Requirement Development Alliance

Openthology新概念モデリング手法
（TFP分割手法について）

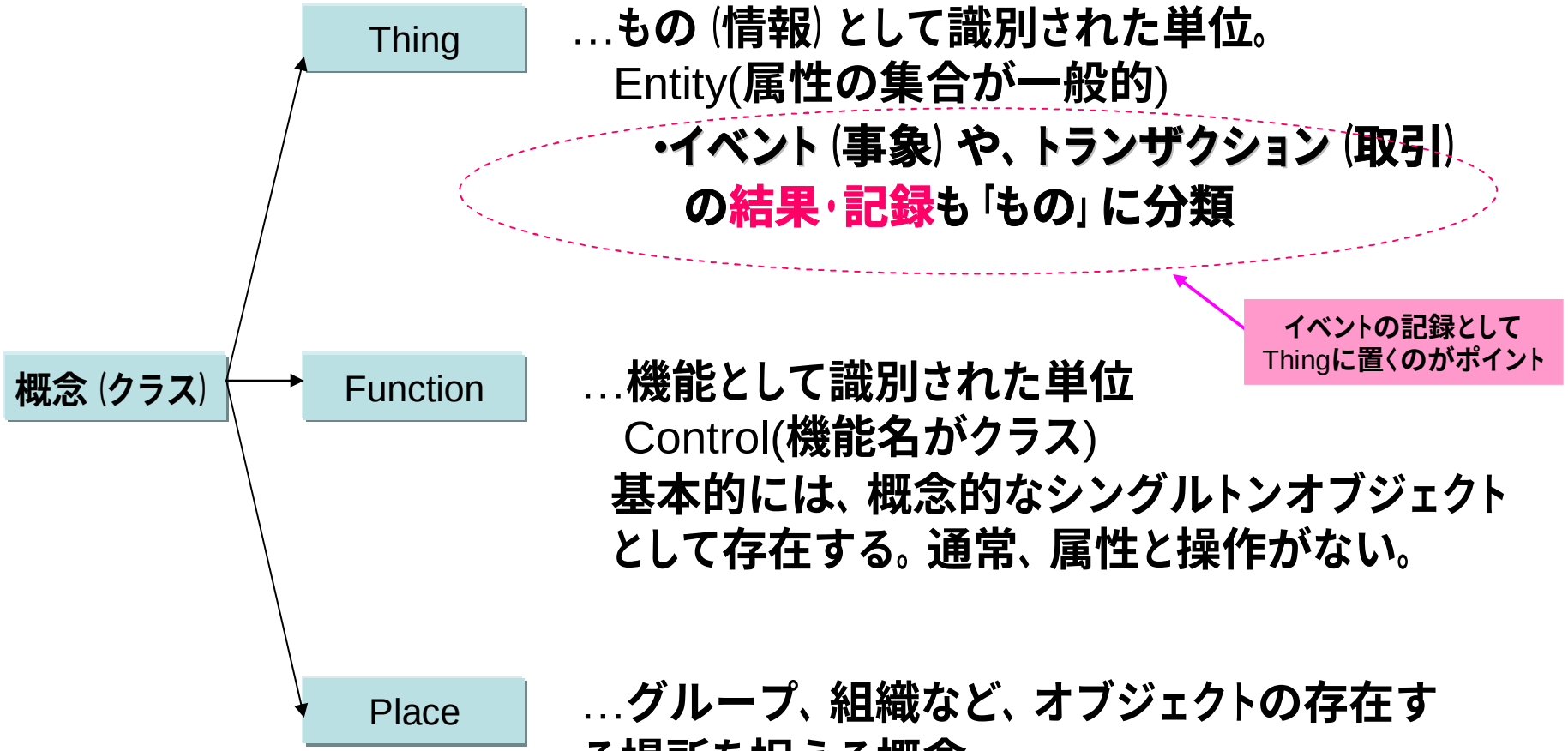
2006.1.27

豆蔵
萩本順三

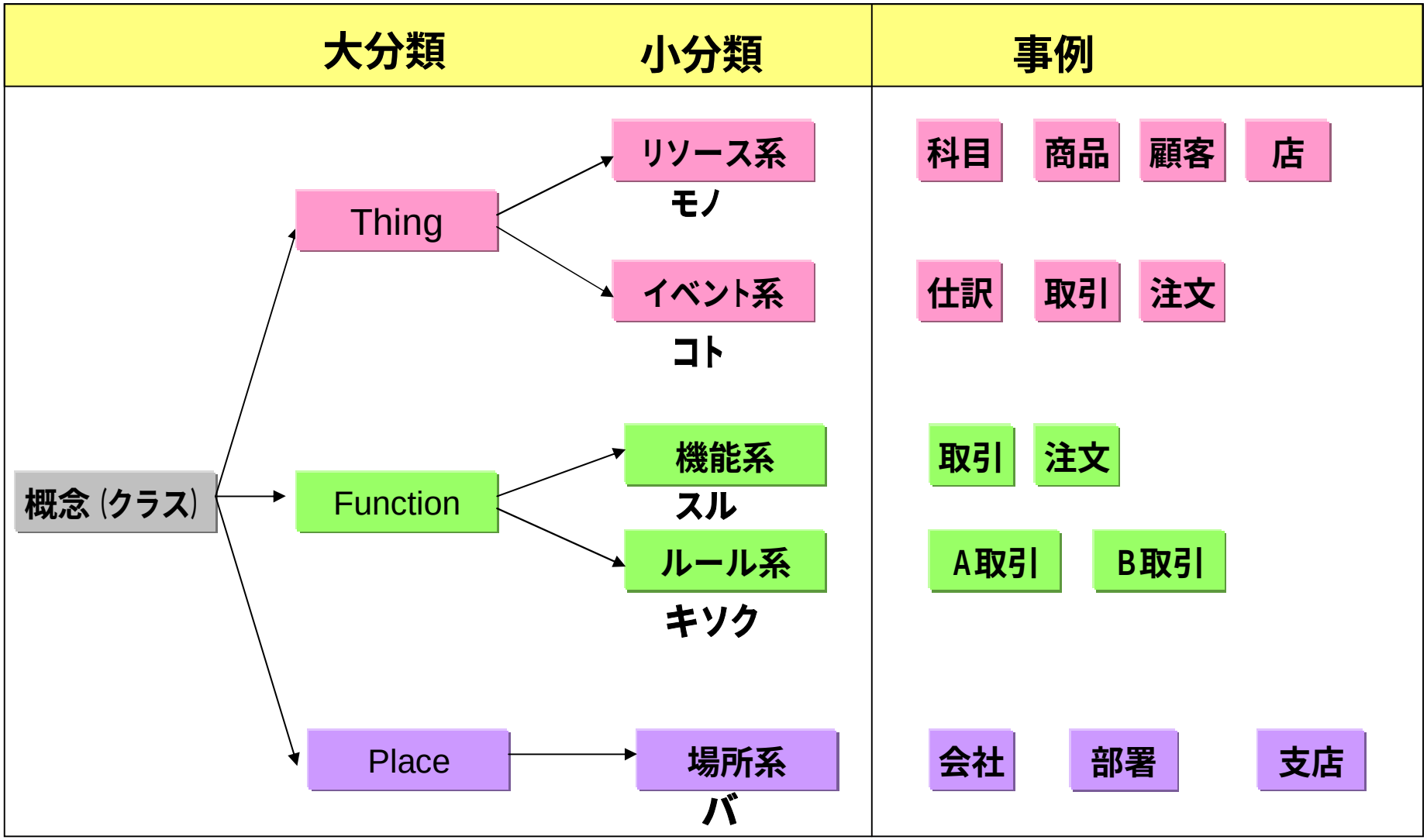
データモデルとオブジェクトモデルの課題

- データモデル
  - 長所
    - 物と場だけに着目する。所詮はデータ集合を静的に現すものなので、非常に解りやすく曖昧性がない。
  - 短所
    - 機能（業務）との関係が希薄。業務処理の概念化が異なる図(DFD)として表現するしかない。
- オブジェクト（クラス）モデル
  - 長所
    - 機能的なクラスを抽出するために、機能概念を含んだ自然な概念世界を表現できる。
  - 欠点
    - 概念の中に機能的なクラスが存在すると、機能の中に関連が隠されてしまったり、機能の入出力関係と、「もの」と「もの」との関係が混在することになり非常に曖昧になる。

Openthologyで採用提案する･新クラス分類



Openthologyで採用提案する
クラス分類



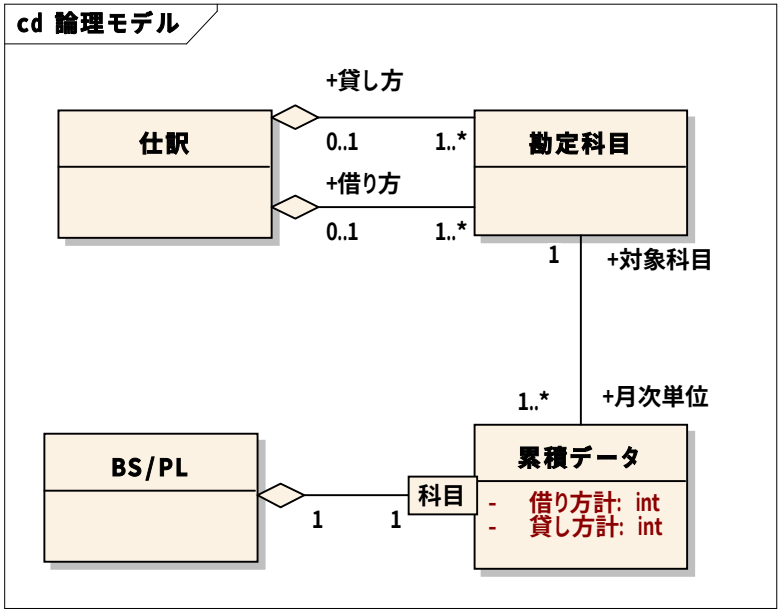
Openthologyで採用提案する
図の分類

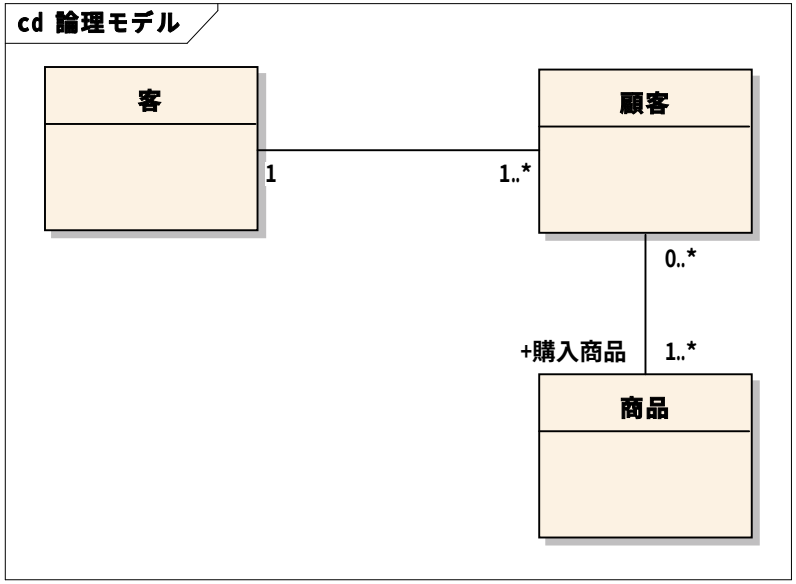
- 概念モデルを3つの図に分類する。
  - Thing図……ものとものとの関係構造図
  - Function図…..機能から見た構造図
  - Place図　……ものと機能を置くための場の構造図
- 3つの図に登場するオブジェクトに制限をかける。
  - Thing図には、Thingのみ登場させる。
  - Function図には、Function中心ではあるが、Thingを含めてよい。
  - Place図には、ThingとFucntionを含めてよい。
- これによって、オブジェクトモデリングの罠を回避できる

Openthologyで採用提案する図の分類と表記ルール

Thing図 には Function と Place を含めてはならない。

Thing図例 ピンク系統で表現。またはステレオタイプ<<Thing>>を使う



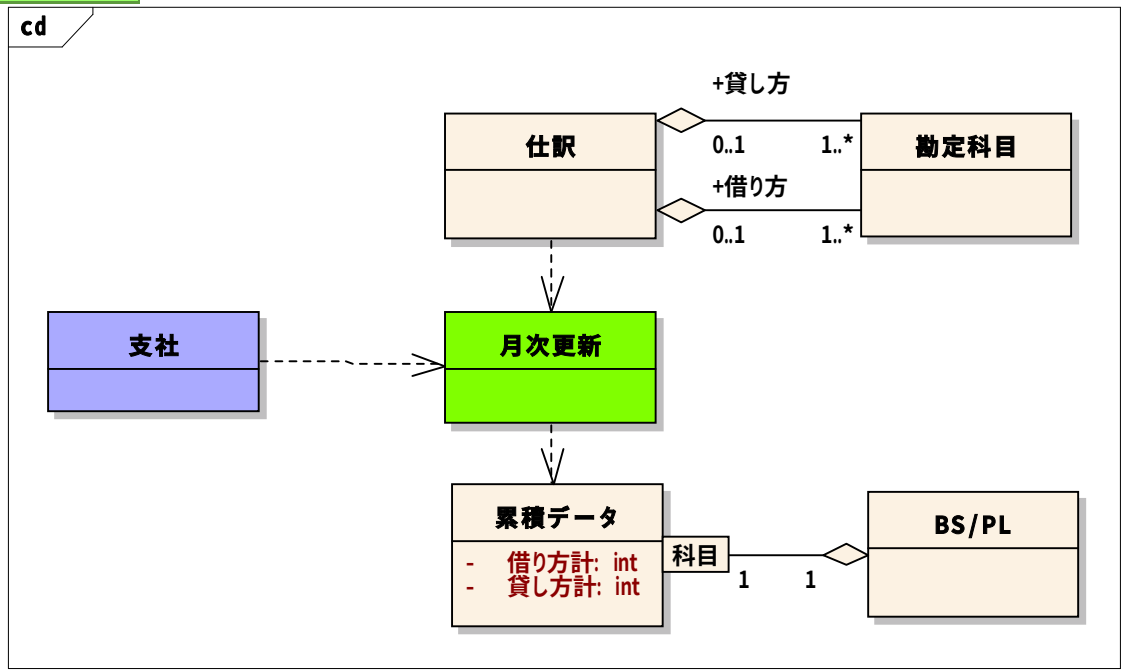


■システム開発段階におけるThing図の使い方
データモデリングのベースとなる。

Openthologyで採用提案する表記ルール

Function図 には Function のほか Thing を含めてよい

Function図例 グリーン系統で表現。またはステレオタイプ<<Function>>



■他のモデルとの関係

●Functionは、アクティビティ図のアクティビティの候補

■システム開発段階におけるThing図の使い方

●アーキテクチャモデルのベースとなる。

Place図 には Thing と Function を含めてよい

※注 Function図との違いは、「場」を中心としたモデルであるか、そうでないかの違いとなる。ただし、次ページの例で示すパッケージ表現やサブシステム表現を使う場合は無条件にPlace図として分類される。

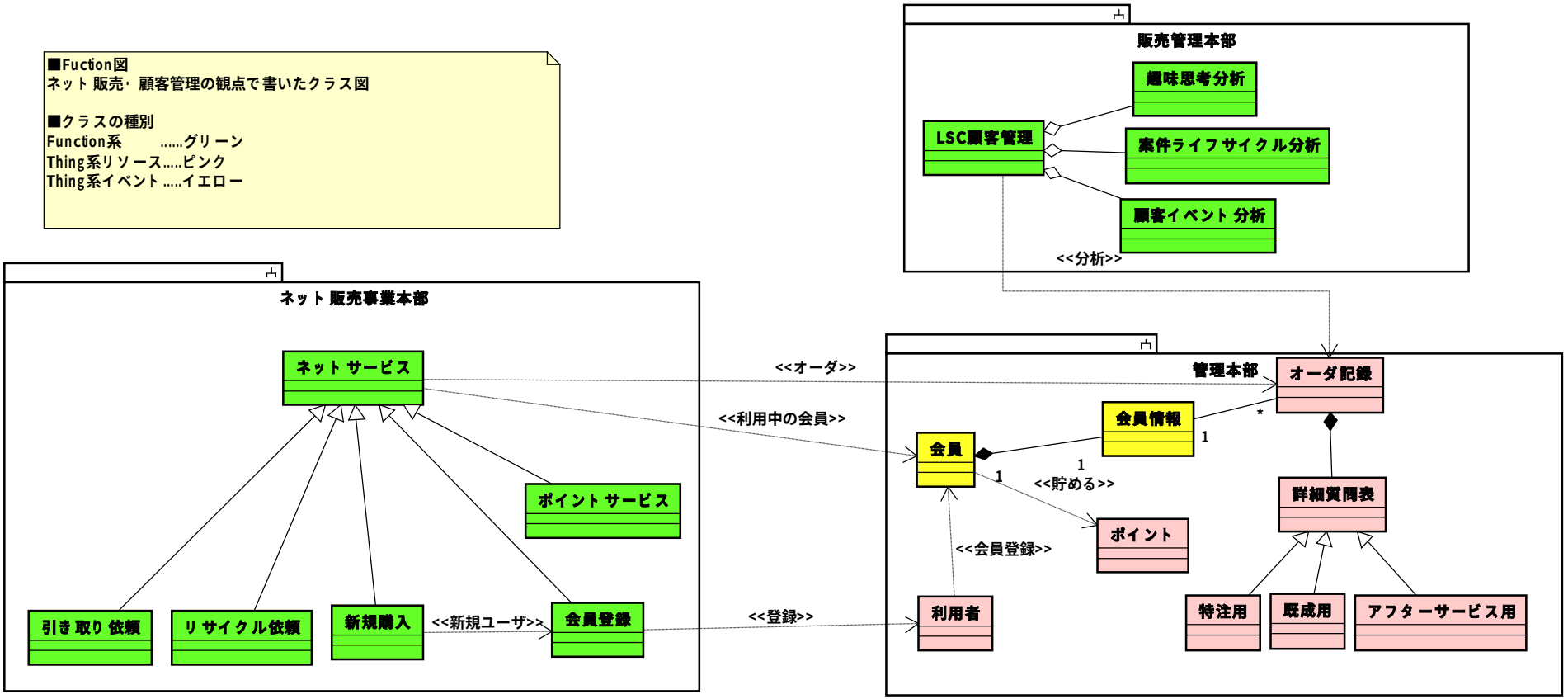
■システム開発段階におけるThing図の使い方

●データモデリングのベースとなる。

Openthologyで採用提案する表記ルール

Place図例

UMLサブシステムやパッケージでPlace図を示した例

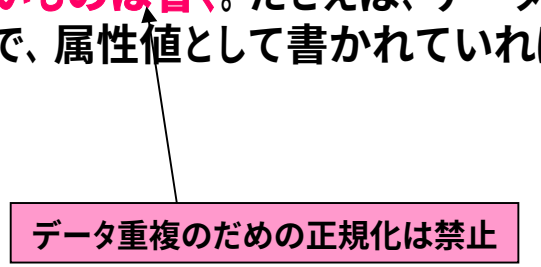


■システム開発段階におけるThing図の使い方

●データモデリングのベースとなる。

Openthologyの概念クラス図詳細

- ステレオタイプを下記のルールに従って、<<Thing/Resource>>と<<Thing/Event>>に分類してよい。
  - □<<Thing/Resource>>
    - 他のクラスから参照されないもの。
    - ビジネスにおけるライフサイクルが長い。
      - » 区分、種別のうち概念化しても意味のないものは省く。たとえば、データの重複を避けるためにリソース化しただけで、属性値として書かれていればよいもの。
  - □<<Thing/Event>>
    - 他のクラスを参照したりされたりする。
    - ビジネスにおけるライフサイクルが短い。



TFP導入理由：
（いままで、クラス図は、なぜ曖昧になっていたのか？）

■一般的な「もの」、「こと」分析

- モノ（「物」・「者」）
  - 物： 物理的・有形的な対象物
  - 者： 人の演じる役割（ロール），組織（機能や単位）
  - よそモノ： 外部システム

この部分は、あくまで事の記録を行うオブジェクトであり、実際の機能を表すものではない。

- コト（「事」象・「事」実）
  - 事象（事件）
    - ビジネスの重要なイベント（事象）やトランザクション（取引），それらの明細
  - 事実
    - モノやコトを記述した仕様，ビジネスプロセスやビジネスルール

若きエンジニアの寝言

まてよ！ではControlクラスは、明細同様の「コト」なのか？
正解：オブジェクト指向では、出来事も1つのインスタンスとして表す。
よって、Controlクラスはここでいうコトである。
でも設計上Controlクラスはシングルトンにすることが多いよね？？？
大混乱！

オブジェクトモデルの曖昧さ（例）

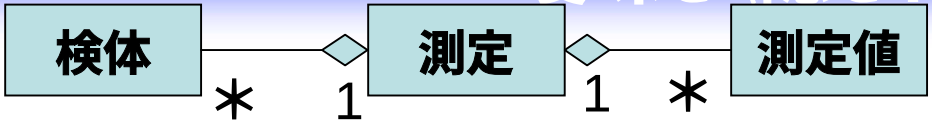
検体（もの）は測定（機能）によって測定値（もの）が生成される。

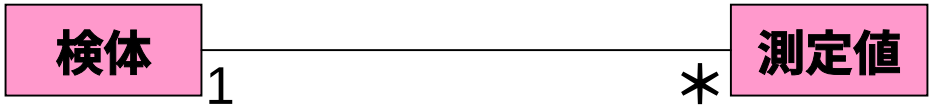
顧客登録（機能）によって管理された客になる。

仕訳（もの）は月次更新（機能）によって、累積データ（もの）となる。

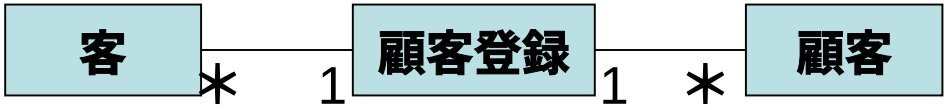
＊ここでの「測定」「顧客登録」「月次更新」は機能をオブジェクト化したもので、基本的にシングルトンと考えた場合のモデルを示しています。このような考え方もできますし、測定という出来事をオブジェクト化することもできます。
　通常オブジェクト指向の本来の考え方は、後者になりますが、最近のMVCのCでは、前者と捉えるほうがシンプルな設計になるのでそのようにしています。さて、ビジネスモデリングの世界では、どちらで捉える方が効果的なのでしょうか？
何れにせよ、オブジェクト指向によるモデリングでは、このような事で混乱を招く事が多い。

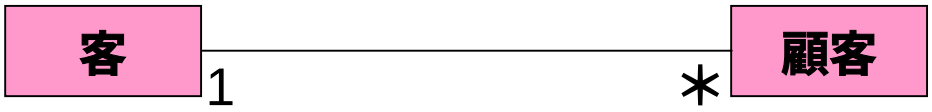
オブジェクトモデルのControlクラスによる曖昧さ(隠された関連)



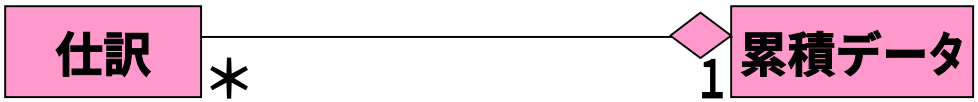


検体からはいくつかの測定値が取得される。





現状は、客一人に複数の顧客管理がされている。(AsIsの問題)



仕訳は累積データに集約される。

＊もし機能をオブジェクト化したようなクラスを登場させてしまうと、データオブジェクトの普遍的な関連が、機能によって分断され、隠されることになります。このような機能クラスが登場してしまうとモデルが非常に曖昧になってしまいます。しかし、機能もビジネスにとって重要です。何らかの形でクラス図に登場させるべきでしょう。

要求開発

- Openthology ver1.0リリース
  - まずは書籍にて3月初旬に発売

- Version 1.0では、次のような方法論の強化を行った。
  - プロセスの強化（テーラリング可能なプロセス）
    - プロセスキャビネットの導入
    - プロセスセルの導入
  - 戦略モデルの強化（ＢＳＣ戦略マップ）
    - IT貢献度マップ
    - プロジェクト実施計画書へのリバース
  - 概念モデルの強化
    - TFP分割手法の導入

TFP分割手法の革新性

TFP分割の最初の発想は、Thing,Function,Placeというクラスの類別化に始まる。しかし、これだけでは従来のMVCやBCEなどの3分割とあまり代わりがない。TFP分割手法の特筆すべき点は、このクラスの類別化に加えて、クラス図を3つのタイプに分類したことにある。そしてそれぞれのタイプ(Thing図、Function図、Place図)に特徴を持たるため、それぞれの図に登場できるクラスのタイプ(Thing,Function,Place)を制限したことである。

これによって、Thing図は最も普遍的な概念構造を表すことになり、Function図、Place図は、具体的なビジネス価値の構造を示すことができる。

1．Thing図
　　本質概念の整理

2．Function図
　機能的な概念で本質概念の用途を整理

3．Place図
　概念の価値を与える場の整理

- このような類別化でオブジェクトモデルのビジネス概念の普遍性追求に、ビジネス価値追求を加えることができるようになる。また、Thing図は、データモデルとも相性がよく、さらにFunction図やPlace図によって、SOAを含むシステムアーキテクチャモデルへ誘うことができる。
- このように、単にクラスの類別化がTFP分割モデルの特徴ではなく3タイプの図を作ることがTFP分割手法の革新性といえる。

プロセスキャビネットの革新性：
仮説と検証のプロセス

●仮説検証的な反復計画：フェーズの側面

複雑な問題や大きな問題ほど、計画段階で作業にリアルなイメージを持てずに作業スケジュールを組み、その後は、無駄があっても突き進むことが多い。本来、計画を立てるためには、その前フェーズの活動が必要となる。その事前活動によって、現実味を持った計画を立てることができるようになるのである。このように、仮説を立て、実施して、その実施経験の中から、次の計画を立てるような流れ、それが仮説検証的な反復計画なのである。Openthologyの「準備」「立案」「デザイン」「シフト」のフェーズ名は、まさにこの「仮説と検証」の過程から生まれたものである。

●仮説検証型PDCAサイクル：作業領域の側面

一方、作業領域の側面では、作業をPDCAにマッピングすることで仮説検証を実現している。従来のプロセスでは、プロセス自体にPDCAの概念がマッピングされておらず、プロセス利用者が意識の中でPDCAに落とし込む必要があった。プロセスキャビネットでは、活動（アクティビティ）を最初からPDCAに対応付けているので、自然にPDCAによって反復を回す習慣がつけられるのである（図2.3参照）。

複雑な問題を解決するための作業についてプロセス化したものの多くは、今後このようなデザインとなるだろう。その意味では、プロセスキャビネットは、要求開発プロセスをつきつめた結果、生まれたものといえる。

プロセスセルの革新性：
自立分散活動可能なプロセス概念

RUPを代表する反復開発プロセスで、筆者が危惧していた事は、反復プロセスの中に定義されているアクティビティ（活動）の中には、逐次作業的の反復ではなく、並行作業的な側面が多いという点である。

しかし反復というと逐次的作業を反復するというイメージが強く誤解を招いてしまうし、プロセスも逐次的作業の反復としてデザインされている。本来反復プロセスで達成すべき事は、逐次的または並行的に進めることができる作業単位を明確化とその作業単位の独立性をできるかぎり保つようにすることである。

プロセスセルでは、この反復開発における達成目標を取り入れるために、ＰＤＣＡにセットされたアクティビティ（活動）群と成果物、そして入出力を、活動目的とともにカプセル化している。
これにより、従来のどのプロセスよりも、自立性と並行動作性が高まると同時に、プロセス概念がとても理解しやすくなり、ミニマムプロセスをＰＤＣＡとして回すという教育的なアプローチにより、利用者のプロセススキルの向上も期待できるのである。

プロセスセルの導入は、従来のプロセスの考え方を拡張し、反復作業と並行作業をよりやりやすくする概念を新たに提案しているのである。

このようにプロセスキャビネットのポリシーに支えられたプロセスセルによって、従来のRUPなどの反復開発プロセスの概念を大幅に超えるものである。今後、システム開発のプロセスでも、これらプロセス概念を応用することを検討している。

要求開発プロセスは、誕生したばかりであり、今後要求開発アライアンスの中で、更なる洗練化を図る予定である。たとえば、プロセスセルの中に作業ロールを組み込み、作業ロールのスキルセットを明確化するという計画も予定されている。