7. 付録

付録では、UML と OCL の概要を説明する。また、本書の参考文献を紹介する。

7.1UML1.1 概要

ここでは、UML（Unified Modeling Language）[*1] の記法を中心にその概要を説明する。UML の中心的な図である UseCase 図・クラス図・状態遷移図・順序図・配置図を、本書に出てきた記法を中心に解説する。

7.1.1UseCase 図

システムの要求を機能単位でまとめるための図である。ただし、オブジェクト指向なので、オブジェクトになるであろうアクターの周りに機能をまとめる方向で記述していく。

システム外部のオブジェクトや、内部のオブジェクトのロール (役割) を表すアクターと、ある独立した機能を表す UseCase からなる。UseCase 図は UseCase で表す機能を適当な大きさになるまで分解していくので、階層を構成する。

下図で、四角で表されるのが UseCase で、楕円で現されるのが 1 階層下の UseCase である。人マークで表されるのがアクターである。

破線の矢印は、矢印の元の UseCase が、矢印の先の UseCase に依存していることを示す。[*2]

---

1. 『UML Notation Guide version 1.1, Rational Software, 1 September 1997』参照。詳細な意味的定義は、『UML Semantics, Rational Software, 1 September 1997』参照。
2. どういう種類の依存かまでは、UseCase では記述しない。

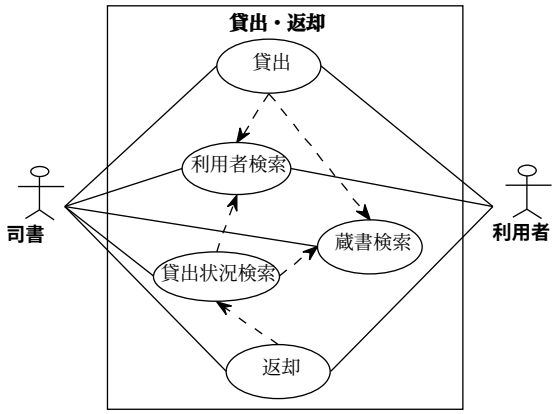

図 88 UseCase 図の例

7.1.2 クラス図

システムの静的な構造である、クラス同士の関係 (継承）とインスタンス同士の関係 (関連) を同時に表す図である。

以下の図で、クラス「本実体」は、属性「貸出中」と「ID」を持ち、操作「貸出可能か」「貸し出す」「返却する」を持つ。

「貸出可能か」操作は、Boolean 型[*3] の返値を返す。操作名のあとに何も書いていない操作は、返値がない。「貸し出す」操作は、引数「ある貸出」を持ち、その型は「貸出」である。引数は引数リストとして複数持てる。その場合は、各引数をカンマで区切る。

本実体は本クラスのサブクラスであり、本クラスは抽象クラス[*4] 著作物のサブクラスである。{abstract} は抽象クラスあるいは抽象操作[*5] であることを示す注釈であるが、斜体の文字列も抽象クラスあるいは抽象操作であることを示す。白抜きの三角形は継承を表し、三角形の頂点側がスーパークラス、底辺側がサブクラスである。

---

3. あるいはクラス。UML では型とクラスをあまり区別しない。
4. いくつかの基本操作が抽象操作であって、インスタンスを持てないクラスを言う。
5. 操作のインターフェースのみを定義し、サブクラスで実装することを宣言した操作のこと。

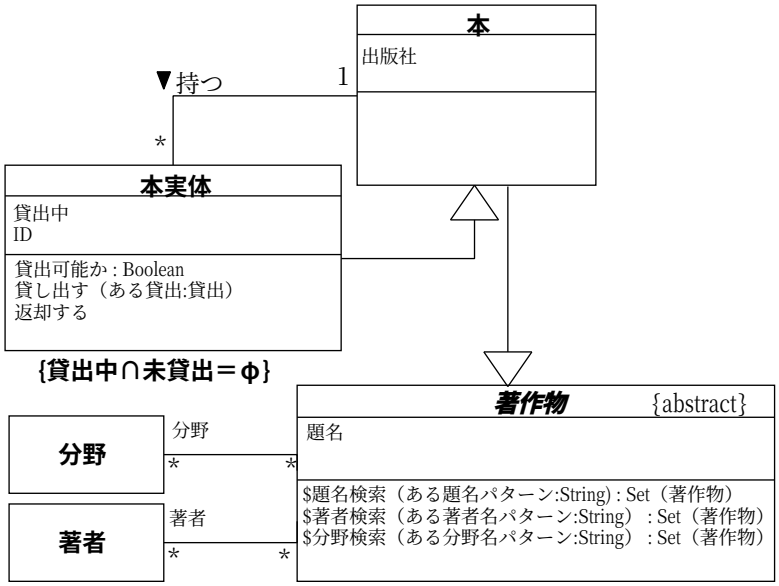

図 89 クラス図の例

著作物クラスの題名検索などのように、名前の頭に $[*6] 印が付いている操作はクラスレベルの操作[*7] である。

本のインスタンスは、本実体のインスタンスと 1 対多のリンクで結ばれていて、お互いにメッセージを投げることができる。このリンクの束を「関連」と言い、今の場合「持つ」という名前を付けている。アスタリスク記号（＊）は、多であることを示すので、分野クラスのインスタンスと著作物のサブクラスのインスタンスは多対多の関連があることになる。分野クラスのオブジェクトは、著作物クラスのサブクラスのオブジェクトから見ると「分野」という役割を果たすのでロール名「分野」

6. 最新の UML では、下線の付いた操作名が、クラスレベル操作を表すが、本書では見やすさのため、旧来の記法を用いた。
7. Smalltalk ではクラス操作、C++ では static なメンバー関数になる。

が関連を表す直線の「分野」クラスよりに付いている。今の場合、クラス名とロール名が同じだが、異なる場合もある[*8]。

{} の中に OCL や自然言語で制約を書く。UseCase の場合と同じく、破線の矢印は依存関係を表す。

クラス名の上に <<>> で囲んだものは、ステレオタイプである。下図の場合は、速度センサークラスが Adaptor の役割をすることを示している。

また、属性や操作名の頭に－や＃や＋を付けて、それらのスコープ（有効範囲）を示す。下図では、2 つの属性と 3 つの操作に－が付いていて、私有属性あるいは私有操作であることを示している。

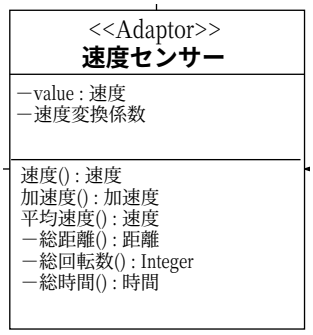

図 90 スコープの例

スコープの記号は以下の意味になる。

- ＋

  他のオブジェクトに公開された公開属性または公開操作であることを示す。記号が省略されている場合は、＋の意味になる。

- ＃

  公開が制限された属性または操作であることを示す。

8. 例えば、人クラスにロール名「従業員」が付いた会社クラスとの関連が考えられる。

- **－**

  他に公開されていない、自分のクラスの操作からのみアクセスできる私有属性や私有操作であることを示す。

関連自身をクラスにした方が都合がよいことがある。関連が属性や操作を持って、ある責任を果たす場合である。関連に破線を付けたクラスを付けて下図のように表す。

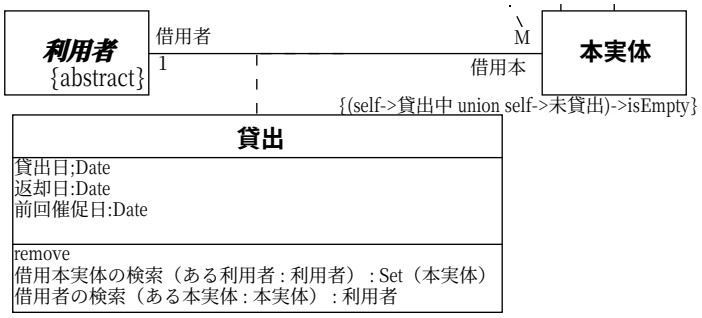

図 91 関連クラスの例

関連が全体部分関係や has-a 関係を示すとき、集約と呼んで、下図のように菱形を付けた線分で表す。今の場合、巡航制御オブジェクトがセンサーとアクチュエーターオブジェクトを「持って」いる。

2 つのオブジェクト間の関係で、片方だけが相手を知っていればよい場合、下図のように 1 方向関連として矢線で表す。矢線の元のクラスのオブジェクトが、矢線の先のクラスのオブジェクトを知っているが、その反対は無いことを示す。

また、関連名に方向性があるとき、その名前の方向を下図のように黒三角形で表す。

注釈として制約などを記入したいとき、下図のように四角の角が折れた四角形の中に記述する。

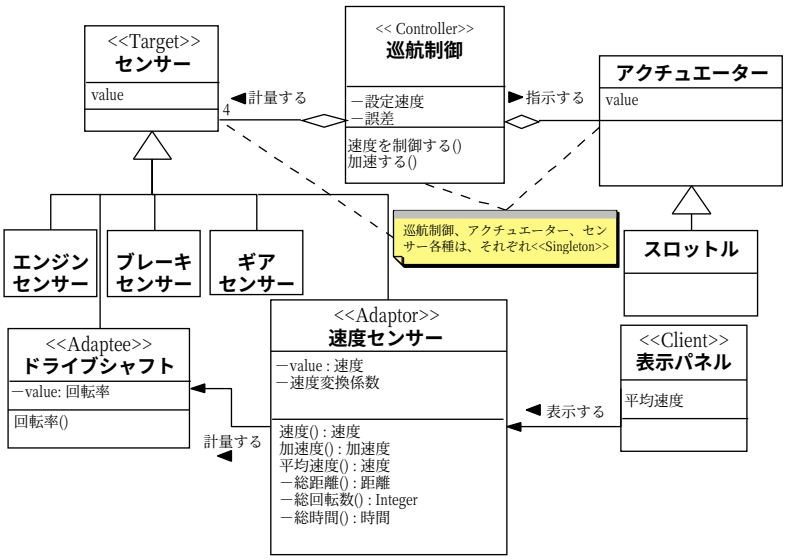

図 92 集約と 1 方向関連の例

7.1.3 状態遷移図

システムの動的振る舞いを表すための図である。このために、下図のようなDavid Harel の STATECHART を改良した状態遷移図を使う。

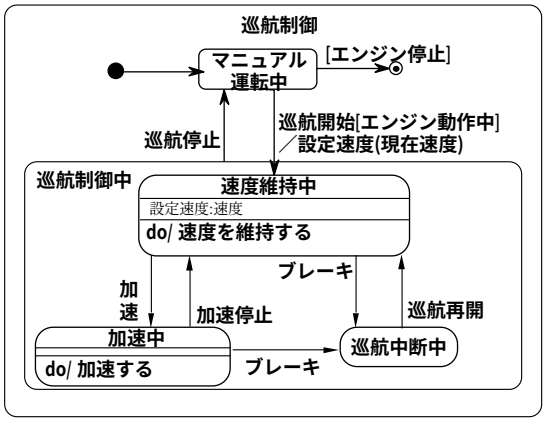

図 93 状態遷移図の例

ここで、状態は角の丸い四角形で表し、遷移を矢印で表す。遷移に付いている文字列は以下の構文になっている。

仕様 39 **遷移の構文**

遷移 →イベントシグネチャ '[' ガード条件 ']' '/' 動作式 '^' 送付句

イベントシグネチャ → イベント名 '(' 引数 ',' . . . ')'

イベント名 → 遷移を起こすイベントの名前

ガード条件 → 遷移が起こる条件を表す OCL 条件式

動作式 → 遷移が発生したときに実行される式

送付句 → 対象式 '.' メッセージ名 '(' 引数 '.' . . . ')'

対象式 → オブジェクトかオブジェクトの集合を返す式

メッセージ名 → 対象となるオブジェクトにある操作名

例えば、下記のような記述になる。

仕様 40 遷移の記述例

```
巡航開始 ( 開始時刻 ) [ エンジン動作中 ] / 設定速度 ( 現在速度 )
^ 表示パネル . 巡航開始表示 ()
```

状態「速度維持中」は状態「巡航制御中」のサブ状態であり、状態「巡航制御中」は状態「巡航制御」のサブ状態である。

状態「巡航制御中」の四角から出た巡航停止イベントによる「マニュアル運転中」への遷移は、「巡航制御中」のどの状態にいても、この遷移が発生することを示す。

状態を表す角の丸い四角形は、上から状態名・属性・内部遷移を表す。内部遷移は、他の状態への遷移を起こさない遷移で、構文は以下のようになる。

仕様 41 内部遷移の構文

```
遷移 →イベントシグネチャ '[' ガード条件 ']' '/' 動作式
イベントシグネチャ → イベント名 '(' 引数 ',' . . . ')'
イベント名 →
    遷移を起こすイベントまたは疑似イベント（entry ｜ exit ｜ do）の名前
ガード条件 → 遷移が起こる条件を表す OCL 条件式
動作式 → 遷移が発生したときに実行される式
'entry' '/' 動作式 → この状態に入るときに起動する動作式を示す
'exit' '/' 動作式 → この状態から出るときに起動する動作式を示す
'do' '/' 状態マシン名 '(' 引数 ',' . . . ')' →
    この状態で実行する下位の状態マシンを示す
```

黒丸から出た矢印は「初期遷移」を表し、その状態のレベルで最初どの状態にいるかを示す。2 重丸への矢印は、そのレベルの状態を抜けることを示す。

H を○で囲った記号は、履歴状態指標（history state indicator）を意味し、状態が「前の状態」を覚えていることを示す。

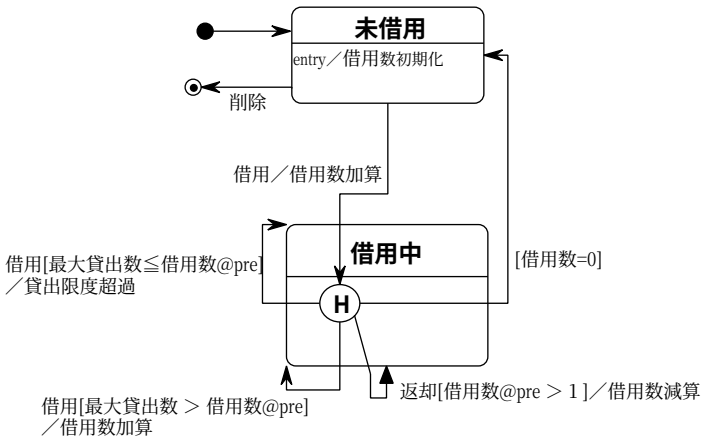

図 94 履歴状態指標の例

7.1.4 順序図

順序図はオブジェクト間にどのような順番でメッセージが流れるかを記述するための図である。四角の箱の中にクラス名が記述されているが、下線を付けて、そのクラスのあるインスタンスであることを示す[*9]。メッセージは横方向の矢印で示し、メッセージ名 (パラメータ) : 返値という形式をとる。

時間は上から下に流れ、オブジェクトの下の縦棒 (生命線）がオブジェクトを表す。破線の矢印は、制御の戻りを示す[*10]。

生命線が破線の時、そのオブジェクトは生まれていない。生命線が実線の時、そのオブジェクトは生成はされているが活性化していない。生命線が縦に細長い四角の時、そのオブジェクトは活性化[*11] している。

---

9. 使用ツールの関係で、下の例では下線が付いていない。
10. 制御の戻りを強調する場合に記述する。
11. 活性化していることを強調したい場合にこの記法を使う。

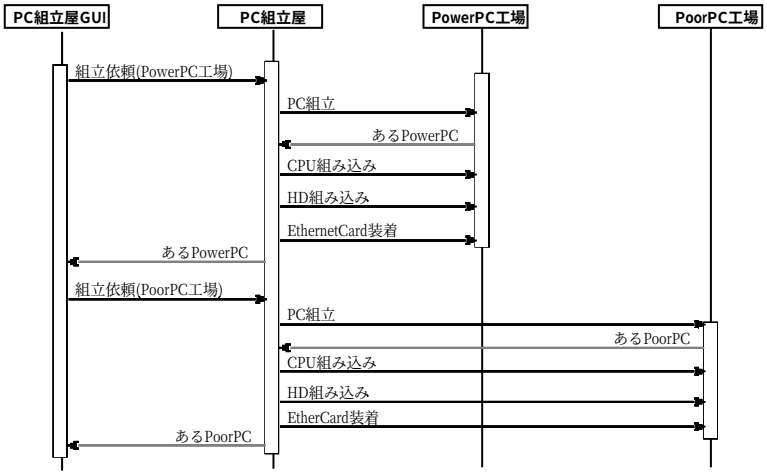

図 95 順序図の例

7.1.5 配置図

配置図は、実行時のオブジェクトやプロセスやその他のコンポーネントの配置状況を記述するためのものである。

ここでは、司書クライアントの Mac に貸出処理画面と検索画面という２つのアプリケーションがあり、図書館サーバーの UNIX の利用者クラスと著作物クラスに依存していることを示している。

下図で、貸出・題名検索などの名前が付いた「◯の付いた線」は、実行時のコンポーネントが提供するサービスを示し、破線の矢印は「依存関係」を示す。例えば、検索画面は著作物オブジェクトの題名検索・分野検索・著者検索の 3 つのサービスに依存している。

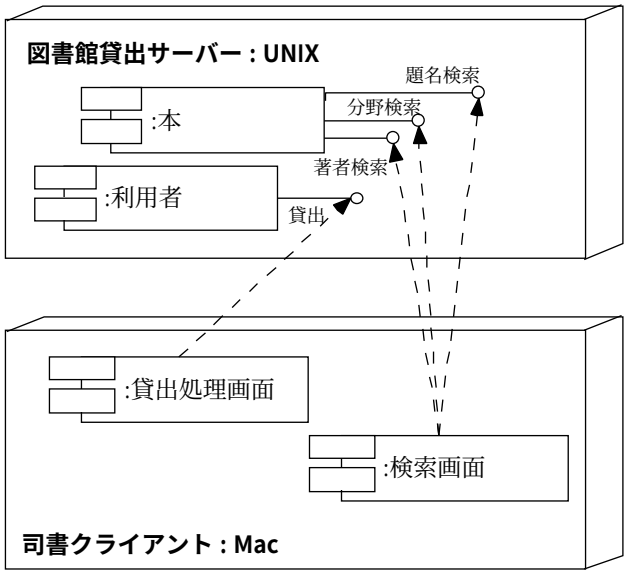

図 96 配置図の例

7.2 仕様記述言語（OCL）概要

1997 年秋に、UML に追加された形式仕様記述言語である。元々は、IBM の保険部門で開発された Syntropy メソッドに由来する。以下に示す特徴がある。

- **参照の透過性**
  式や記述の一部をそれと等価なものに書き換えても、全体の評価値や意味が変わらないという性質がある。このため、副作用がなく状態が変化しないという、信頼性を維持するための性質を満たしている。
- **プログラミング言語ではないため、実装に関わることは表現できない**
  プログラムロジックとフロー制御は書けない。ただし、本書ではデザインパターンの一部の詳細な実現方法を記述するため、RSL という他の形式仕様記述言語から一部の構文を拝借した。RSL については次の節で説明する。

- **型のある言語**

  C++ や Java と同じく型のある言語であるが、型とクラスを同一視している。

- **再帰可能**

  ある操作の仲で自らを呼ぶことができる再帰呼び出しが可能である。

- **ASCII 文字だけで構成される**

  本書では、String には日本語文字も使えることにした。

7.2.1RSL から追加した構文

本書では、RSL（Raise Specification Language）という形式仕様記述言語から代入やフロー制御のための構文を借りてきて OCL の仕様に追加している。RSL については、『The RAISE Language Group, The RAISE Specification Language, Prentice Hall, 1992』参照のこと。追加部分は、以下の通りである。

(1) 変数への代入

変数 := OCL 式

という代入のための構文を追加した。

(2) 繰り返し文

while 蓋の条件[*12] **do**

OCL 式

end;

という繰り返し文を追加した。

(3)if~then~elsif 文

if 条件文 1 then

OCL 式 1

---

12. 繰り返すための条件式である。

elsif 条件文 2 then

OCL 式 2

end

という形の if~then~elsif 文を追加した。

7.2.2OCL を使う場所

OCL を使うのは以下の場所である。

- **クラスモデルのクラスや型の不変条件の記述**

  クラスや型で不変の条件を記述する。

- **操作とメソッドの前件と後件の記述**

  操作の起動前の状態 (前件) と終了後の状態 (後件) を記述する。

  操作の定義形式は以下のようになる。

  型名[*13] :: 操作名 (引数 1 : 型 1, ...): 返値の型

  pre[*14] : 引数 1 > …

  post[*15] : result = ...

  後件の記述中、「値 @pre」という形式で、前件の時点の値を指定することができる。例えば、以下のように指定できる。

  post : 年齢 = 年齢 @pre + 1 -- 前件中の年齢に 1 を加えて年齢とする。

- **操作の制約の記述**

  操作 (引数リスト）= 式

  という形で、操作本体の制約を記述する。

- **状態遷移図のガード条件の記述**

- **ナビゲーション言語**

  着目しているオブジェクトから、他のオブジェクトの特性 (Property) を参照するための言語として使う。

---

13. 操作が定義されている型名を示す。
14. 前件 (pre-condition) を記述する。
15. 後件 (post-condition) を記述する。

7.2.3OCL の型

OCL は以下の基本型を持つ。

表 7.1OCL の基本型

型	値の例	演算子の例
Boolean	true, false	and, or, xor, not, implies, if-then-else
Integer	1,2,-32	*,+,-,/,abs
Real	1.5, 3.14, -2.0	*,+,-,/,floor
String	'To eat or not to eat'	concat, substring

モデル上のすべての型とクラスは、OCL の型となる。すなわち、型とクラスを区別しない。型の一致しない型同士の演算はできない。型の一致規則は以下の通りである。

- **ある型はスーパー型と一致する**
- **型の一致は推移律を満たす**

例えば、String と Integer の演算はできないが、Integer のスーパー型が Real なので、Integer と Real 間の演算は可能である。

表 7.2 スーパー型

型	スーパー型
Set	Collection
Sequence	Collection
Bag	Collection
Integer	Real

OCL であらかじめ定義された操作である oclAsType を使って、ある型のオブジェクトをその型のスーパー型に変換することができる。

オブジェクト .oclAsType(スーパー型)

enum{#value1,#value2,#value3} という形の列挙型が使える。各要素は同じ型でなければならない。

7.2.4 演算子の優先順位

演算子の優先順位は以下のようになる。

- . **演算子と** -> **演算子**
- **単項** not **と単項－**
- **＊と／**
- **＋と** 2 **項演算子－**
- and, or, xor
- implies
- if-then-else-endif
- <, >, <=, >=, =

7.2.5 注釈

2 個のダッシュ「--」の後、行末までに注釈を書く。

```
-- これは注釈です。
```

のようになる。

7.2.6 未定義の値

照会結果が未定義の場合、全体の ocl 式も未定義になる。

ただし、「true or 式」なら常に true で、「false and 式」なら、常に false である。

7.2.7 特性 (Property)

以下の項目を特性 (property) と呼び、

```
型
self. 特性
```

という形式で表記する。

(1) 属性

例えば、人クラスの属性「生年月日」は下記のように表す。

人

self. 生年月日

(2) 照会操作

副作用のない操作は特性とする。副作用のない操作は、属性とあまり区別が付かないからである。

例えば、人クラスの操作「年齢」は次のように表記する。

人

self. 年齢 ()

(3) 関連の端 (AssociationEnd)

「self. ロール名」という形式で、ロール名の付いた相手オブジェクトの集合を記述する。相手側の多重度が 1 か 0 ならば、「self. ロール名」は相手のオブジェクトを示す。

Set や Bag や Sequence という Collection は、OCL であらかじめ定義された型であり、多くの操作がすでに定義されている。Collection 自身の特性は「-> 特性」という形式で指定する。

例えば、下記のようなクラス図を考える。

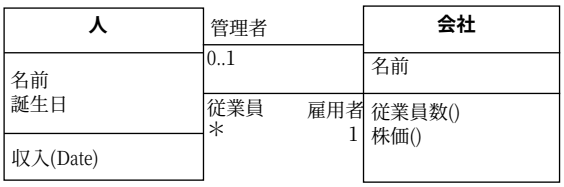

図 97 会社と人の関連

特性の指定は以下のようになる。

仕様 42 特性の指定例

会社

self. 管理者 -- 会社の管理者である人クラスのインスタンス

self. 従業員 -- 会社の従業員である人オブジェクトの集合

self. 従業員 ->size -- 人オブジェクトで、従業員である人の個数 = 人数

self. 従業員 ->isEmpty -- 従業員が居なければ true

関連にロール名が記述されていないときは、その型の名前がロール名になる。[*16]

(4) 特性 (Property) の結合

特性を結合させて、より複雑な式にしていくことができる。

下の例は、特性を結合させて不変条件を記述した例である。

仕様 43 不変条件の記述例

- **会社には高々 50 人の従業員しかいない**

  self. 従業員 ->size <= 50

- **結婚は男と女の間で行われる**

  self. 妻 . 性 = # 女 and self. 夫 . 性 = # 男

- **人は夫と妻を同時に持つことはできない**

  not ((self. 妻 ->size = 1) and (self. 夫 ->size = 1))

- **日本での結婚条件**

  self. 妻 ->notEmpty implies[*17] self. 妻 . 年齢 >= 16 and

16. 型名が英語の時は、大文字で始まっている型名の頭一文字を小文字にして名前の重複を避ける。型名が日本語の場合、本書では、型名の後ろに「役」を付けて名前の重複を避ける。
17. 数学的には⇒で表される操作。A implies B は not A or B と同値である。

self. 夫 ->notEmpty implies self. 夫 . 年齢 >= 18 and
self. 妻 . 性 = # 女 and self. 夫 . 性 = # 男 and
not ((self. 妻 ->size = 1) and (self. 夫 ->size = 1))

7.2.8 すべてのオブジェクトに共通の定義済みの操作

すべてのオブジェクト共通の操作が以下のように定義されている。

仕様 44 オブジェクト共通の操作

- **あるオブジェクト** .oclType : oclType
  あるオブジェクトの型を返す。oclType は OCL の型を示す。
- **あるオブジェクト** .oclIsTypeOf(**ある型** : oclType) : Boolean
  ある型があるオブジェクトの型と等しいかどうか判定する。
- **あるオブジェクト** .oclIsKindOf(**ある型** : oclType) : Boolean
  ある型があるオブジェクトのスーパー型か判定する。
- **ある型** .allSupertypes : Set(oclType)
  ある型のすべてのスーパー型を返す。
- **ある型** .allInstances
  ある型のインスタンスすべてを返す。
  例えば、以下の式は、選手クラスのすべてのオブジェクトで、選手 p1 と選手 p2 が等しくないならば、選手 p1 の背番号と選手 p2 の背番号も等しくないことを示している。
  選手 .allInstances->forAll(p1, p2 | p1 <> p2 implies p1. 背番号 <> p2. 背番号)

7.2.9 Collection

物の集まりを表す OCL の型 Collection は、oclAny のサブ型 (部分型）であり、重複ある物の集まりを表す Bag 型と、要素の重複を許さない物の集まり（集合）を表す Set 型と、重複が許され、要素に順序がある[*18] Sequence 型の３つをサブ型として持つ。

- Bag -- **重複のある物の集まり (順序はない)**

  Bag {1, 3, 4, 3, 5, 1}

- Set -- **数学の集合（重複はなく、順序もない）**

  Set{1, 2, 3, 4, 5, 6} = Set{ Set{1, 2}, Set{3, 4}, Set{5, 6}}

- Sequence -- **要素に順序がある物の集まり（重複がある）**

  Sequence{ 1, 3, 45, 2, 3} -- 45 は 3 番目の要素

  Sequence{1..10} = Sequence{1..(6+4)} = Sequence{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

7.2.10 Collection の既定義操作

Collection にあらかじめ定義されている操作のうち主なものを説明する。

(1)select 操作

集合から抽出する操作である。引数で指定された条件の集合を抽出する。

仕様 45 select 操作使用例

会社

-- 50 歳より上の従業員の集合を求める

self. 従業員 ->select(年齢 > 50)

self. 従業員 ->select(p | p. 年齢 > 50) -- 同上

18. 要素の値の順番にソートされているということではなく、何番目の要素かが問題にされると言うこと。

self. 従業員 ->select(p : 人 | p. 年齢 > 50) -- 同上

(2)reject 操作

集合から削除する操作である。引数で指定された条件を満たさない要素の集合を抽出する。

仕様 46 reject 操作仕様例

会社

-- 50 歳以下の従業員の集合を求める

self. 従業員 ->reject(p. 年齢 > 50)

-- 18 以上の未婚従業員

self. 従業員 ->reject(p. 年齢 < 18 or p. 既婚か)

(3)collect 操作

元の Collection から異なる Collection を取り出す。

仕様 47 collect 操作仕様例

会社

-- 全従業員の誕生日の Bag(重複有り) を求める

self. 従業員 ->collect(誕生日)

-- 全従業員の誕生日の Set(重複無し) を求める

self. 従業員 ->collect(誕生日)->asSet

以下の形で、collect 操作の省略記述ができる。

collection->collect(特性名) = collection. 特性名

例えば、以下の左辺と右辺は同値である。

self. 従業員 ->collect(誕生日) = self. 従業員 . 誕生日

(4) 限量子（forAll, exists）

- forAll 操作

  数学的には∀という記号で表される操作で、引数で与えられた条件がいつも成り立っていれば true を返す。

仕様 48 forAll 操作仕様例

チーム

-- 全選手の背番号が異なっていれば true を返す。

self. 選手 ->forAll(e1, e2 | e1<> e2 implies e1. 背番号 <> e2. 背番号)

- exists 操作

  数学的にはヨという記号で表される操作で、引数で与えられた条件が 1 つでも成り立てば true を返す。

仕様 49 exists 操作仕様例

会社

-- 従業員に一人でも名前が ’ 伸 ’ がいれば true を返す。

self. 従業員 ->exists(p : 人 | p. 名前 = ’ 伸 ’)

(5) その他の Collection 操作

- collection->size : Integer -- collection の要素数
- -- object が collection に含まれていれば真
  collection->includes(object : OclAny) : Boolean
- -- collection に含まれている object の個数
  collection->count(object : OclAny) : Integer
- collection->isEmpty : Boolean -- collection が空なら真
- collection->notEmpty : Boolean -- collection が空でなければ真
- collection->sum : T -- collection の全要素の合計

(6)Set の操作

Collection のサブ型である、Set 型の操作は以下の通りである。

- set->union(set2 : Set(T)) : Set(T) -- 集合の和 set ∪ set2
- set->union(bag : Bag(T)) : Bag(T) -- set asBag ∪ bag
- set = (set2 : Set) : Boolean -- 同値 set = set2
- set->intersection(set2 : Set(T)) : Set(T) -- 共通集合 set ∩ set2
- set->intersection(bag : Bag(T)) : Set(T) -- set ∩ bag asSet
- set ー (set2 : Set(T)) : Set(T) -- 集合の差 set \ set2。set2 の要素を除く set
- set->including(object : T) : Set(T) -- 要素の追加 set ∪ {object}
- set->excluding(object : T) : Set(T) -- 要素の削除 set \ {object}
- set->asSequence : Sequence(T) -- Sequence への変換
- set->asBag : Bag(T) -- Bag への変換

(7)Bag の操作

Set 型の操作と同様な操作が定義されている。

7.3 参考文献

7.3.1 オブジェクト指向

(1) Bertrand Meyer **著、酒匂寛** , **酒匂順子 訳、**Object-Oriented Software Construction **オブジェクト指向入門、アスキー、**1990

オブジェクト指向の必要性を、プログラミング技術面から分かりやすく解説した、他の文献からの参照度が非常に高いバイブル的な本である。すでに、絶版になっているが、今年夏には第 2 版が出る予定である。

(2) J.Rumbaugh, M. Blaha , W.Premerlani, F.Eddy, and W.Lorensen. **羽生田訳** . **オブジェクト指向方法論：ＯＭＴ** . **トッパン** , 1992

現時点で日本語になっている中で、最も完成された OOA/OOD 技法 OMT の教科書である。OMT の記法は UML の元になった。

(3) J.Rumbaugh, M. Blaha , W.Premerlani, F.Eddy, and W.Lorensen。Solution Manual Object-Oriented Modeling and Design。Prentice Hall, 1991

上の本の演習問題の解答集である。

(4) I. **ヤコブソン** , **他著、西岡・渡邊・梶原監訳、オジェクト指向ソフトウェア工学** OOSE use-case **によるアプローチ、アジソン ウェスレイ・トッパン、**1995.

OOA/OOD 技法、特に UseCase の教科書である。プロジェクト管理やテストについても概要が説明されている。

(5) **青木淳 著、オブジェクト指向システム分析設計入門、ソフト・リサーチ・センター、**1992

OOA/OOD/OOP の分かりやすい解説書である。

(6) **青木淳 著、例題による！！ オブジェクト指向分析設計テクニック、ソフト・リサーチ・センター、**1994

OOD/OOP の分かりやすい解説である。

(7) **佐原伸 著、オブジェクト指向システム分析／設計 Ｑ ＆ Ａ、ソフト・リサーチ・センター、**1995 **年** 11 **月**

オブジェクト指向分析・設計の適用で感じる疑問点に答えた本である。

(8) UML Notation Guide version 1.1、Rational Software、1 September 1997
UML の記法を定義したマニュアルである。

(9) UML Semantics, Rational Software, 1 September 1997
UML の厳密な意味定義を、OCL と自然言語を使って行ったマニュアルであり、OCL による記述例にもなっている。

7.3.2 デザインパターン

(1) Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides 著、Design Patterns : Elements of Reusable Object-Oriented Software、Addison-Wesley、1994.
デザインパターン教科書の定番であるが、共著のせいか、各パターンの記述レベルが統一されていない。

(2) Sherman R.Alpert, Kyle Brown, Bobby Woolf 著、The Design Patterns Smalltalk Companion、Addison Wesley、1998
Smalltalk に最適化したデザインパターンの教科書である。

(3) T.J. トーマス , R.C. マルボー著、大谷真監訳、CORBA Design Pattern、IDG コミュニケーションズ、1998 年
CORBA のデザインパターンの解説であるが、具体性はあまりない。

(4) Martin Fowler, "Analysis Patterns: Reusable Object Models"Addison-Wesley, 1997
分析パターンの教科書である。

(5) David Hay, "Data Model Patterns: Convention of Thought", DorsetHouse, 1996
分析パターンの教科書である。

(6) 青木淳　著、「Smalltalk イディオム」、ソフト・リサーチ・センター、1997 年
Smalltalk のイディオムを記述した教科書である。

7.3.3 仕様記述

(1) Object Constraint Language Specification、Rational Software、1 September 1997

OCL の言語定義マニュアルである。

(2) The RAISE Language Group **著**、The RAISE Specification Language、Prentice Hall、1992

仕様記述言語 RSL の言語解説である。

(3) The RAISE Language Group **著**、The RAISE Development Method、Prentice Hall、1995

RSL での仕様記述法や段階的詳細化や正当性チェックを解説した、RAISE 開発技法の教科書である。

(4) **中川 中著** . **代数的仕様記述言語** CafeOBJ. SRA, 1993

CafeOBJ の解説マニュアルであるが、日本語で書かれた最も分かりやすい仕様記述言語の教科書でもある。

(5) D. **グリース著、筧捷彦訳、プログラミングの科学、培風館、**1991 **年**

段階的詳細化について述べた歴史的名著。

7.3.4 アルゴリズムとデータ構造

(1) **島内剛一他編、アルゴリズム辞典、共立出版、**1994 **年**

日本のソフトウェア科学界が総力を挙げた、アルゴリズム辞典。

(2) A.V. **エイホ他著、原田賢一訳、コンパイラ　原理・技法・ツール、サイエンス社、**1990 **年**

コンパイラー技術のバイブル的教科書。

7.3.5 その他

(1) Edited by John A. McDermid, Software Engineer's Reference Book, Butterworth-Heinemann Ltd., 1991

ソフトウェア技術者のための網羅的参考書。デザインパターンの基礎となるコンピュータ科学やソフトウェア工学の知見が、各分野ごとに見事にまとめられている。

(2) **佐原伸、プログラマのデスクトップ、ソフトウェア技術者協会、**SEAMAIL 1996 **年**

パズルを解く問題を例にして、StandardML や ConcurrentClean などの関数型言語を紹介している。

(3) G.J. Myers **著、松尾正信訳、ソフトウェア・テストの技法、**1980 **年、近代科学社**

形式技術を利用したテスト技法は記述していないが、伝統的なソフトウェア工学的テスト技術を解説した教科書。

(4) **宮本勲著、ソフトウェア・エンジニアリング：現状と展望、産学社、**1982 **年**

ソフトウェア工学全般の網羅的参考書。

(5) Tom DeMarco, Timothy Lister **共著、日立ソフトウェアエンジニアリング生産性研究会訳、ピープルウェア、日経** BP **社、**1989 **年**

高い品質を求めることが高い生産性をもたらすことを発見した、ソフトウェア開発の人間的側面を述べた名著。

(6) **情報科学事典、岩波書店、**1990 **年**

情報科学の用語を解説した良書。

7.4 関連 URL

7.4.1 オブジェクト指向

(1)SRA のオブジェクト指向グループのホームページ

- http://www.sra.co.jp/smalltalk/
  オブジェクト指向関連のフリーソフトや他のオブジェクト指向関連ページへのリンクを持つ。

(2)Smalltalk ホームページ

- http://st-www.cs.uiuc.edu/
  イリノイ大学の Smalltalk ホームページ。Smalltalk に関するリソースは、ほとんどここに集まっている。

(3) オブジェクト指向関連情報ページ

- http://iamwww.unibe.ch/cgi-bin/ooinfo?info-page
  他のオブジェクト指向関連ページへのリンクを持つ。

7.4.2 デザインパターン

(1) パターン・ホームページ

- http://st-www.cs.uiuc.edu/users/patterns/patterns.html
  パターンに関連する最も役に立つページ。パターンのカタログや参考書あるいは関連ツールなどを探すことができる。

7.4.3 仕様記述

(1) 代数仕様記述言語 CafeOBJ のホームページ

- http://192.218.88.131/STC/CafeP/cafeproject.html
  仕様記述言語への入門マニュアルをダウンロードできる。

(2) 形式仕様ホームページ

- http://kielder.ncl.ac.uk/projects/FME/
  VDM, Z, RAISE and LOTOS などに関するホームページ。

7.4.4 ビジネスオブジェクト

(1) ビジネスオブジェクト情報へのリンクページ

- http://www.yy.cs.keio.ac.jp/~suzuki/object/bo.html