
目次

OMT分析

OMT分析作業概要

OMTモデル図関連表

OMT分析1 (オブジェクトモデル1)

〃 2 (オブジェクトモデル2)

〃 3 (動的モデル)

〃 4 (機能モデル)

〃 5 (操作の追加)

〃 6 (分析の繰り返し)

OMTシステム設計

OMTシステム設計1

〃 2

OMTオブジェクト設計

OMTオブジェクト設計1

〃 2

〃 3

実際のポイント

モデルと実装の関連

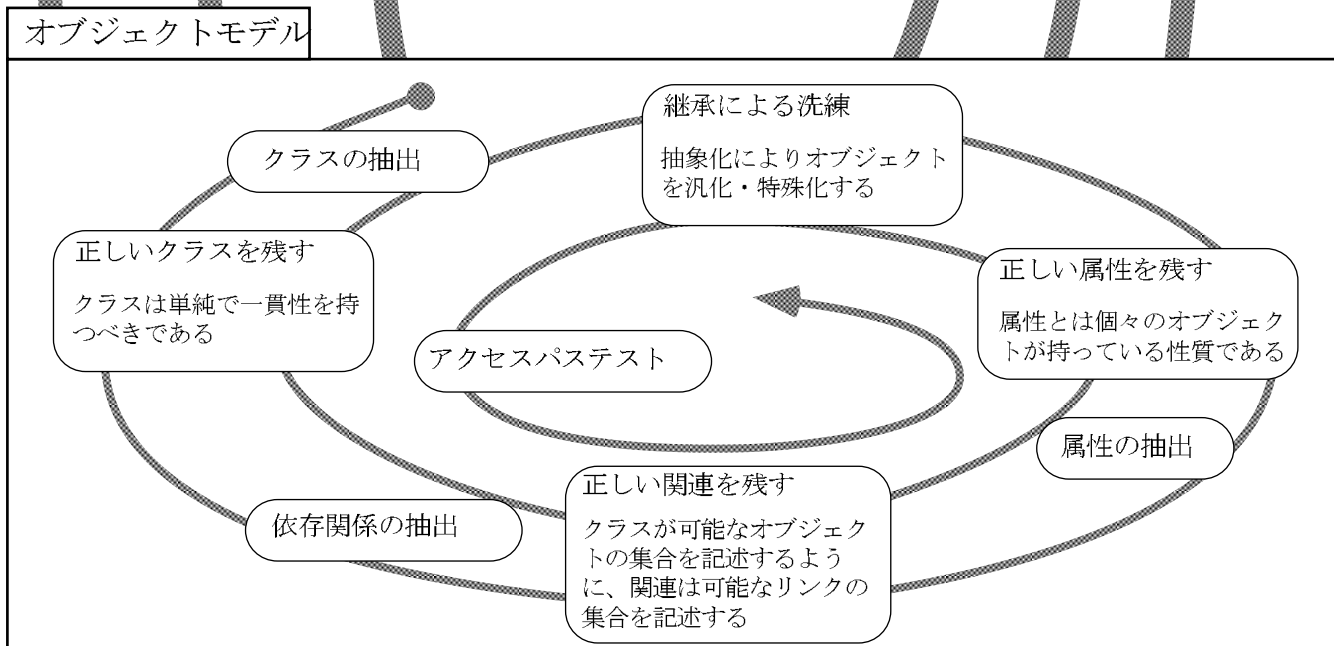
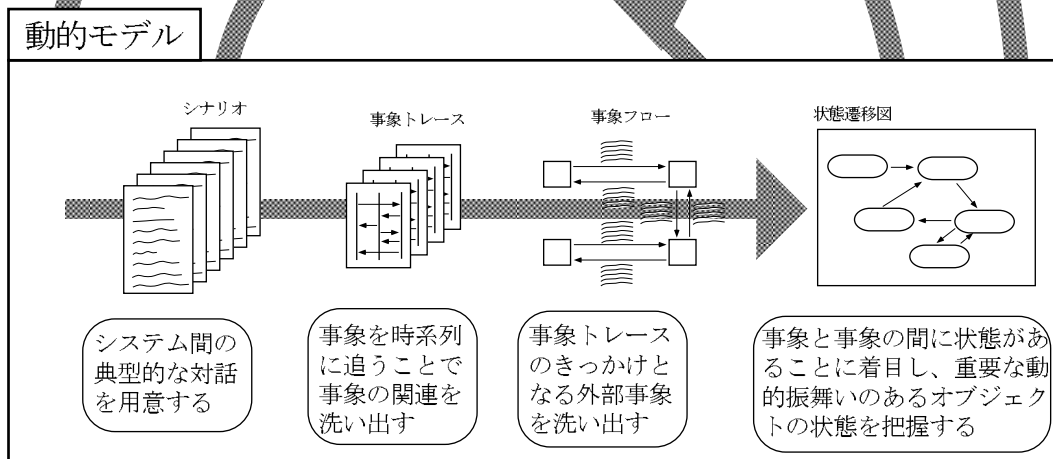
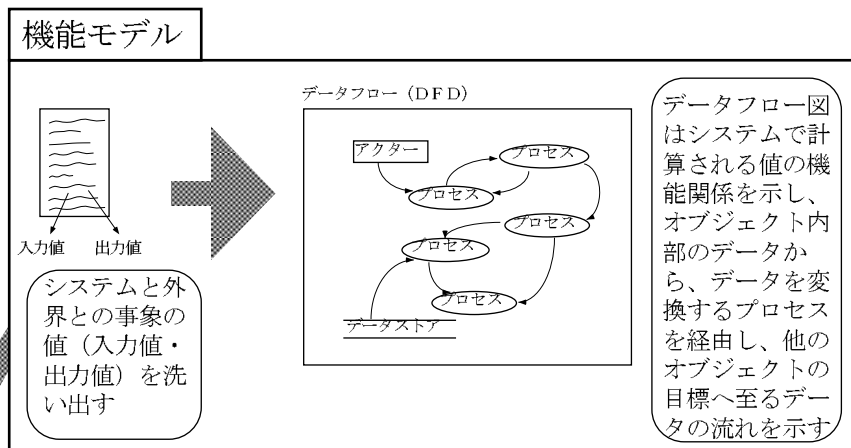
オブジェクト図の見方

状態遷移図の見方

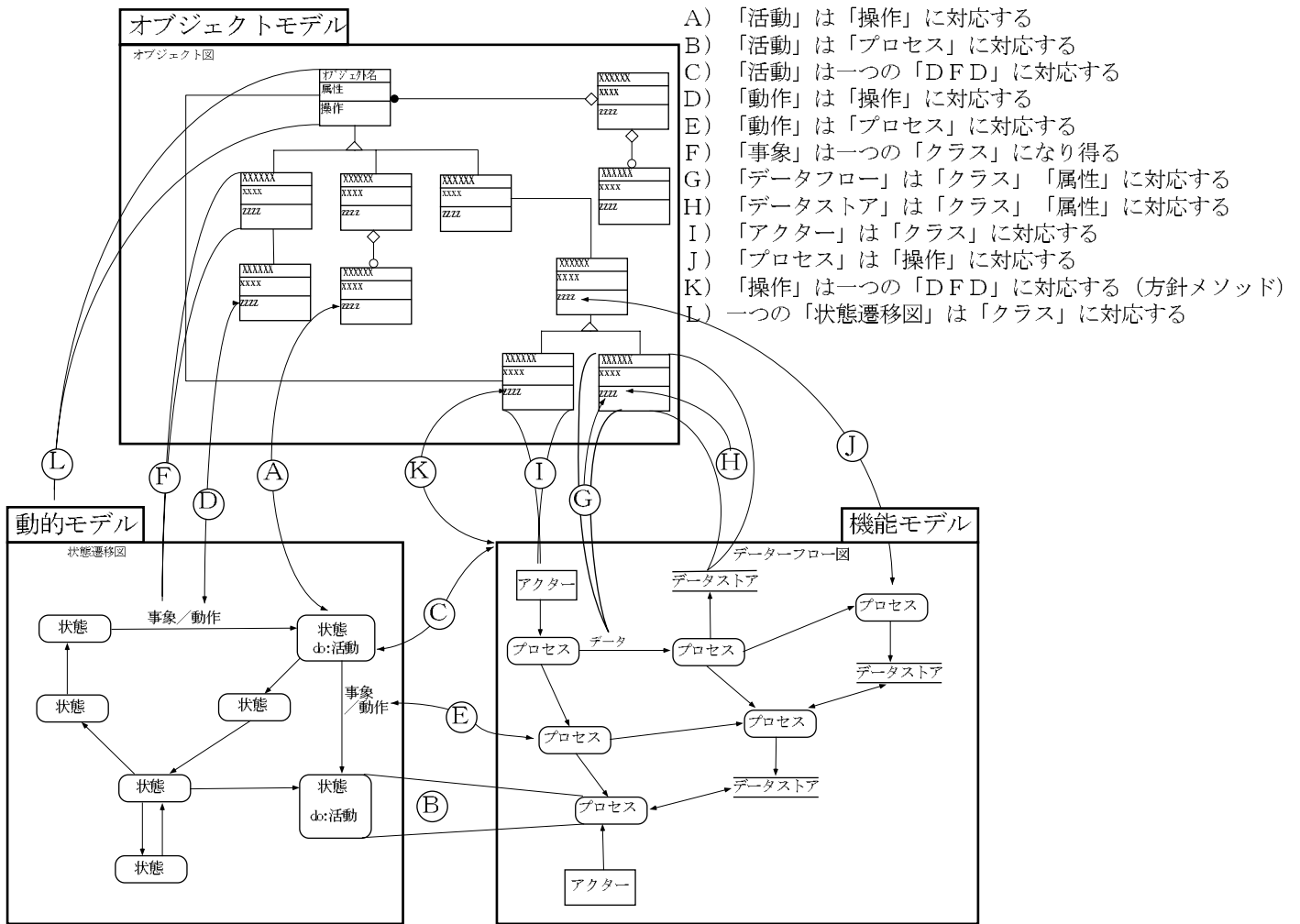
DFDの見方

DFDの構造

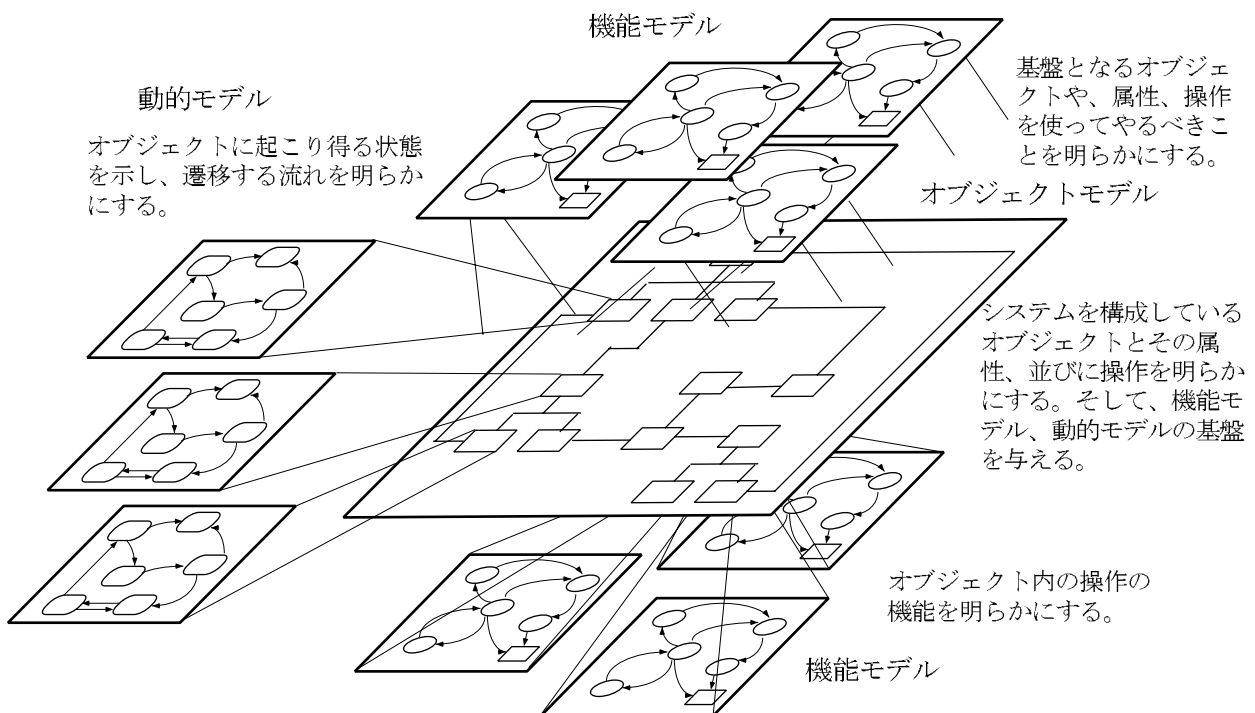
- 分析の位置付 要求を明らかにし、ソフトウェアの要求者と開発者の合意の基盤を与え、後の設計と実装の枠組を与える。
- モデル化 目的：実世界のシステムを理解できる形でモデル化すること。
内容：何をなすべきかを示していること。
成果：設計の準備段階として問題を理解すること。



モデル内の要素同志の関連



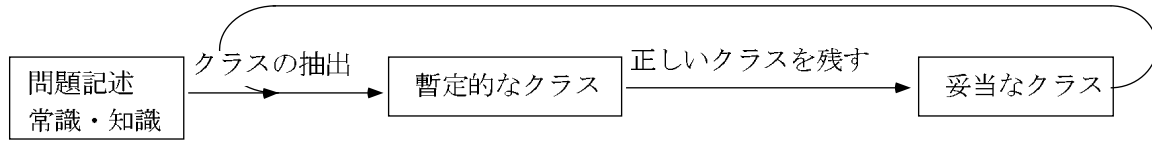
モデルの役割による関連



オブジェクトモデルの構築

1. オブジェクトクラスの識別

オブジェクトモデル = オブジェクトモデル図 + データ辞書



クラスの抽出

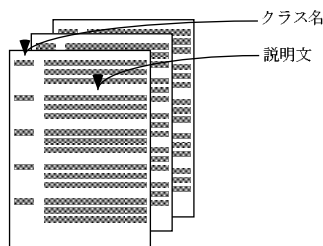
- クラス**
- ・物理的 家、従業員、機械...
 - ・概念的 軌道、座席予約...
- 方法**
- ・名詞的なものを拾い出す。
 - ・一般常識の中の構造的知識から拾い出す。
 - ・情報、知識、ノウハウを関連毎にまとめて洗い出す。
- 手がかかり**
- ・有形事象：形による分類
 - ・役割：同じオブジェクトでもその役割に着目
 - ・出来事：飛行、事故、演技、
 - ・相互作用：買い手売り手、男女の関係、
 - ・詳述：標準や定義の性質 (有形事象の細分類)
- 姿勢**
- ・最初から正しいものだけを選ぼうとせず、心に浮かぶ全てのクラスを書き出す。

以下の基準に沿って正しいクラスを残す。

- ・冗長 2つの情報が同じ情報を示しているようなら最もよく対象を表した名前を残す。(「客」は飛行機に乗る人を表しているが「乗客」とする方がより実体を表している)
- ・不適切 その問題にほとんど関係ないクラスは除去する。
- ・あいまい あいまいな境界を持っているものは見直す。
- ・属性 ある性質が独立した存在として重要な場合は属性ではなくクラスである。
- ・操作 ある名前が複数のオブジェクトに適用される操作を記述しており、それだけで独立に扱われていないものはクラスではない。
- ・役割 クラスの名前はクラスの本来の性質を反映すべきであり、そのクラスを持つ関連の中で演ずる役割を反映するものではない。
- ・実装要素 実世界とは無関係な実装のための要素は分析では排除する。

2. データ辞書の作成

各オブジェクトクラス毎のそれを正確に記述した簡単な文章を作る。



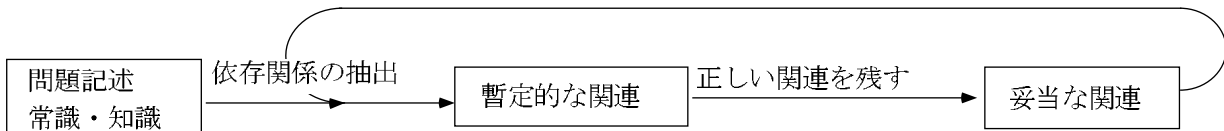
説明文の内容

- クラスに属するための条件
- クラスの使用法 (仮定、制限)
- 現在の問題におけるクラスの担当範囲

3. 関連の識別

2個以上のクラス間の依存関係は関連である。関連はクラス自身と同じ抽象レベルでクラス間の依存関係を表す。関連の実装は様々な方法で行われるので、分析時にその方法を固定してしまうのはよくない。

リンクは関連のインスタンスである。



依存関係の抽出

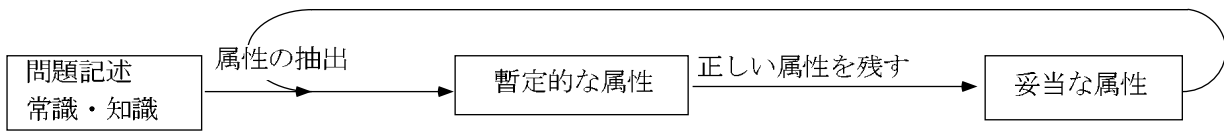
- 関連**
- ・関連はクラス間の依存関係を表す。
 - ・オブジェクトを値として持つ属性は依存関係を隠してしまう。
- 方法**
- ・問題記述から候補を抜き出し紙に書く、あまり初めのうちから絞り込んではいけません。
 - ・問題記述中の状態記述動詞句や動詞句からとれる。
 - ・実世界の知識、常識から洗い出す。
 - ・関連と集約を区別することに時間を費やしてはいけません。自然と思われる方を採用する。

以下の基準に沿って正しい関連を残す。

- ・不適切 問題領域外の関連は除去する。
- ・動作 関連は瞬間的な事象ではなく、構造的な性質を記述すべきである。
- ・3項関連 3つ以上の関連は2項関連に分解できたり、限定付関連で表現し直したりできる。
- ・派生関連 他の関連を用いて定義できる関連 (無駄な関連) は除去する。
- ・誤り名前 その状況が「どのように何故発生したのか」をいうのではなく「なんであるか」を示す。
- ・ロール名 適切な場所にロール名を付ける。
- ・限定関連 限定子を用いてより正確に関連を表現する。
- ・多重度 多重度を指定しない。ただし、分析中は変更がありうるので厳密な値を指定しようとしなくてよい。
- ・実装要素 実世界とは無関係な実装のための要素は分析では排除する。

4. 属性の識別

属性とは個々のオブジェクトが持っている性質のことである。



属性の抽出

属性

- ・派生属性は操作として指定するのではなく派生属性と分る形で属性として定義する。
- ・名前には意味のある名前を付ける。

方法

- ・属性は通常所有句に続く名詞に対比する。
- ・最も重要な属性を取り出す。
- ・特定の業務に直接関係する属性のみを考慮する。

姿勢

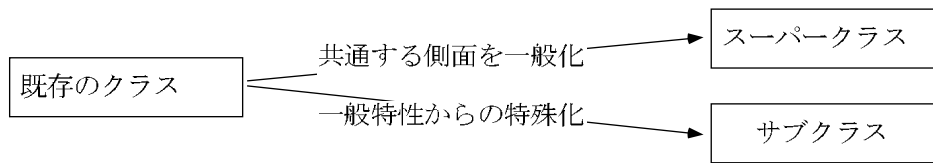
- ・属性を発見するのに深入りしすぎたはいけない。
- ・詳細は後で付け加えることができる。

以下の基準に沿って正しい属性を残す。

- ・オブジェクト 実体の独立した存在がその保持する値よりも重要であるならば、それはオブジェクトである。
- ・限定子 属性の持つ値がそのオブジェクトをユニークにするものであれば、それは限定子となる可能性がある。
- ・識別子 実装時に必要になるとと思われるオブジェクト識別子であっても分析段階では列記しない。
- ・リンク属性 ある性質がリンクの存在に依存するならば、その性質はリンク属性である。それは、多対多の関連において明らかである。もし属性が外部から見えないオブジェクトの内部状態を記述しているならば、その属性は分析では排除すべきである。
- ・内部値
- ・不調和 他の全ての属性と全く異なり、関係のないような属性の存在は2つの異なるクラスに分割すべきである。

5. 継承による洗練

共通の構造を共有するために継承を使ってクラスを組織化する。



一般化 (汎化)

方法

- ・よく似た属性、関連、操作をもつクラスを調べ、共通の性質を見つける。
- ・幾つかの属性さらにクラスさえも多少の再定義が必要になる場合もある。
- ・ある種の汎化は実世界に存在する分類学に基づいて自然に示される場合がある。
- ・可能ならば既存の概念をもとに継承を行うべきである。
- ・同じ関連名が実質上同じ意味で何箇所にも存在する場合には、それらの関連に含まれるクラス間で汎化を試みるといい。

多重継承

- ・多重継承は共有化を促進するために使用されるが、概念的及び実装上の複雑さも増加するので、本当に必要なときだけ使うようにする。

特殊化 (特化)

方法

- ・アプリケーション領域の知識から明らかになるときがある。
- ・様々な形容詞に結び付いた名詞を探す。(蛍光灯ランプ、白熱灯ランプ.. 固定式メニュー、ポップアップメニュー)
- ・アプリケーション領域に於て列挙できるような個別事例の集合が存在するとき、それらの事例は特殊化の源泉となる。

6. アクセスパスのテスト

オブジェクトモデルが意味ある結果を生み出すかどうかを見るために、オブジェクト図中をアクセスパスに沿ってたどってみる。

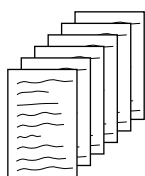
テスト項目

- ・ユニークな結果を生成するパスが存在するか?
- ・多重度が「多」の部分に対して、必要なときに特定の値を選び出す方法が存在するか?
- ・システムに対して問い合わせたい質問を考えてみよ。
- ・問題領域にとって有益な質問がモデルで答えられるか?
- ・実世界では単純なものが複雑に表されていないか?

動的モデルの構築

動的分析はまず事象すなわち外部から見える刺激と反応を見つけることから始める。次にその事象を各オブジェクトに割り付けていき事象系列を作成する。事象と事象の間には状態があることに着目し、各オブジェクト毎に状態遷移図としてまとめる。最後に、各オブジェクト間で交換されている事象が一致しているかを検証する。

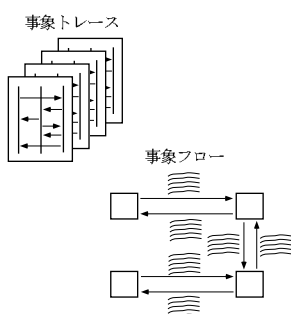
1. シナリオの用意



期待されるシステムの動作の感触を得るために、ユーザーとシステム間の典型的な対話を幾つか用意する。

シナリオは事象のシーケンスである。事象はシステム内のオブジェクトとユーザーセンサ、別タスクといった外部エージェントとの間で情報が交換された時に発生する。そこで交換された情報の値がその事象のパラメーターとなる。特に大きなシステムにおいてはこのシナリオは有効である。

2. 事象の識別



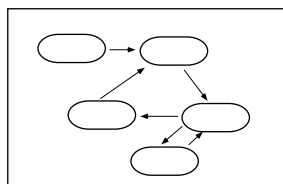
事象：あるオブジェクトからある情報を伝達する。
単に何かの発生を通知するだけのものもある。

全ての外部事象を識別するために、シナリオを検討する。

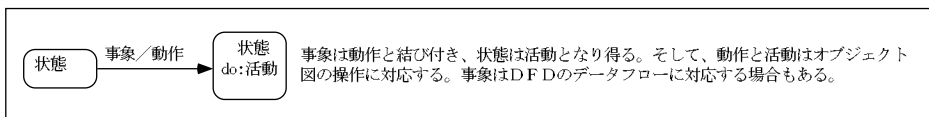
事象トレースを作成する。(事象を時系列に追ったもの)
事象を時系列で追うことで抜けている事象を洗い出す。

事象フローを作成する。(クラス間の事象の流れを表したもの)
事象トレースのきっかけとなる外部事象を洗い出す。
事象トレースの結果関連する事象をならび上げる。

3. 状態遷移図



重要な動的振舞いを持つオブジェクトクラスの各々について状態を用意し、各オブジェクトが送受信している事象を示す。基本的に事象と事象の間に状態が存在することに着目し、先に洗い出した事象を手がかりに状態を洗いだし状態遷移図として整理する。



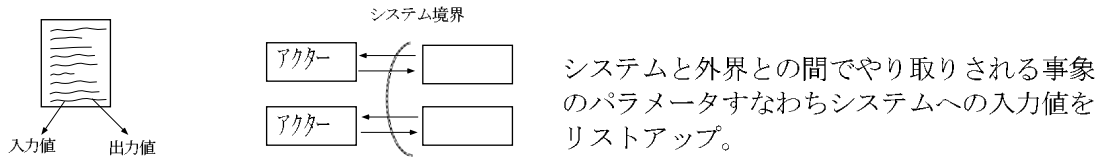
1. 事象トレースから1つのオブジェクトに着目し、それに影響を及ぼしている事象だけを考察する。
2. トレース図の縦1列に沿って入出力事象を状態図に経路に沿って配置する。
3. 状態図の中からループを見つける。もしループを記述している場合は状態を2つ(スタート・エンド、ループ中)に分ける。
4. 現在の状態図に他のシナリオをマージする。前のシナリオからの分岐を見つけ、新しい事象シーケンスを付加する。
5. 正常な考察をした後は境界条件・特殊ケースを追加する。

4. オブジェクト間の事象の整合性を取る。

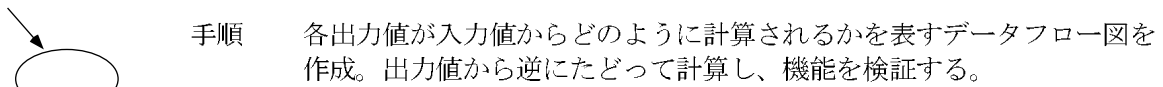
- 全ての事象は送信者と受信者を持つ。
- 前や後に別の「状態」を持たない「状態」はチェックが必要である。
- シナリオとの確認のためにシステム中をオブジェクトからオブジェクトへと入力事象の効果を追跡する。

機能モデルは計算順序や条件判断やオブジェクト構造を無視して、値に対する全ての可能な計算経路を示している。つまり、どの経路が実行されるのか、どの順序で起こるのかは示していない。機能モデルは機能と機能の依存関係を表すのに有効である。

1. 入力値・出力値の識別

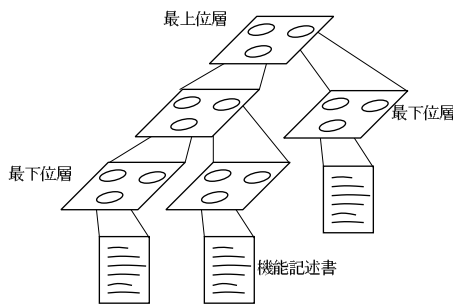


2. データフロー図の構築



手順 各出力値が入力値からどのように計算されるかを表すデータフロー図を作成。出力値から逆にたどって計算し、機能を検証する。

定義 データフロー図はシステムで計算される値の間の機能的な関係を示し、入力値、出力値、内部データストアが含まれる。データフロー図はオブジェクト内部のデータソースからデータを変換するプロセスを経由して、他のオブジェクト内部の目標へ至るデータの流れを示すグラフである。



プロセス プロセスはデータを変換する。プロセスは通常階層的に構成される。最上位層は単一のプロセスで構成されるか、入力集計、出力生成の各々一つのプロセスとして構成されるかのいずれかである。最下位のプロセスは純粋な機能（関数）である。

データフロー データフローは計算の中間データ値を表している。かならずしも実世界で重要な意味を持つわけではない。

アクター アクターは値を生産／消費することによりデータフローグラフを駆動する活動的なオブジェクトである。

データストア データストアは後でアクセスするためにデータを格納するデータフロー図中の受動的なオブジェクトである。

3. 機能の記述

データフロー図が十分に洗練されてきたら、各機能の説明を記述する。

どのように実装するかではなく何を行うかに焦点を絞る。記述は手続的であっても宣言的であってもよいが、宣言的な方が望ましい。

例) 「ソートして重複した値を削除する」 → 「入力リストの全ての値は必ず一度だけリストに表れ、出力リストは入力リストからの値だけを含み、出力リストの値は全て昇順になっている」

操作の仕様化。 データフロー図のプロセスは結局オブジェクトに対する操作として実装される。

- ある操作の外部仕様にはその操作の外部から見える変化のみを記述する。
- アクセス操作はオブジェクトの属性やリンクを読み書きする操作である。分析段階ではアクセス操作を列挙したり仕様化する必要はない。
- 自明でない操作は「問い合わせ」「動作」「活動」の3つのカテゴリーに分類できる。
- 「問い合わせ」とはオブジェクトの持つ外部状態に対して副作用をおよぼさない操作である。
- 「動作」とは対象オブジェクトからたどることが出来る他のオブジェクトに対して副作用を持つ変換である。動作は時間的に継続しないものであり、論理的に瞬間的なものである。ほとんどの場合動作は動的モデル中の事象に結び付いている。
- 「活動」とはオブジェクトによる操作で継続時間を持つものである。ほとんどの場合活動は動的モデル中の1つの状態に対応する。

4. オブジェクト間の制約の識別

制約とは入力値出力値の依存性に関係しないオブジェクト間の機能的な依存関係のことである。事前条件は入力値が満たしなればいけない制約。事後条件は出力値が満たすことを保証している制約である。その制約が成立する回数や条件を明示する。

5. 最適化基準の指定

最大化、最小化、最適化すべき値を指定する。

例) 平行処理を円滑に行うために口座がロックされる時間を最小にする。

操作の種類

従来のオブジェクト指向言語上での操作は以下の3つに対応する。

- ・属性、関連に対する問い合わせに対応する場合
- ・動的モデル中の事象に対応する場合 (利用者からATMへの取消のような)
- ・機能モデルの機能に対応する場合

操作の洗いだし

1. オブジェクトモデルから得られる操作

- ・属性値や関連リンクの読み書きがある。 分析段階ではオブジェクトモデル中には明示しない。

2. 事象から得られる操作

- ・オブジェクトに送られる各事象はそのオブジェクトの操作に対応する。
- ・事象は状態遷移図のラベルとして表現するのが適切であり、オブジェクトモデルにリストすべきでない。代りに事象と結び付く動作を操作として洗い出す。

3. 状態図の動作及び活動から得られる操作

- ・状態図中の動作と活動は機能として見ることが出来る。
- ・従ってオブジェクト図の操作とすべきである。

4. 機能から得られる操作

- ・データフロー図中の各機能は1つあるいは複数のオブジェクト上の1つの操作に対応する。しかし、オブジェクトモデル上には集約した形でリスト化すべきである。
- ・アクターからのデータフローもしくはアクターへのデータフローはオブジェクトによる操作もしくはオブジェクトに対する操作を表している。(アクター=オブジェクト)
- ・対象オブジェクト (どのオブジェクトの操作か?)
 - ・同じクラスのオブジェクトが入力と出力で表れれば、そのクラスが対象オブジェクトである。
 - ・出力がデータストアならばそのデータストアが対象オブジェクトである。
 - ・入力データストアならばそのデータストアが対象オブジェクトである。
 - ・データストアから入力またはデータストアへの出力を伴ったプロセスは2つのメソッドに対応することが多い。
 - ・入力が1つのオブジェクトで出力がそのオブジェクトの一部であったり、隣接するオブジェクトであるならば、その入力オブジェクトが、対象オブジェクトである。
 - ・出力オブジェクトが入力要素から生成されるならばそのプロセスはクラスメソッドを表している。
 - ・以上の規則に当てはまらなければ、対象オブジェクトは暗黙的であって、入力出力のどちらでもないことが多い。
 - ・あるプロセスの対象オブジェクトがそのプロセスを含むサブダイアグラム全体にとっての対象オブジェクトであることもある。

5. 買物メモとしての操作

- ・実世界での振舞い (そのオブジェクトが本来持つであろう振舞い) が操作を示唆している。

6. 操作の単純化

- ・類似した操作を調べてみる。
- ・バリエーションや特殊例を包括するように一つの操作を拡張できないか試してみる。
- ・なるべく継承を使って操作の数を減らそう (不自然でない範囲で)。
- ・各々の操作をクラス階層の正しいレベルに位置付ける。

分析モデルは完成までに何度かの繰り返しを必要とする。分析と設計との間には確固とした境界はないので、分析から設計に踏み込まないように注意する。

1. 分析モデルを洗練する

- ・初期の分析で放っておいた部分の詳細を追加しよう。
- ・間違った概念により、モデルがしっくりいかない場合もある。
- ・同じように見えるのにうまくまとめることが出来ない。構成要素がたくさんあるときは、たぶん、より一般的な概念を見落としている。
- ・論理的に異なる2つの側面を持つ物理的には一つのものとして存在するオブジェクトは2つのオブジェクトとして定義する。
- ・分析の初期の段階で有用と思えても今となつては無関係と考えられるオブジェクトや関連を削除しよう。
- ・二つの異なるオブジェクトの区別がモデルの残りの部分に影響を与えない場合は一つのオブジェクトにできる。
- ・例外、多くの特殊ケース、存在すべき対称性の欠如、関係のない属性群や操作群を持ったオブジェクトの場合は、モデルの構造をより適切な制約を表現するように再構成する。

2. 要求の再定義

分析が終ると、そのモデルはさまざまな要求の土台として機能するようになり、以後の議論の範囲を定義する。そのために最終的なモデルは要求者によって検証される必要がある。

最初の問題記述は、分析の間に見いだされた修正や理解を組み入れるために改定される必要がある。

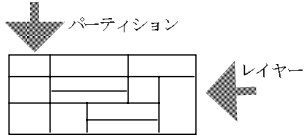
オブジェクトモデル・動的モデル・機能モデルの関係

- ・プロセスはオブジェクトに対する操作にあたる。
- ・各プロセスはオブジェクトのメソッドにより実装される。
- ・動的モデルは操作が実行される順序を示している。
- ・3つのモデルはメソッドの実装において一緒になる。
- ・機能モデルはメソッドへの指針である。
- ・動的モデルは各オブジェクトの状態、及び事象を受け取り、状態が変化した時に実行される操作を示している。
- ・機能モデルは動的モデルでは未定義の末端の動作や活動を示している。

システム設計＝要求をどのように実現するかの基本的アプローチを選択する。
(全体的構造とスタイル)

1. サブシステム分割

一つのサブシステムは問題のある側面に対する一貫した見方を定義する。そして、それが提供するサービス群によって識別される。サブシステム間のつながりは小さなインターフェースになっている。



サブシステムは上位・下位のレイヤー、並列的に独立しているパーティションに分けることによってその関係が分り易くなる。次にサブシステム間のデータフロー図を作成し、システム間の相互作用を減らすように調整する。

2. 並行性の識別

システム設計の1つの大きな目標はオブジェクトが並行に活動可能か、排他的に活動しなければならないのかを識別することである。本質的に並行なオブジェクトは別々なタスク、ハードウェアで実行可能である。

3. プロセッサやタスクへのサブシステムの割当

並行サブシステムはハードウェア単位に割り当てられなければならない。システム設計者がすべきことを以下に列挙する。

- ・必要な性能を満足するために必要な資源を見積もる。
- ・サブシステムの実装の判断（ハードウェア、ソフトウェア）
- ・性能を満足させるためにサブシステムをプロセッサに割り付ける、そして、プロセッサ間通信を最小にする。
- ・サブシステムを実装する物理的な単位の接続性を決定する。

4. データストアの管理

システムの内部及び外部のデータストアは定義されたインターフェイスを提供し、サブシステム間の明確な分割点になる。

正式のデータベースに格納すべき種類のデータの特徴を以下に指針として示す。

- ・複数のユーザーが適切なレベルの詳細さでアクセスする必要があるデータ
- ・DBMS コマンドによって効率的に管理されることが可能なデータ
- ・いろいろなハードウェアとオペレーティングシステムを持つ多数のプラットフォームにポートしなければならないデータ
- ・一つ以上のアプリケーションプログラムからアクセス可能でなければならないデータ

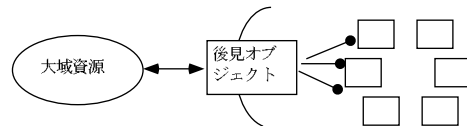
関係データベースに向いてなくてファイルにすべき種類のデータの特徴を指針として以下に示す。

- ・量的に膨大だがDBMSのもとで構造化するのが難しいデータ（ビットマップのようなグラフィクスデータ）
- ・量的に膨大かつ情報密度の低いデータ（アーカイブファイルとかデバッグ中のダンプとか履歴レコードのようなもの）
- ・データベースの中で集計される「生」データ
- ・短期間だけ保持されすぐに捨てられる揮発性のデータ

5. 大域資源の扱い

システム設計者は大域資源を洗い出し、それらへのアクセスを制御するためのメカニズムを決定しなければならない。そして、各大域資源はそれへのアクセスを制御する「後見オブジェクト」によって所有されなければならない。1つの後見オブジェクトは数個のオブジェクトを制御でき、資源へのアクセスは、後見オブジェクトを通して行う。

物理装置：プロセッサ、テープドライブ、通信装置～
空間：ディスクスペースや画面、マウスボタン～
論理的名前：オブジェクトID、ファイル名、クラス名～



6. ソフトウェア制御の実装方式の選択

システム設計者はシステム全体に対して制御スタイルを選択する。制御の流れは外部制御と内部制御の2つに分けられる。

外部制御

外部制御は外部に見えるシステム内のオブジェクト間の事象である。そして、以下の3つに分けられる。

- 手続き駆動システム：制御はプログラム内にあり、外部からの入力要求を出し、その入力が来た時に内部制御が再開される。
- 事象駆動のシステム：言語やOS、によって提供されるディスパッチャに制御が存在し、アプリケーションの手続きは事象に張り付けられ、対応する事象が発生したときに呼び出される。
- 並行システム：並行システムにおいては制御は幾つかの独立したオブジェクトに存在し、各々は別のタスクである。現在はまだ研究段階である。

内部制御

内部制御はプロセス内での制御の流れである。

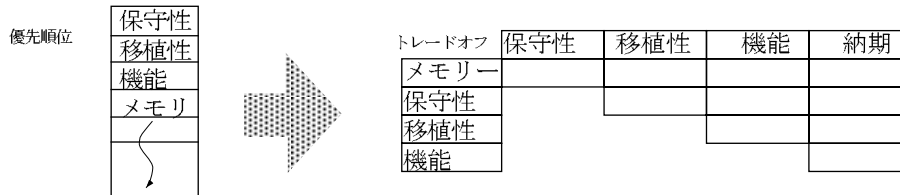
7. 境界条件の扱い

システム設計者は初期化、終了、障害などの境界条件を考慮しなければならない。

- 初期化 : システムは無活動の初期状態から安定状態へと移行しなければならない。しかし初期化の間はシステムの機能のサブセットしか得られないことに注意しなければならない。
- 終了 : 終了は多くの場合内部オブジェクトが単純に捨てられていくだけなので初期化より簡単である。
- 障害 : 障害は予定外のシステムの終了である。外部から起きる障害に対してはきちんとした見通しを立てる必要がある。また、バグによる障害の場合も、停止する前に環境をきれいにし、多くの障害情報を印刷して秩序ある終了を行うようにする。

8. トレードオフ優先順位の決定

システム設計者はしばしば、どちらも望ましいが両立し難い目的の間で選択を迫られる。そのトレードオフの判断の依りどころとなる優先順位を設定しなければならない。



9. 共通アーキテクチャの枠組

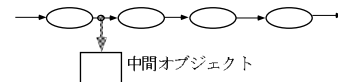
現在するシステムには共通の模範的なアーキテクチャの枠組がある。その枠組を出発点とすることで労力の削減が出来る。

システムの種類を以下に示す。

一括変換

入力から出力までの逐次的変換であり、入力は開始時に全て供給されそこから結果を計算する。

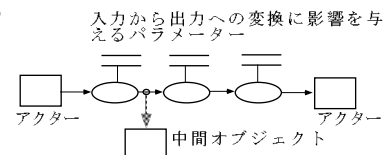
- 変換全体を幾つかのステージに分ける。
- 互いに前後する一対のステージ間のデータフローに対して中間オブジェクトを用意する。
- 操作がそのまま実装できるぐらい詳細になるまで順番にステージを展開しなさい。
- 最適化のために最終的な流れを再構成しなさい。



連続変換

その出力が、変化する入力に能動的に依存しており、周期的に更新されるべきシステムである。例としては信号処理、ウインドウシステム、プロセス監視など。

- システムに対するデータフロー図を作成する。
- 連続するステージ間に中間オブジェクトを定義する。
- 各ステージでインクリメンタルな変化を受け取る操作はどれなのかを区別する。
- 最適化のために中間オブジェクトを加える。



動的シミュレーション

実世界のオブジェクトをモデルかあるいは追跡するものである。

- オブジェクトモデルからアクター及び能動的な実世界オブジェクトを識別する。アクターは定期的に更新される属性を持つ。
- 離散事象を識別する。「電源を入れる」、「プレーキをかける」... 離散事象はオブジェクト上の操作として実装される。
- 連続的な依存関係を識別する。
- 一般にシミュレーションは適切な時間スケールのタイミングループによって駆動される。

実時間システム

実時間システム是对話型システムであるが、それに対する動作の時間制約が特に厳しいシステムである。

トランザクションマネージャ

主な機能が情報の格納とアクセスにあるデータベースシステムのことである。

- オブジェクトモデルを直接的にデータベースに結び付ける。
- 並行性の単位を決定する。必要に応じて新しいクラスを導入する。
- トランザクションの単位を決定する。一般にトランザクションは全体として成功か失敗の何れかである。
- トランザクションに対して並行な制御を設計する。

オブジェクト設計

システム設計で選択した戦略を実行して、詳細を詰め肉付けする。ここではアプリケーション領域の概念からコンピューター領域の概念にシフトする。

1. 3つのモデルを組み合わせる

オブジェクトモデルの「操作」を中心に3つのモデルの整合性をとる。「OMT分析（操作の追加）」を参照

2. アルゴリズムの設計

1) アルゴリズムの選択

「操作」の多くは機能モデルの仕様が既に満足のいくアルゴリズムになっている。自明ではないアルゴリズムは仕様が手続き的に与えられていない機能、もしくは、最適化を必要とする機能の2つである。

幾つかの候補から一つアルゴリズムを選ぶ場合の考慮点

・計算（処理）の複雑さ	出来るだけ複雑さを回避する方向で効率を求める。
・実装と理解の容易さ	効率の面でもあまり重要ではない操作については単純なアルゴリズムを採用する。
・柔軟性	ほとんどのプログラムは遅かれ早かれ機能拡張されることになる。
・オブジェクトモデルの調整	効率が悪い関係の場合に新たなオブジェクトを追加し複雑さを回避する。

2) データ構造の選択

アルゴリズムの選択は対象となるデータ構造の選択も含まれる。ここでは、効率的なアルゴリズムを可能にするようなデータ構造の形式を選択しなければならない。

3) 内部的なクラスと操作の定義

アルゴリズムを展開していく際に、複雑な操作はより単純なオブジェクトの上の低レベルの操作によって定義できる。また、中間的な結果を保存するための新しいクラスが必要になることもある。

4) 操作の割り当て責任

アルゴリズム中の複数の場所で幾つかのオブジェクトによって実行される自明なオブジェクトを持たない操作がある。以下に対象オブジェクトを割出す質問を示す。

- ・操作の対象となっているオブジェクトはないか
- ・変化を受けるオブジェクトはないか
- ・クラス間の関連に着目し星型に配置された中央のオブジェクトはないか
- ・実世界のオブジェクトだと仮定したときにどのオブジェクトを活性化したり、動かしたりするか

3. 設計の最適化

1) 効率的なアクセスのために冗長な関連を追加する。

- ・オブジェクトへのアクセスを改善するためにインデックスを付ける。
- ・操作を各々調べて情報を得るためにどの関連をたどるかを調べる。（関連は片方向か双方向か）
- ・その操作はどのくらいの頻度で呼び出されるか。
- ・関連ネットワークでの参照経路の「場合の数」はどのくらいか。（実データによる大体の目安）
- ・探索の目的となっているクラスの「的中」率はどのくらいか。（的中件数の全体に占める割合）

2) 効率化のための実行順序の再調整

効率化のためのデータ構造の最適化を調整したならば、次にアルゴリズム自身を最適化する。

3) 派生属性の保存による再計算の回避

他のデータから派生させることができる冗長なデータは再計算を避けるため値を保存しておく。

4. 制御の実装

設計者は動的モデルに示された状態-事象モデルを実装するための戦略を具体化する必要がある。動的モデルの実装には3つの基本的なアプローチがある。

1) プログラム中の位置による状態の表現

プログラム内での制御の位置が暗黙的にプログラムの状態を定める。

2) 状態機械エンジン

制御を実装する最も直接的な方法は状態機械を明示的に表現し実行する手段を用意することである。

3) 並行タスクによる制御

これは現実のオブジェクトが本来持っている並行性を保存するので最も汎用的な技法である。しかしサポートしている言語は少ない。

5. 継承の調整

クラスと操作の定義をうまく調整することによって、継承の利用を増やすことができる。

1) クラスと操作の再構成

別々のクラスの操作は似ているが完全に同一でない場合のほうが多い。この場合にもクラスや操作をほんの少し変えることで、それらの操作を継承された操作によって扱えるようにすることができる。

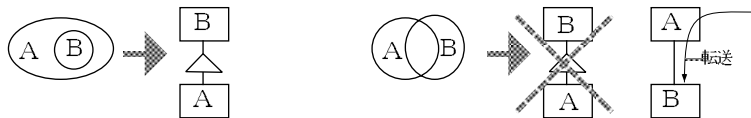
- 他の操作より引数の少ない操作があるかも知れない。引数を合わせる。
- 似たような属性、操作がクラスによって異なる名前で定義されているかも知れない。
- ある一群のクラスの中では定義されているが、その中の一部のクラスで定義されていないとした場合に、定義していないクラスの方で無効な操作として定義することで継承することが出来る。

2) 共通の振舞いの抽出

共通した振舞いが認められたならば、スーパークラス（抽象クラス）として抽出する。抽象クラスは一つのクラスにおいても操作の機能を一般的側面と特殊な側面に分離する意味でも有効である。また、似たような機能がケースによって微妙に違う場合は、その違いをサブクラスとして実現し、ケース毎に対応するサブクラスでインスタンスを生成するようにする。

3) 実装を共有するために委譲を利用する

AはBを継承するといった場合AはBの全ての機能を必要としている場合に限られる。Bの一部の機能だけをAで使いたい時に継承を使うのは好ましくない。その場合は委譲として関係ある操作を転送して活用する。



6. 関連の実装

1) 関連のたどられ方の分析

双方向の関連か、一方向の関連かを見極める。双方向の関連の方が将来の変更に耐えられる。

2) 片方向の関連

一方向にのみたどられる関連は属性として実現できる。

3) 双方向の関連

関連の実現には以下の3つの方法がある。

- 片方向だけを属性とし、逆方向にたどるときは検索する。
- 両方向共に属性として持つ。
- 関連オブジェクトを作成し、両方のポインタを持つ。

4) リンク属性

リンク属性の実現は以下の3つの方法がある。

- 1対1：どちらのオブジェクトの属性としてもいい。
- 1対n：n側のオブジェクトに属性を持つ。
- n対n：両側のオブジェクトとは別個の関連オブジェクトを実装する。

7. オブジェクトの表現

各々のクラスは他のクラスを使って定義されるが、最終的には全て組み込みの基本データ型を使って実現される必要がある。新しいクラスを定義する方がより柔軟であるが、参照が無駄な遠回りになってしまうことも多い。

```
基本データ型  chr
                int
                float
                ~
```

8. 物理的なパッケージ

1) 情報隠蔽

設計目標の1つはクラスを「ブラックボックス」として扱うことである。分析の段階では情報隠ぺいについては考慮しなかった、しかし、設計の段階では各々のクラスの公開インターフェイスを注意深く定義する必要がある。隠ぺいしたい属性はオブジェクトモデルにおいて明示的に記述する。また、メソッドに対してその有効範囲を制限するよう試みるべきである。操作から見える範囲を制限するための規則を以下に示す。

- ・各クラスに対して操作の実行及びそのクラスに関連した情報を提供する責任を割り当てる。
- ・他のクラスの属性を参照するときには操作を通じて行う。
- ・そのクラスに直接接続されていない関連をたどるのは避ける。
- ・できるだけ高い抽象レベルでインターフェイスを定義する。
- ・システムの境界上にある外部オブジェクトはインターフェイスクラスを定義することで隠ぺいしてしまう。
- ・メソッドの戻り値にメソッドを適用するのは、戻り値のクラスが呼び出し側にとって既に他のメソッドのサブライヤになっている場合に限る。

2) 実体の一貫性

クラス、操作、モジュールといった実体が一貫しているとは、この実体が一貫性のある計画のもとで構成され全ての部分が共通の目的にとって適合していることである。

メソッドは2つの種類に分けられる。1つの操作だけを行う「実装メソッド」とその実装メソッドをコントロールする「方針メソッド」である。方針と実装を分けることで実装メソッドの再利用性が向上する。

1つのクラスは一度に多くの目的を果たすべきではない。

3) モジュールの構成

分析及びシステム設計時に行ったモジュール分割を、オブジェクト設計において再度やり直す。オブジェクト図において関連の強いクラスをモジュールとしてまとめるのが適当である。

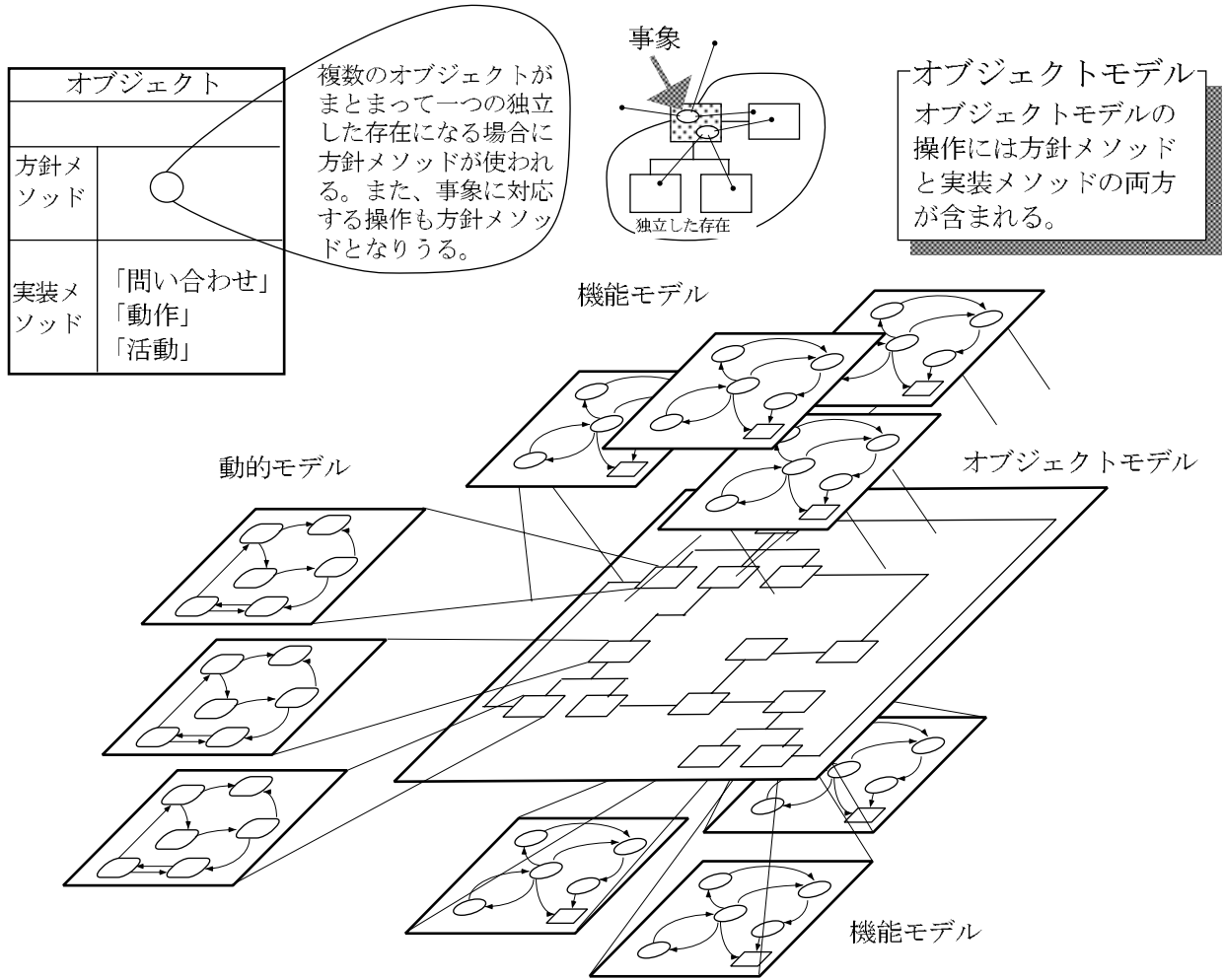
9. 設計における決定事項の文書化

設計文書は要求文書に対する文章の拡張になっているべきだ。

オブジェクトモデルではオブジェクト図とテキスト形式の説明書から出来ている。オブジェクト図には関連をたどる矢印や属性から他のオブジェクトへのポインタといった、実装上の決定事項を示すために新たな記法を追加するのが望ましい。

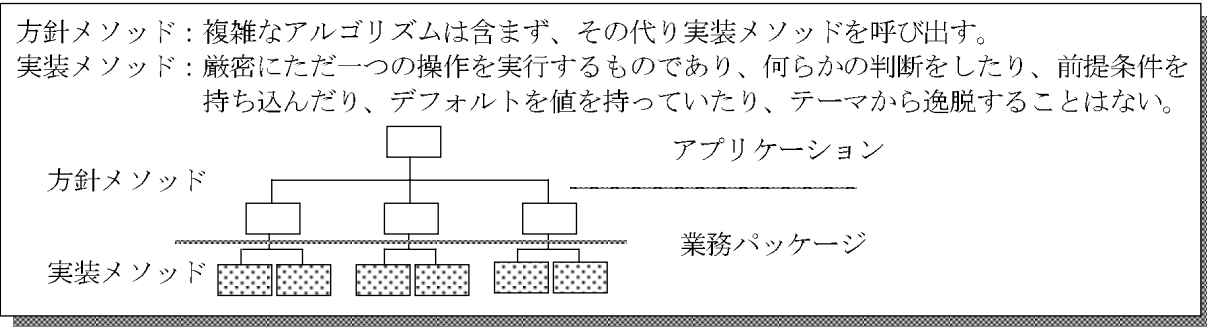
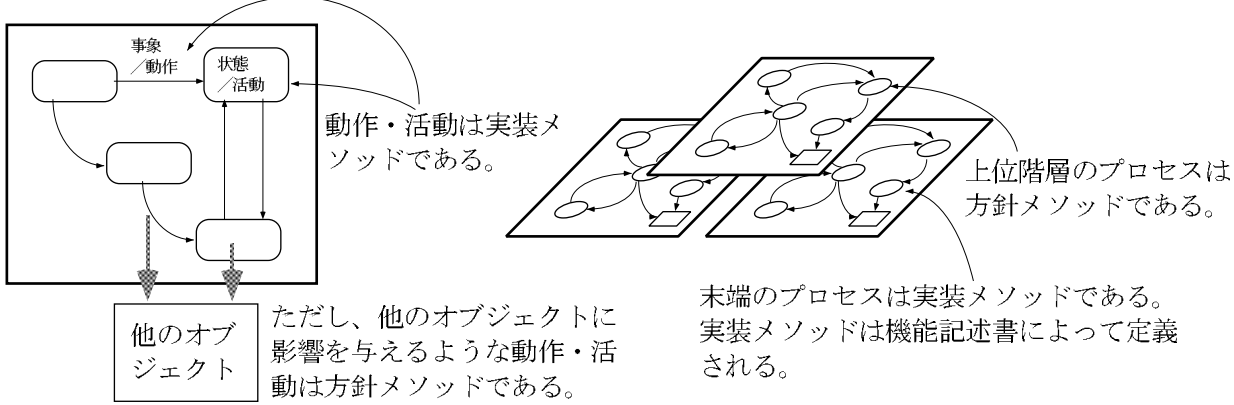
機能モデルは全ての操作についてその引数と戻り値と入出力の対応と副作用を記述することで、そのインターフェイスを仕様化する。

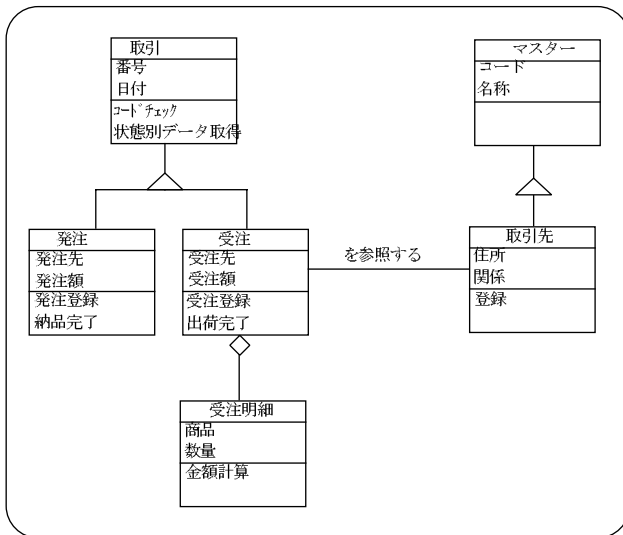
動的モデルは事象駆動あるいは並行動作方式で実装されるのであれば分析モデルの拡張で充分である。また、手続き駆動で実装される場合はアルゴリズムを表す文書が必要。



動的モデル
通常は状態・活動は実装メソッドである。

機能モデル
通常プロセスは方針メソッドと実装メソッドの両方が含まれる。





例) 説明

- 1) 取引オブジェクトは受注・発注オブジェクトの抽象操作を持つ。
- 2) 取引先オブジェクトは受注オブジェクトの受注先として参照される。
- 3) 受注は受注明細を集約しているので、受注明細のない受注はなりたたないと解釈する。
- 4) 取引オブジェクトはインスタンスを持たない抽象クラスである。

見方のポイント

オブジェクト図は対象システムの構成要素を示している。そこで、見るポイントとしてはオブジェクトの属性と操作から、そのオブジェクトの特徴をつかむようにする。また、オブジェクト同志の関連はこのシステムをどういう位置付けで見ているか、その分析者の位置を意識しながら見る。なかなかイメージがつかめないときは、そのオブジェクトがマスター的な性格なのか、トランザクショナルな性格なのかを考えながら一つ一つのオブジェクトを見ていく。関連の多重度が記述されているところはインスタンスとしての関連を類推するようにする。

分析のポイント

各オブジェクトの操作はそのオブジェクトを特徴付けるものに重点を置いて記述する。

分析の初期の段階での属性はそのオブジェクトにとって重要なものをまず抽出するようにする。無理に継承を作ろうとしなくてもよい、それよりも見やすい構造に気を配る方が大事である。

オブジェクトの抽出では画面や帳表、DBなどの実装のオブジェクトが気になるが、ここではあくまでもデータ構造を中心にモデル化する。

従来のシステムからのデータ構造を一旦捨てて、純粹に対象領域のオブジェクトとして捉えなおしてみる。そうすると新たな面を発見できる可能性が出てくる。

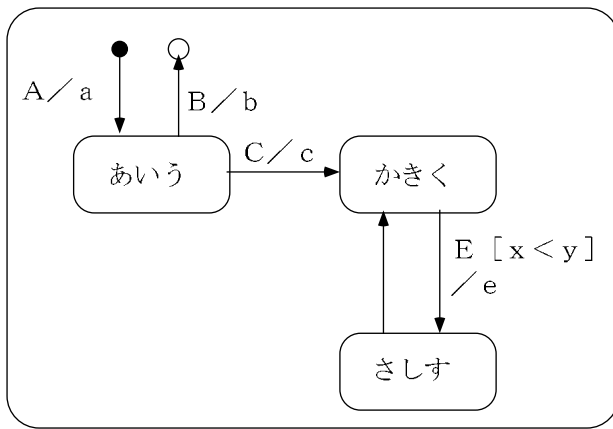
設計のポイント

オブジェクト図ではオブジェクト間の関連を表すために、オブジェクト内の属性として保持されるオブジェクトであっても、属性とはしていないので、そのようなオブジェクトの見極めが必要である。

オブジェクト図上ではカプセル化の考え方が明示されていないので、それを考慮した操作を考える。

オブジェクト同志があまりに密接な関連があると再利用しにくくなるので、極力関連は少なくする。

複数のオブジェクトがまとまって初めて独立した意味を持つオブジェクト群があるとき、そのオブジェクト群がサポートするサービスを受け付けるオブジェクトは明確にしておく。



例) 説明

- 1) このオブジェクトは事象 (A) 動作 (a) により発生する。
- 2) このオブジェクトは事象 (B) 動作 (b) により消滅する。
- 3) この状態「あいう」は事象 (C) 動作 (c) により状態「かきく」に遷移する。
- 4) 状態「かきく」は条件 $[x < y]$ が真のとき事象 (E) が発火し、動作 (e) により状態「さしす」に遷移する。

見方のポイント

- ・ある状態において起こり得る事象を明確にする。
- ・ある状態から次の状態への遷移を明確にする。
- ・事象が起こる条件を明確にする。
- ・事象と結び付いている動作を明らかにする。
- ・オブジェクトの発生、消滅の事象と条件を明らかにする。
- ・そのオブジェクトが持つ状態を明らかにする。

分析のポイント

「操作」抽出のきっかけを与える

分析段階においては上記表現によりそのオブジェクトに起こり得る状態、そして、状態を起こす事象を明らかにすることで、そのオブジェクトの操作のきっかけを与える。つまり、事象により状態遷移が起こるといことは、その遷移を起こす「動作」が必要であり、その「動作」はオブジェクト図上の「操作」に対応する。

状態の把握

実世界・実業務で行っている、または必要とされている状態を洗い出す。また、オブジェクトをその発生から消滅という観点から状態を取り出すことも重要である。

設計のポイント

状態はそのまま状態属性を表す

状態遷移図上に表れた状態はそのまま、そのオブジェクトが持つ属性（他のオブジェクトへのポインタも含む）の必要性を表している。

事象発生の条件はプログラム上での制御を明らかにする

事象に制約（条件）が書かれているときはプログラム設計時のコントロールの条件として参照される。

一貫性制御の抽出

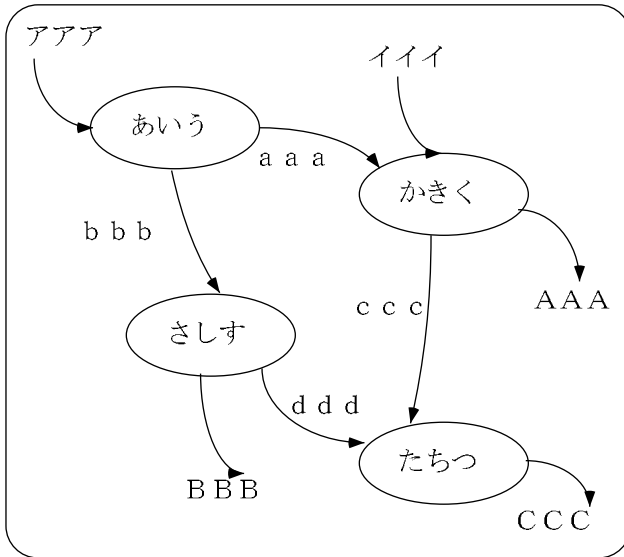
複数の状態遷移図上のある事象を中心に見ていくとき、一つの事象から複数のオブジェクトにおいて、幾つもの状態遷移が起こる場合がある。その場合はその事象により遷移を起こすオブジェクト間での一貫性制御の必要性を指し示している。

設計ミスの発見

設計されたシステムにおいて以下の条件に当てはまるとき、何らかの設計ミスの可能性が高い。

あるオブジェクトのとりうる状態が、状態遷移図上に描かれている以外の状態を発生し得る場合
ある状態において発生する事象が状態遷移図上に描かれている以外の事象を発生し得る場合

2.3 あいうえお



例) 説明

- 1) このプロセスは入力（アアア、イイイ）から出力（AAA、BBB、CCC）のデータフローへの変換を行う。
- 2) このプロセスは「あいう」「かきく」「さしす」「たちつ」の4つのプロセス（機能）から出来ている。
- 3) このプロセスが使う中間データは「a a a」「b b b」「c c c」「d d d」の4つがある。

見方のポイント

- ・ある機能（プロセス）がどのような機能から出来ているかを明らかにする。
- ・オブジェクトに対する機能がそのつながりの中で理解できる。
- ・オブジェクトとオブジェクトの関連を明らかにできる。
- ・入力事象から出力事象の間に発生するデータを明らかにする。
- ・ある事象の発生から終了までにやるべきことを明らかにする。
- ・第1レベルのDFDではシステムの外部事象（ビジネス事象）を明らかにする。

分析のポイント

ビジネス事象を中心とした機能分割

ビジネス事象に着目し、その発生から終了までの機能を明らかにすることで、より変更能耐えられるシステムを心がける。

設計のポイント

再利用可能な機能分割

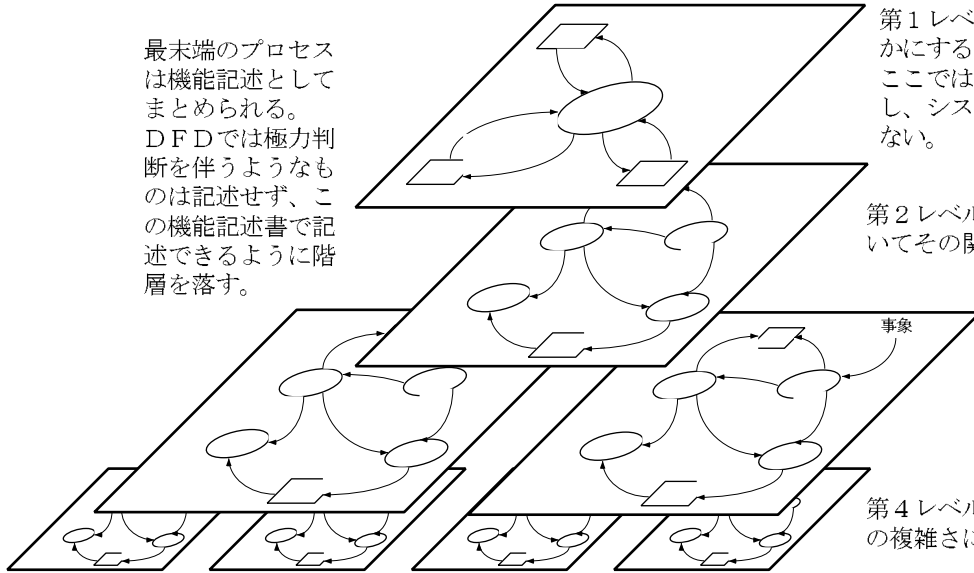
ビジネス事象に対応したプロセスは対象領域のビジネス用語で語られるような形で、機能分割にするよう心がける。同時に分割された機能は極力方針メソッドとなるようにする。すなわち、ビジネス上の作業に対応する機能は、対象となる作業が変わることで変更を受ける可能性がある。その場合その機能が方針メソッドで実現されていれば容易に変更能耐えられる。

方針メソッド・実装メソッドを明確にする。

プロセスはそのままオブジェクト図の操作と対応することが出来るが、そのプロセスが実装メソッドであれば最終プロセスとして機能記述できる。逆に複数のプロセスを含んでいる一つ上のプロセスは方針メソッドと考えられる。

方針メソッド：実装メソッド（操作）をコントロールする操作
実装メソッド：ただ一つの単純な機能を実行するメソッド

最末端のプロセスは機能記述としてまとめられる。DFDでは極力判断を伴うようなものは記述せず、この機能記述書で記述できるように階層を落す。



第1レベルは外界（システム外）と事象を明らかにする。ここではその事象すなわちビジネス事象に着目し、システムから発生する事象はここでは扱わない。

第2レベルは各ビジネス事象毎のプロセスについてその関連を明らかにする。

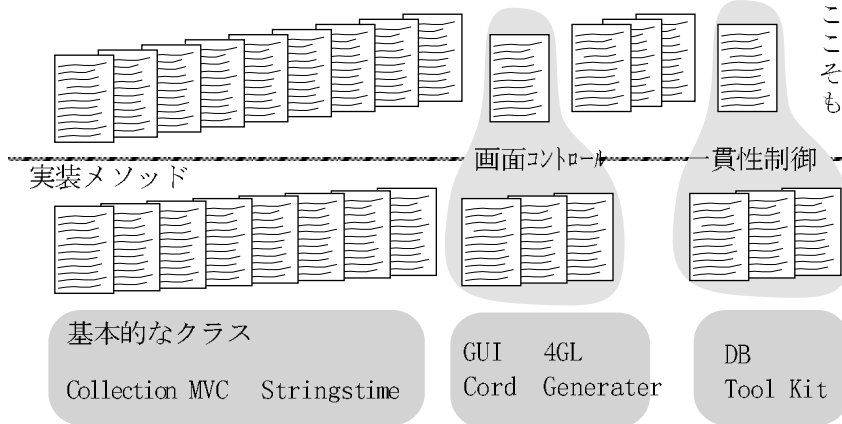
第3レベルは各ビジネス事象毎のプロセスを分解し、動のような機能（プロセス）から出来上がっているかを明らかにする。

第4レベルは第3レベルのプロセスの複雑さにより存在の有無が決る。

「動作」
「活動」
「問い合わせ」

一貫性制御
画面コントロール
業務ツール

ここでは判断やループが許されており、この機能記述書が方針メソッドになる。そしてこの機能記述書から呼び出されるものが実装メソッドとなる。



ここでは実装メソッドとして対象オブジェクトの状態遷移や、入力、表示、単純な「問い合わせ」を行う。

このレベルで実装される。

属性の構造以外での判断やループを使わないものを対象とする。

DFD構成要素

プロセス

プロセス分割を考えるとときはまず、以下のことを考慮に入れて分割する。

- このメソッドは「問い合わせ」「動作」「活動」のどれに当たるか？
- このメソッドは方針メソッドか、実装メソッドか？
- このメソッドはどのオブジェクトのメソッドか？

データフロー

データフローには以下の2つのデータが考えられる。

基本データ構造

データフローに基本データ構造が使われるときは、そのデータ構造は中間的なデータ構造を表している。従って、そのようなデータ構造は共通化するように考慮する。

オブジェクト

オブジェクトがデータフローとしてプロセスに渡されるときは、そのオブジェクトの「問い合わせ」操作を使うことを前提にしている。何故ならば、プロセスは入力データを出力データに変換するものであるからである。このようなオブジェクトはコンテナオブジェクトといえる。

データストア

基本的に受身のデータを保持することが目的のオブジェクトであるが永続的な保存を要求されるものをデータストアとする。従って通常のDBなどがそれに当たる。

アクター

動的なオブジェクトであるが通常のアプリケーションレベルで考えられるところでは、境界オブジェクトがそれに当たる。

例) 画面入力・表示、制御機器、バーコードなどの外部入力...