cls_1 | tr_2 | ⋯ |
| sc_1 | cls_1 | tr_3 | ⋯ |
| sc_1 | cls_1 | tr_4 | ⋯ |
| sc_1 | cls_1 | tr_5 | ⋯ |
| sc_2 | cls_2 | tr_6 | ⋯ |
| sc_2 | cls_2 | tr_7 | ⋯ |
| sc_2 | cls_2 | tr_8 | ⋯ |
| sc_2 | cls_2 | tr_9 | ⋯ |
| sc_2 | cls_2 | tr_10 | ⋯ |
| sc_2 | cls_2 | tr_11 | ⋯ |
| sc_2 | cls_2 | tr_12 | ⋯ |
| ⋯ | ⋯ | ⋯ | ⋯ |

表 7.5: “Transition” テーブルの一部

| name | effect | source | target | trigger | ⋯ |
|---|---|---|---|---|---|
| tr_1 | msg_2 | s_1 | s_2 | msg_11 | ⋯ |
| tr_2 |  | s_2 | s_3 | msg_10 | ⋯ |
| tr_3 |  | s_3 | s_4 | msg_12 | ⋯ |
| tr_4 | msg_1 | s_4 | s_3 | msg_13 | ⋯ |
| tr_5 |  | s_3 | s_5 | msg_9 | ⋯ |
| tr_6 |  | s_1 | s_2 | msg_1 | ⋯ |
| tr_7 |  | s_2 | s_3 | msg_2 | ⋯ |
| tr_8 |  | s_4 | s_3 | msg_5 | ⋯ |
| tr_9 | msg_2 | s_4 | s_6 | msg_6 | ⋯ |
| tr_10 |  | s_2 | s_4 | msg_7 | ⋯ |
| tr_11 | msg_9 | s_4 | s_7 | msg_8 | ⋯ |
| tr_12 |  | s_4 | s_5 | msg_3 | ⋯ |
| ⋯ | ⋯ | ⋯ | ⋯ | ⋯ | ⋯ |

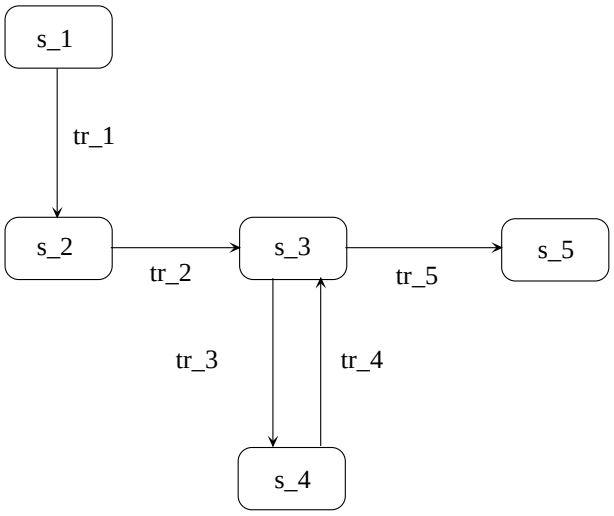


図 7.4: SC_1 中の “Transition” 名

```
context Message inv:
self.receiver.base.behavior.transition.trigger
   -> exists( e | e.name = self.name ))
```

この OCL 式を SQL に変換すると，以下のように表現される．まず，整合性を判定するための SQL クエリーを示す.

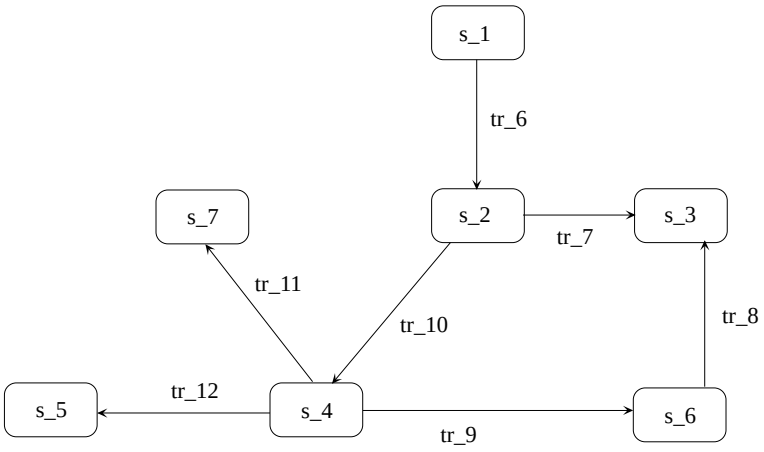


図 7.5: SC_2 中の “Transition” 名

```
select not exists (
  select Message.name
  from Message
  where Message.name not in (
    select Message.name
    from Message, ClassifierRole, Classifier,
         StateMachine, Transition
    where Message.receiver = ClassifierRole.name
      and ClassifierRole.base = Classifier.name
      and Classifier.behavior = StateMachine.name
      and StateMachine.transition = Transition.name
      and Message.name = Transition.trigger
  )
) as boolean;
```

以下，構造化しているサブクエリーを分割して説明する．このクエリーのサブクエリーである以下のクエリーは，制約条件を満たしているメッセージ名を出力するものである．

```
select Message.name
from Message, ClassifierRole,Classifier,
     StateMachine, Transition
where Message.receiver = Object.name
  and ClassifierRole.base = Classifier.name
  and Classifier.behavior = StateMachine.name
  and StateMachine.transition = Transition.name
  and Message.name = Transition.trigger
```

このクエリーは表 7.6 を返す.

表 7.6: サブクエリーの出力テーブル

| name |
|---|
| msg_1 |
| msg_2 |

テーブル全体を把握できるように，上記のクエリーの，“Message.name” を “*” に置き換えたクエリーを以下に示す.

```
select *
from Message, ClassifierRole,Classifier,
     StateMachine, Transition
where Message.receiver = Object.name
  and ClassifierRole.base = Classifier.name
  and Classifier.behavior = StateMachine.name
  and StateMachine.transition = Transition.name
  and Message.name = Transition.trigger
```

このサブクエリーの出力テーブルは，表 7.7 のようになる.

表 7.7: サブクエリーの出力テーブル（その他のテーブル属性も含む）

| name | sender | receiver | name | base | name | behavior | name | context | transition | name | effect | source | target | trigger |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| msg_1 | cr_1 | cr_2 | cr_2 | cls_2 | cls_2 | sc_2 | sc_2 | cls_1 | tr_6 | tr_6 | | s_1 | s_2 | msg_1 |
| msg_2 | cr_1 | cr_2 | cr_2 | cls_2 | cls_2 | sc_2 | sc_2 | cls_1 | tr_7 | tr_7 | | s_2 | s_3 | msg_2 |

表 7.6 は，クエリー中の以下の 5 つの条件を満たしていることがわかる．

```
Message.receiver = Object.name
ClassifierRole.base = Classifier.name
Classifier.behavior = StateMachine.name
StateMachine.transition = Transition.name
Message.name = Transition.trigger
```

しかし，求めたい要素は，制約を満たしていないメッセージなので，前で説明したように，“not in” によって抽出されなかったメッセージ名を抽出する．“not in” を付加したクエリーは以下のようになる．

```
select Messge.name
from Message
where Message.name not in (
   select Message.name
   from Message, ClassifierRole, Classifier,
        StateMachine, Transition
   where Message.receiver = Object.name
     and ClassifierRole.base = Classifier.name
     and Classifier.behavior = StateMachine.name
     and StateMachine.transition = Transition.name
     and Message.name = Transition.trigger
)
```

このクエリーは，結果として表 7.8 を出力する．

このサブクエリーで，“msg_3” と “msg_4” が制約を満たしていない要素として選択された．制約を満たしているかどうかを判定する場合は，組が一つでも存在すれば，“false” となり，組が一つも存在しなければ “true” となる結果を返したいので，“not exists” 述語を付加する．“not exists” 述語を付加することで整合性を

表 7.8: サブクエリーの出力テーブル

| name |
|---|
| msg_3 |
| msg_4 |

判定する SQL クエリーは完成する．その際，“as boolean” とすることで，“true”，もしくは，“false” が入るテーブル属性名を，“boolean” とすることができる．表 7.9 に例の整合性判定の SQL クエリー結果を示す.

表 7.9: 整合性を判定するクエリーの結果

| boolean |
|---|
| false |

整合性を判定する SQL クエリーで “false” と判定されたので，次に，不整合箇所検出のための SQL クエリーを発行して，その箇所をつきとめる．以下に不整合箇所を検出するための SQL クエリーを示す.

```
select message.name
from message
where message.name not in (
  select message.name
  from message, object, statemachine, transition
  where message.receiver = object.name
    and object.behavior = statemachine.name
    and statemachine.transition = transition.name
    and message.name = transition.trigger
)
```

このSQLクエリーは，先程の整合性を判定するクエリーから “`select not exists - as boolean`” を除いたサブクエリーである．“-” は，不整合箇所を抽出するサブクエリーに当たる部分である.

この SQL クエリーの結果を，表 7.10 に示す.

表 7.10: 不整合箇所を特定するためのクエリーの結果

| name |
|---|
| msg_3 |
| msg_4 |

表 7.10 より，“msg_3”，“msg_4” が制約を満たしていないことがわかる．

次に，“メッセージはすべて，メッセージを送信したオブジェクトのステートチャート図において，アクションとして存在しなくてはならない．” という制約をチェックする．この制約を OCL で表現すると以下のようになる．

```
context Message inv:
self.sender.base.behavior.transition.effect
   -> exists( e | e.name = self.name )
```

上記の OCL で記述した制約を SQL クエリーに変換する．整合性を判定するための SQL クエリーは以下のようになる．

```
select not exists (
  select Message.name
  from Message
  where Message.name not in (
    select Message.name
    from Message, ClassifierRole, Classifier,
         StateMachine, Transition
    where Message.sender = ClassifierRole.name
      and ClassifierRole.base = Classifier.name
      and Classifier.behavior = StateMachine.name
      and StateMachine.transition = Transition.name
      and Message.name = Transition.effect
  )
) as boolean;
```

制約を満たしていない要素を抽出するための SQL クエリーは以下のようになる．

```
select Message.name
from Message
where Message.name not in (
  select Message.name
  from Message, ClassifierRole, Classifier,
       StateMachine, Transition
  where Message.sender = ClassifierRole.name
    and ClassifierRole.base = Classifier.name
    and Classifier.behavior = StateMachine.name
    and StateMachine.transition = Transition.name
    and Message.name = Transition.effect
)
```

まず，整合性について判定する．整合性を判定するための SQL クエリーを発行した結果を 7.11 に示す．

表 7.11: 整合性を判定するクエリーの結果

| boolean |
|---|
| false |

整合性の判定の結果は “false” であるので，不整合箇所を特定するための SQL クエリーを発行する．その結果は，表 7.12 である．

表 7.12: 不整合箇所を特定するためのクエリーの結果

| name |
|---|
| msg_3 |
| msg_4 |

よって，“msg_3”，“msg_4” は，制約を満たしていないことがわかる．

# 第8章　おわりに

## 8.1　まとめ

本稿では，UMLで記述されたモデルを関係データベースのテーブルへと対応づけるフレームワークを示した．モデルのテーブル化は，UMLの論理構造に基づいて行なったので，UML領域から関係データベース領域への意味の変換を極力行なわずに済んだ．UMLのすべてのモデル要素のテーブル名とテーブル属性名を列挙した．モデルの管理を関係データベースシステムに委譲することで，UMLの9種類のダイアグラムのモデル要素を一様な扱いをすることが可能であり，OCLで記述されたUMLに対する制約を，関係データベース領域において，SQLクエリーに変換することで，制約を満たしているかどうかをチェックでき，制約を満たしていないのであれば，その箇所をテーブルより抽出することが可能であることを示した．OCL式からSQLへの変換については，SQLクエリーの構造の枠組を示した．

ダイアグラム中の特定のモデル要素の他のモデル要素の対応関係の制約チェックは，本環境で容易にチェックできることを示した．

既存のツールや技術を活用することで，ツールの開発者は，新しい技術を習得する労力を省くことができ，よく知られた技術の利用は，開発者および，利用者のツールへの理解を助け，ツールの発展を促進する．

完璧でオールマイティに活用できる方法論が存在しない以上，ソフトウェア開発におけるモデリングの際に定める制約は多種多彩に存在するはずである．しかしながら，現状では，制約を柔軟にカスタマイズできるツールは存在しない．あらかじめツールに対してツールベンダーが定義した制約のみを強制的に使わされるのは，ソフトウェア開発者達にとって決して便利であるはずがなく，UML自体の可能性を制限しているとも言える．

本研究では，その自由に制約をカスタマイズできるツールの基盤となるアプローチを示した．

## 8.2　今後の課題

OCLからSQLクエリーへの変換は，計算機で扱えるように規則を詳細に定める必要がある．変換規則を定めるためのアプローチとしては，OCLで記述された制約をSQLに人手によって変換し，その変換のパターンを抽出するアプローチで行

なう．その為に，様々な例に適用してみることが必要である．OCL から SQL への変換は，OCL の各ステートメントごとに対応する SQL 式が生成できることをある程度確認できている．

本研究では，モデル要素間の対応関係のチェックについて適用したが，他にも，モデル要素間の順序関係に関するチェックなども本研究のアプローチで行なえる可能性がある．本研究のアプローチでどのような種類の制約についてチェックが可能で，どのような種類の制約のチェックが困難であるのかを明らかにする．

# 謝辞

本研究を行なうにあたり終始御指導を賜わった片山卓也教授に深謝致します．また，青木利晃助手には有用でかつ，適切な御助言をいただきました．伊藤恵助手には日頃のちょっとした質問や相談にも快く応じていただきました．立石孝彰氏は，常に私の研究を気にかけていただき，しばしば長時間にもわたる研究に関する相談にも付き合って下さりました．心から感謝致します．

最後に，本論文をまとめるに当たって有意義な御意見と研究環境を整備して頂きましたソフトウェア基礎講座の諸兄に厚く御礼申し上げます．

2000 年 2 月 15 日　馬場 茂雄

# 参考文献


[1] “OMG Unified Modeling Language Specification Version 1.3”, OMG(Object Management Group), 1999.

[2] Ivar Jacobson，Grady Booch，James Rumbaugh 著，日本ラショナルソフトウェア株式会社 訳，藤井拓 監修，“UMLによる統一ソフトウェア開発プロセス”，翔泳社，2000.

[3] Joseph Schmuller 著，多摩ソフトウェア訳，長瀬嘉秀 監訳，“独習 UML”，翔泳社，2000.

[4] Martin Fowler 著，羽生田栄一 監訳，“UML モデリングのエッセンス 第2版”，翔栄社，2000.

[5] Hans-Erik Eriksson，Magnus Penker 著，杉本宣男，落合修，竹田多美子 監訳，“UMLガイドブック”，トッパン，1998.

[6] 杉野博，安井勝俊 著，今野 睦 監修，“オブジェクトモデリング プロセスガイド”, ピアソン，2000.

[7] C.J.Date，Hugh Darwen 著，QUIP LLC 訳，“標準SQLガイド 改訂第4版”，アスキー出版局，1999.

[8] C.J.Date，藤原譲 訳，“データベース・システム概論”，丸善，1984.

[9] 横田一正，宮崎収兄 著，“新データベース論”，共立出版，1994.