

Rational[®]
the software development company

UMLを用いたコンポーネント設計

日本ラショナルソフトウェア株式会社

藤井 智弘

構造的なものの表記

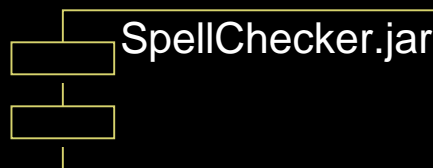
インターフェース



ISpellCheck

サービスを表現する操作仕様の集合(実装ではない)

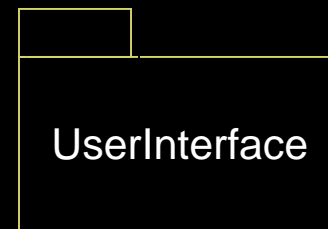
コンポーネント



システム内の物理的かつ交換可能なパッケージング部分で、インターフェースを実現する(実装する)。

Ex.) dll, exe, jar

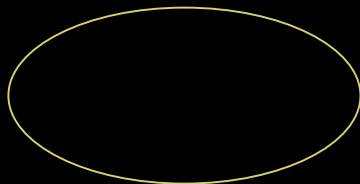
パッケージ



要素をグループ化するもの。純粋に概念的なもの



ユースケース



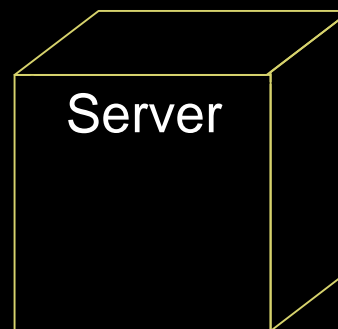
システムが実行する一連のアクションの組み合わせ。観察可能な“価値”を結果として産み出す。

コラボレーション



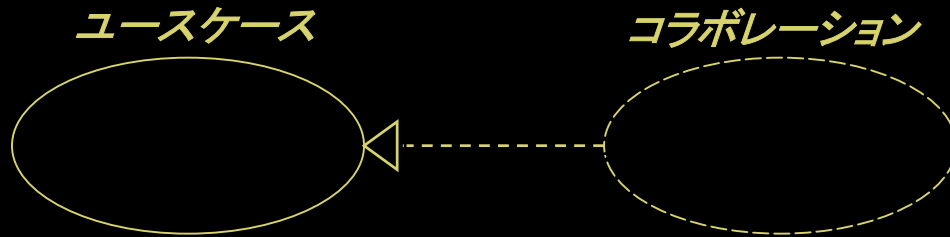
相互作用を定義する。オブジェクト等の協調的振る舞いを提供する

ノード



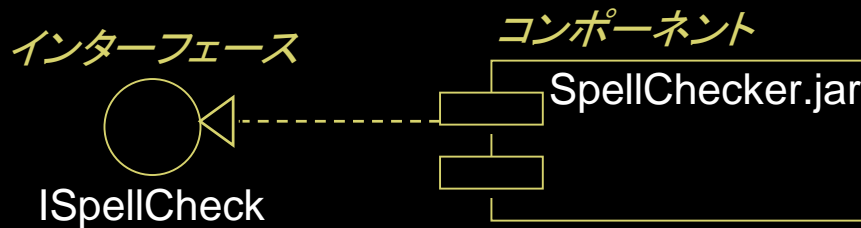
物理的な(通常)計算機資源

UMLの中でコンポーネントに至る



“ユースケースの実現”

情報の追跡可能性を実装



サービスを表現する操作仕様の集合(実装ではない)

システム内の物理的かつ交換可能なパッケージング部分で、インターフェースを実現する(実装する)。

Ex.) dll, exe, jar

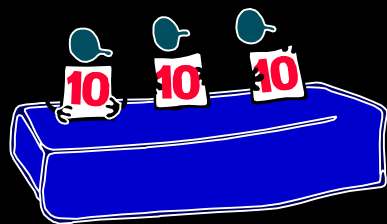
“インターフェースの実現”

インターフェースと実装を分離することによる“部品度”の向上

- ◆ 置換
- ◆ リファクタリング

プロジェクトを成功に導く要因は？

プロジェクト成功の要素



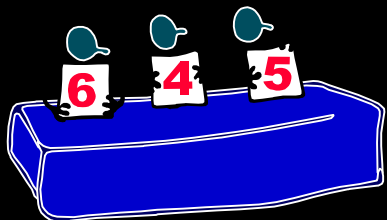
しかし...

- 期限内かつ予算内でプロジェクトが完了するのは全体のわずか 16% (全企業)
 - 9% (大企業)
 - 28% (小企業)
- 53% のプロジェクトが当初の見積もり額を超過
 - 平均超過額: 189% (\$590 億の超過)
- 31% のプロジェクトがプロジェクト完了前に中止 (\$810 億)

1) ユーザの参加	15.9%
2) 実行管理サポート	13.9%
3) 要求の明確化	11.8%

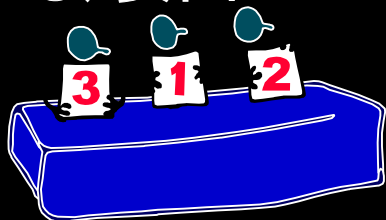
プロジェクトを失敗に導く要因は？

プロジェクトが
難航(期限超過)
する要因



1) ユーザの入力を行っていない	12.8%
2) 要求や仕様が不完全	12.3%
3) 要求や仕様の変更	11.8%

プロジェクトが
行き詰まる
(中止する)要因



1) 要求が不完全	13.1%
2) ユーザが参加していない	12.4%
6) 要求や仕様の変更	8.7%

要求の分類: FURPS+

FURPS+ を構成するもの

機能要求

機能
(Functionality)

特性セット
性能

普遍性
セキュリティ

使いやすさ
(Usability)

人間的な要素
美しさ

整合性
ドキュメント作成

信頼性
(Reliability)

故障の頻度 / 深刻さ
復旧のしやすさ

予測のしやすさ
精度
平均故障間隔

性能
(Performance)

処理速度
効率
リソースの使われ方

スループット
応答時間

サポートのしやすさ
(Supportability)

テスト性
拡張性
適応性
保守性
互換性

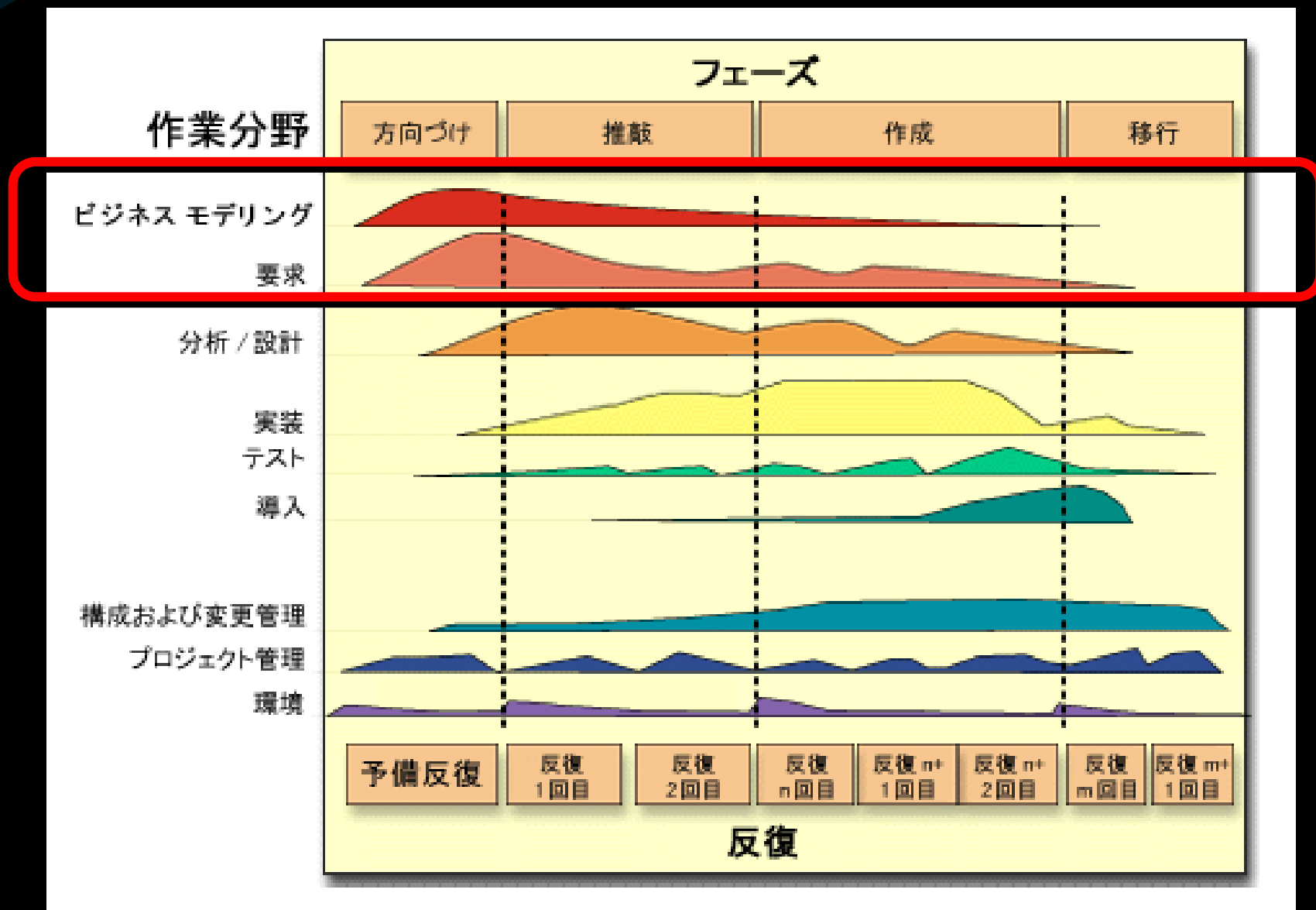
構成容易性
保守容易性
インストール容易性
ローカライズ容易性
頑強性

機能外要求

FURPS+の“+”

- ◆ システム設計上の制約
- ◆ 実装要求
 - コーディング、作成に対しての要求や制約
 - 例: 規約、言語、操作環境等
- ◆ インターフェース要求
 - システムとの相互作用が要求される外部の要素
 - 相互作用で使用されるフォーマット、タイミング等の制約
- ◆ 物理要求
 - 物理的特性
 - 例: 材料、形態、サイズ、重量、物理ネットワーク構成

Rational Unified Process v.2001A



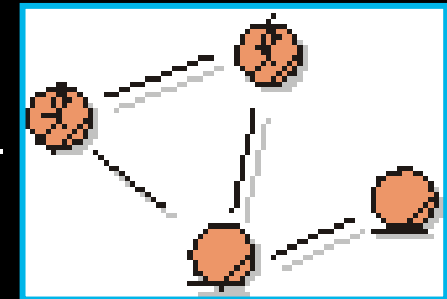
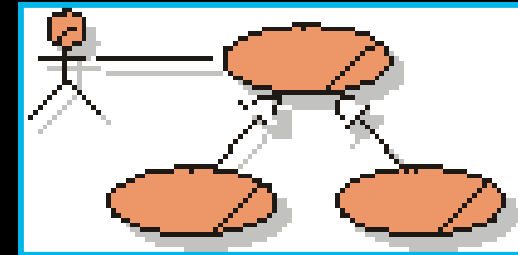
ビジネスモデリングでの3つのモデル

◆ ドメインモデル

- 当該組織の事業ドメインを構成する基本要素のモデル
- 成果物例: ビジネス用語集、ビジネスオブジェクトモデル(ビジネスエンティティのみ)、データモデル

◆ ビジネスモデル

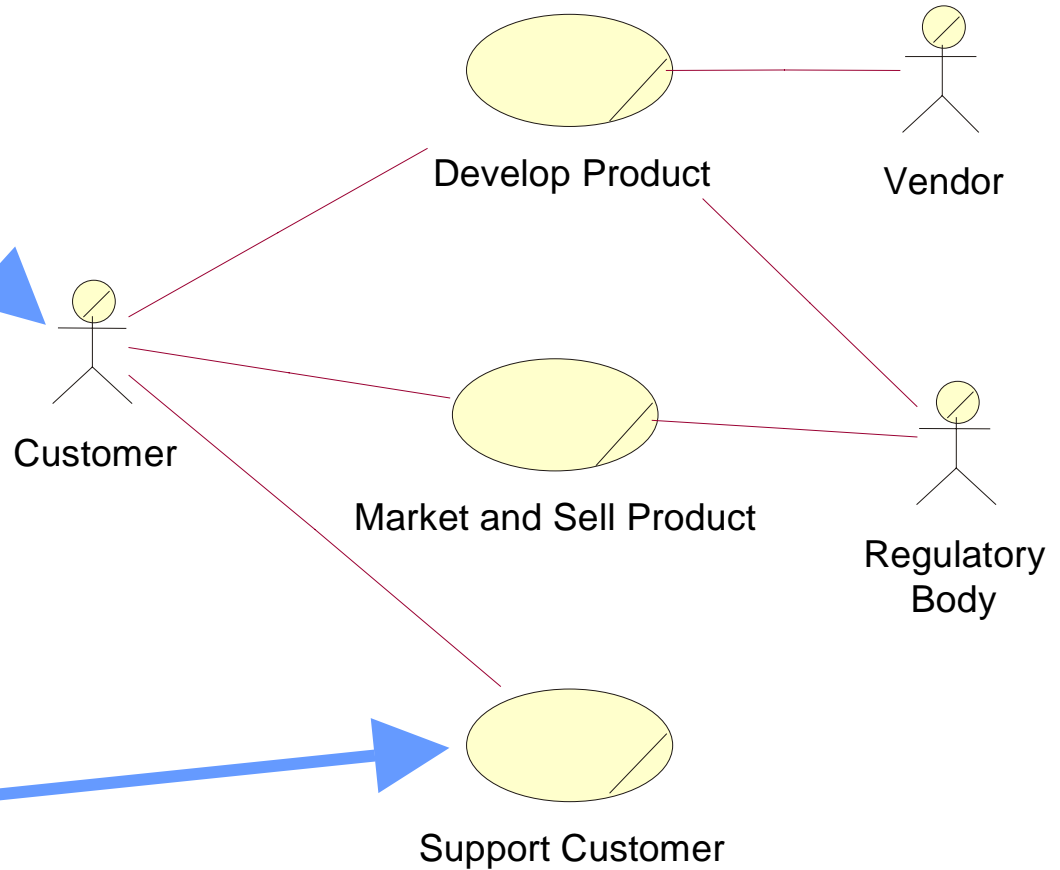
- “ビジネスユースケースモデル” では、ビジネスをブラックボックスとして扱う
 - 当該ビジネスが顧客に対して提供する価値を記述する。
- “ビジネスオブジェクトモデル” は内部のビジネスプロセスをモデル化する
 - ビジネスエンティティ(**Business Entity**)とビジネスワーカー(**Business Worker**)を明確にし、それらがビジネスプロセスを実装するために必要な機能を提供するために行う相互作用をモデル化する。



成果物1: ビジネスユースケースモデル

ビジネスアクター

外部の“モノ”が、
ビジネスとのかかわり
の中で果たす
役割。



ビジネスユースケース
ビジネスアクターに対
して、何らかの観察可
能な価値を提供する、
一連のアクションシー
ケンス

成果物2:ビジネスオブジェクトモデル

ビジネスユースケースの実現を記述する

◆ 構造的要素

- ビジネスエンティティ (内部成果物、製品)
- ビジネスワーカー (役割)
- 責任
- 関連
- イベント

◆ 振る舞い

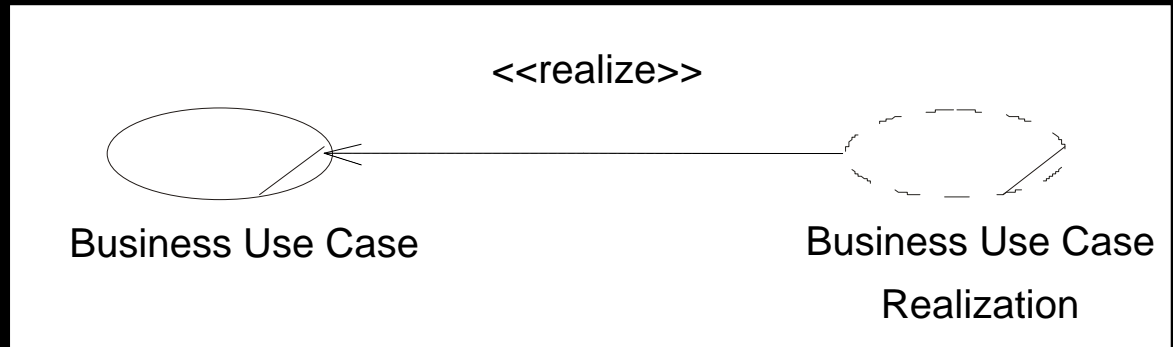
- アクティビティ図

◆ 相互作用

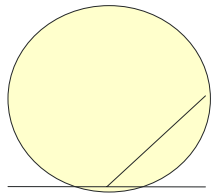
- シーケンス図、コラボレーション図

◆ オブジェクトの状態

- 状態図



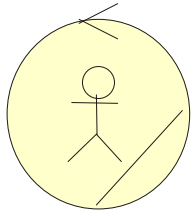
ビジネスエンティティ



Order

- ◆ ビジネスワーカーが、参照し、操作し、生成する対象となるオブジェクト。
- ◆ 例：オブジェクトOrder
 - 名称: Order
 - 詳細説明: 顧客から要求された品物と個数のリスト
 - 責任: 当該オーダーの発注額を算出、等
 - 操作: 出荷日を取得、等
 - 属性: 発注数量、等

ビジネスワーカー



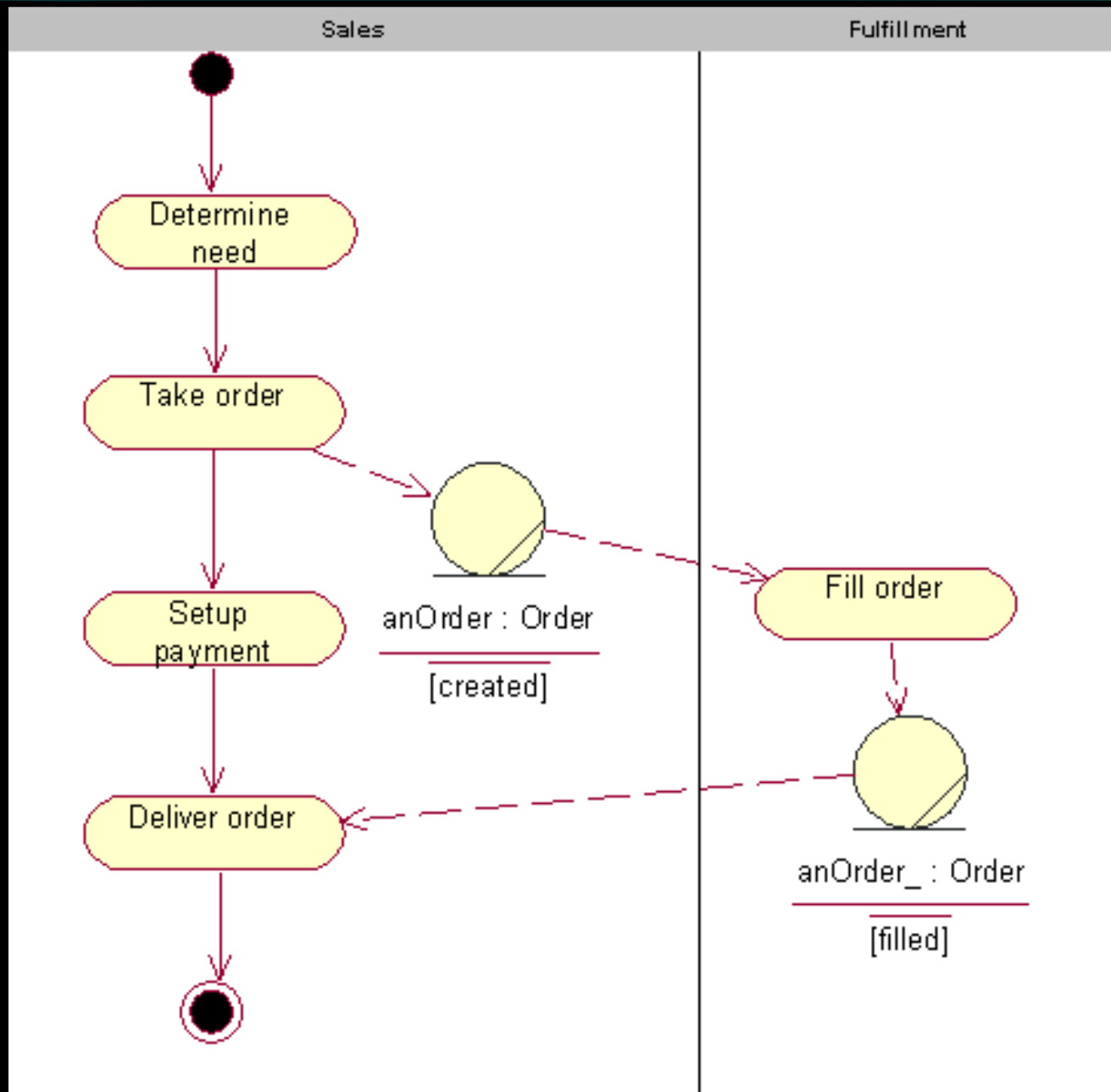
Product Specialist

- ◆ ビジネスの中での役割を表現。他の役割と相互作用し、ビジネスエンティティを操作することにより、特定のユースケースの実現に関与する。

- ◆ 例: オブジェクトProduct Specialist

- 名称: Product Specialist
- 詳細説明: 特定製品の製品知識を提供する
- 責任: 適切な製品を提案する、等
- 操作: 製品を選択する、等
- 属性: 専門分野、担当製品群、等
- 要求される能力: 製品知識、等

オブジェクトフローを伴ったアクティビティ図

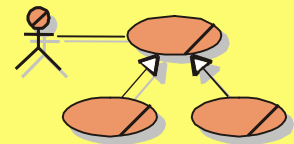


組織単位 (Organization Units)

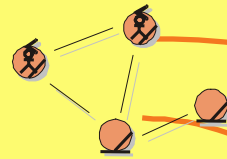
- ◆ 構造をあらわす表記のひとつ
- ◆ ある条件・意味付けに従って、ビジネスワーカー、ビジネスエンティティ、他のユニットをグルーピングする



ビジネスモデルからシステムアーキテクチャへ

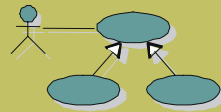


ビジネス
ユースケースモデル



ビジネス
オブジェクト

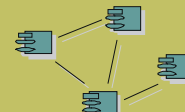
ビジネスモデリング



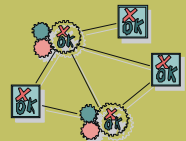
ユースケース
モデル



設計
モデル



実装
モデル



テスト
モデル

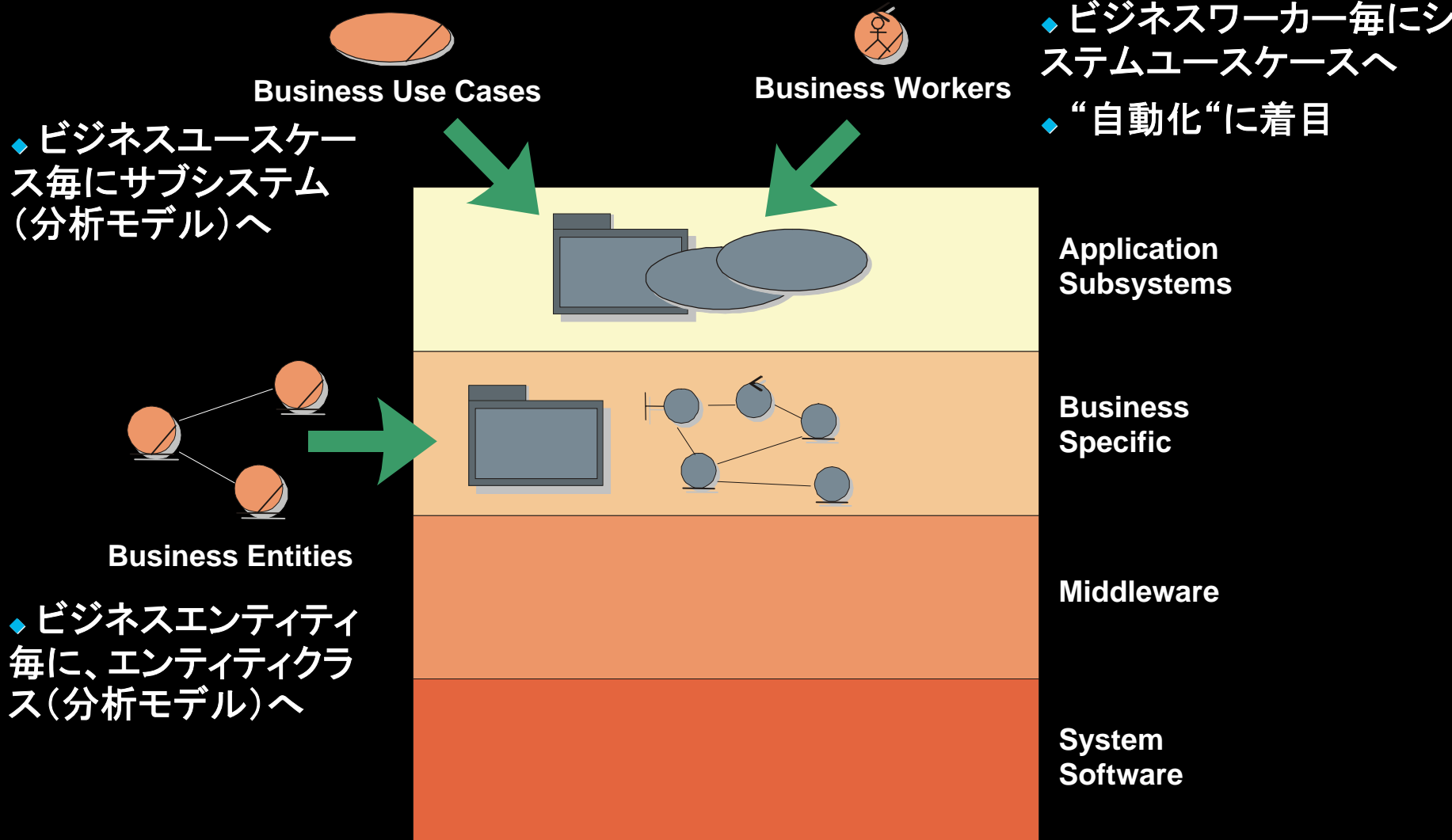
システム開発

ビジネスモデルからシステムアーキテクチャへ

◆ モデル間の関係は…

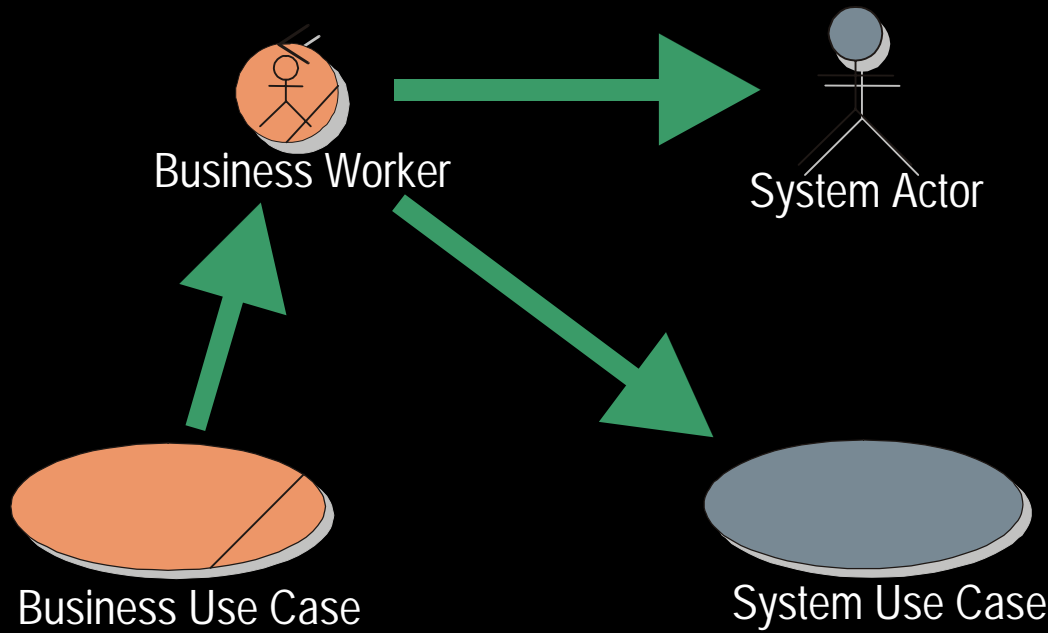
- ビジネス上のコンセプトと、解決策であるITシステムとの追跡可能性を明確にする
- 変更を管理し、それへの対応を計画する際の“仕組み”となる
 - 顧客→ビジネスモデルで“変更”を認識する
 - 開発者→該当するシステムで対応する
- 当該ビジネスが“何をするのか”という視点を開発者にて提示
 - 顧客との間の“理解の共有”

ビジネスモデルからシステムアーキテクチャへ

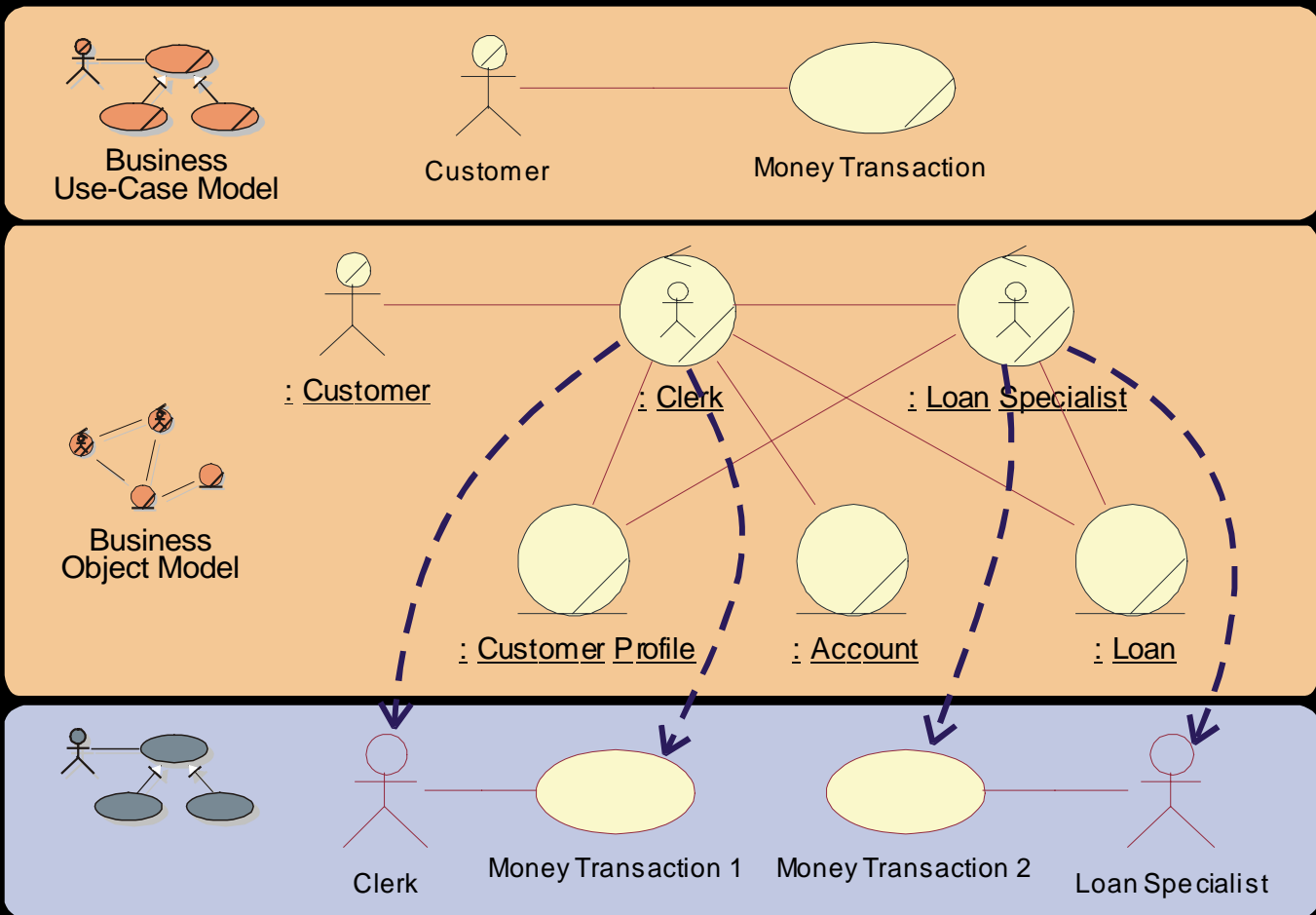


ビジネスモデルとシステムのアクター

- ◆ 各ビジネスワーカーに対応して、システムのアクターを作成する
- ◆ 当該ビジネスワーカーが関与するビジネスユースケースに対応して、システムユースケースを作成する



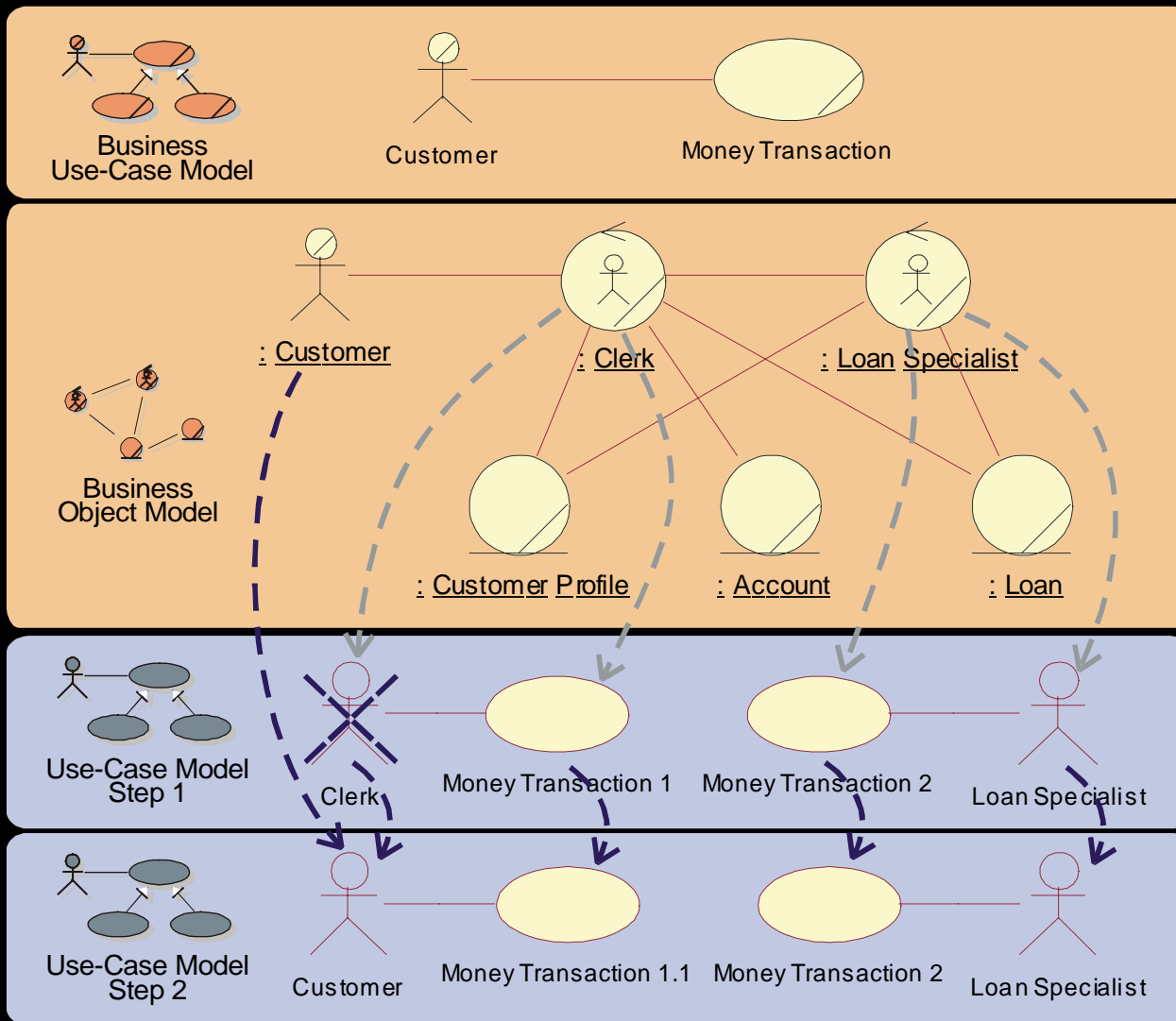
例



“自動化された”ビジネスワーカー

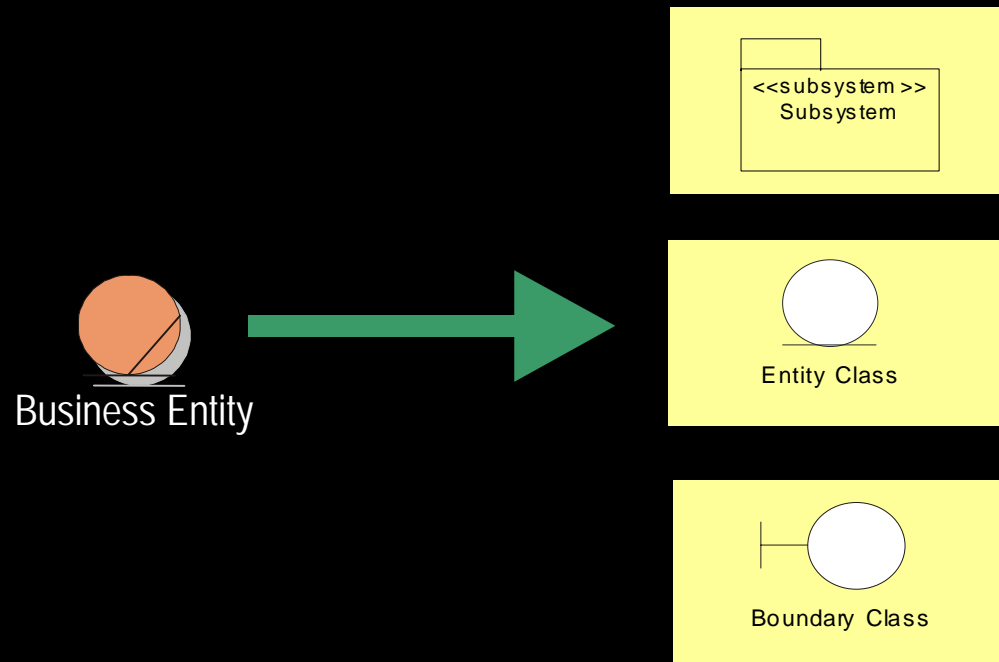
- ◆ 音声応答、Webによるダイレクトアクセスetc...
- ◆ “責任の移動“が発生し得る
- ◆ 例：“Customer”がシステムをダイレクトに相互作用する
 - “Clerk”の責任が“Customer”に移動する
 - ビジネスアクター“Customer”に対応するシステムアクター“Customer”を作成する
 - システムアクター“Clerk”を削除
 - “Clerk”の代わりにシステムアクター“Customer”を使って、システムユースケース“Money Transaction 1”を更新する

例

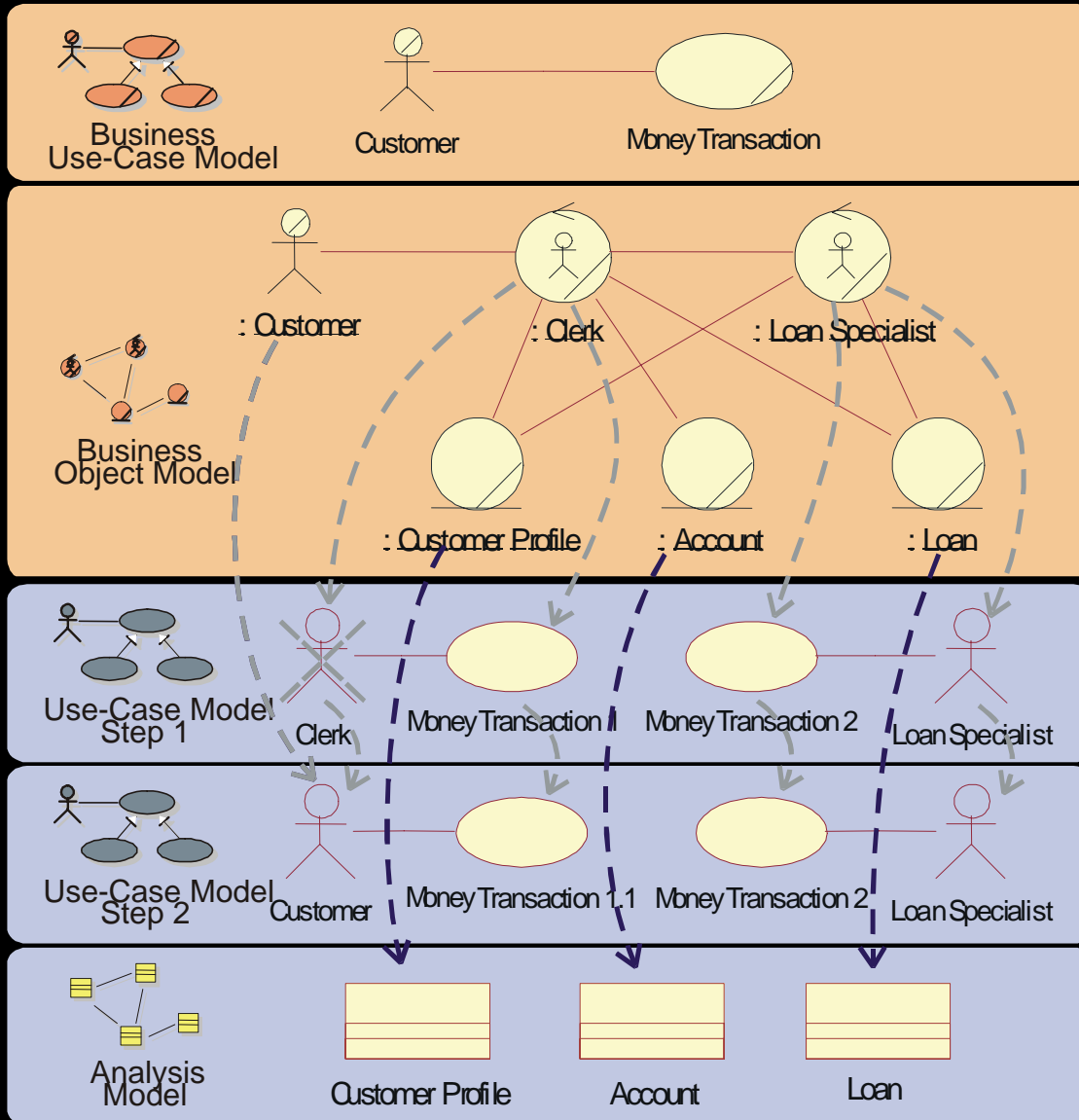


ビジネスモデルと分析モデル

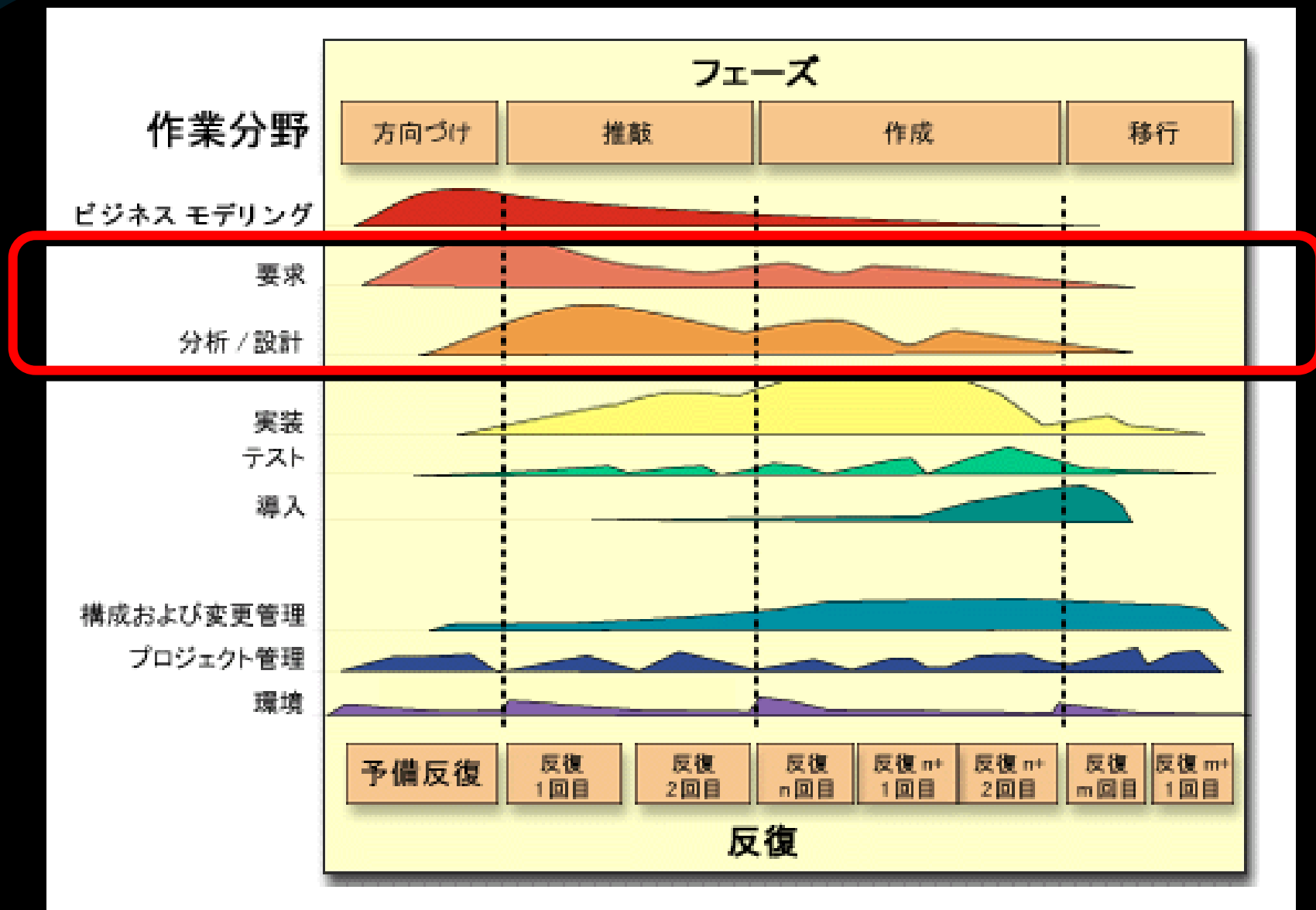
- ◆ ビジネスエンティティから分析クラスを作成する



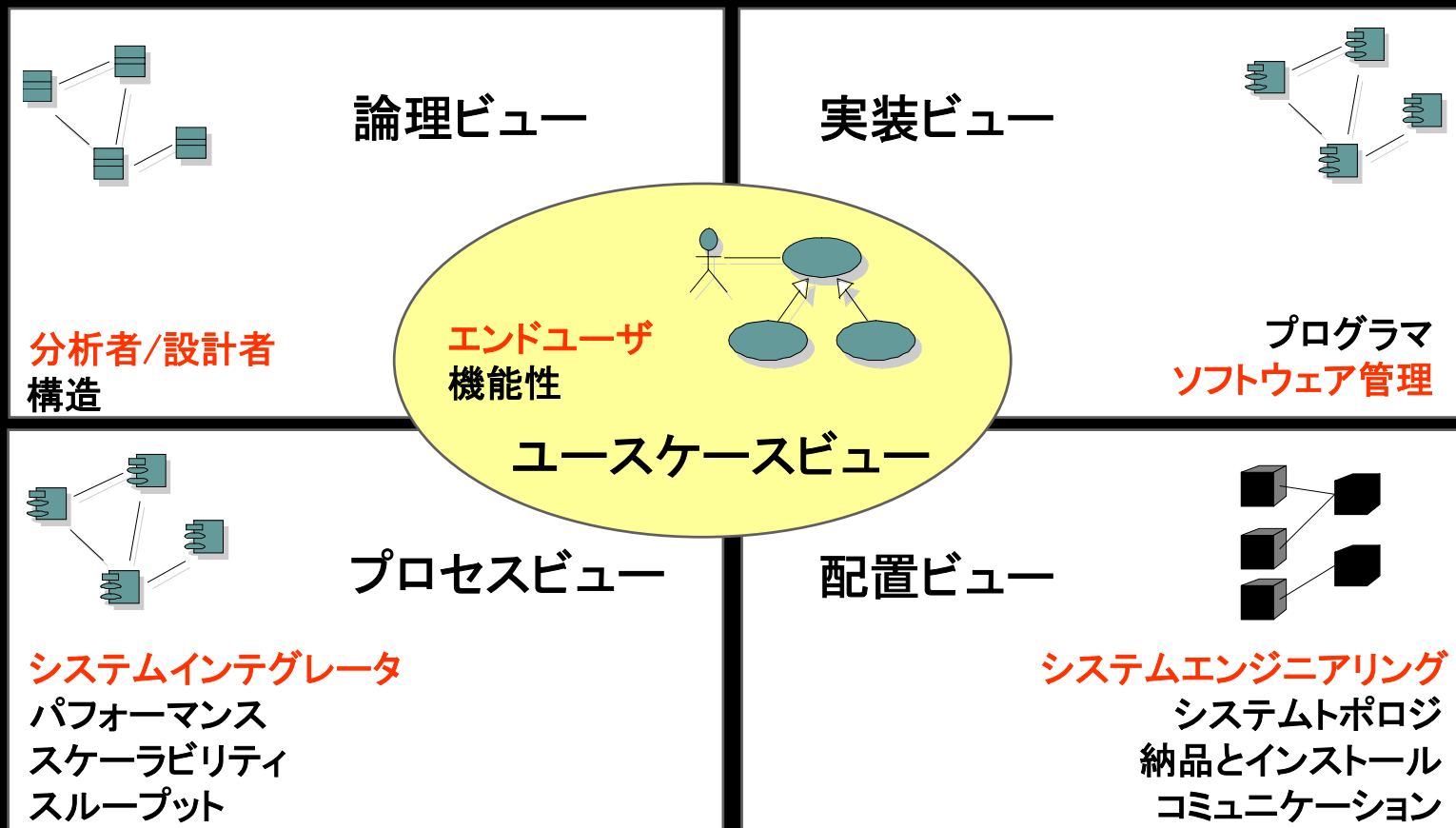
例



Rational Unified Process v.2001A



システムアーキテクチャ：“4+1View”



要求の実現

FURPS+ を構成するもの

機能要求



ユースケース
モデリング

機能外要求



“メカニズム”

機能
(Functionality)

特性セット
性能

普遍性
セキュリティ

使いやすさ
(Usability)

人間的な要素
美しさ

整合性
ドキュメント作成

信頼性
(Reliability)

故障の頻度 / 深刻さ
復旧のしやすさ

予測のしやすさ
精度
平均故障間隔

性能
(Performance)

処理速度
効率
リソースの使われ方

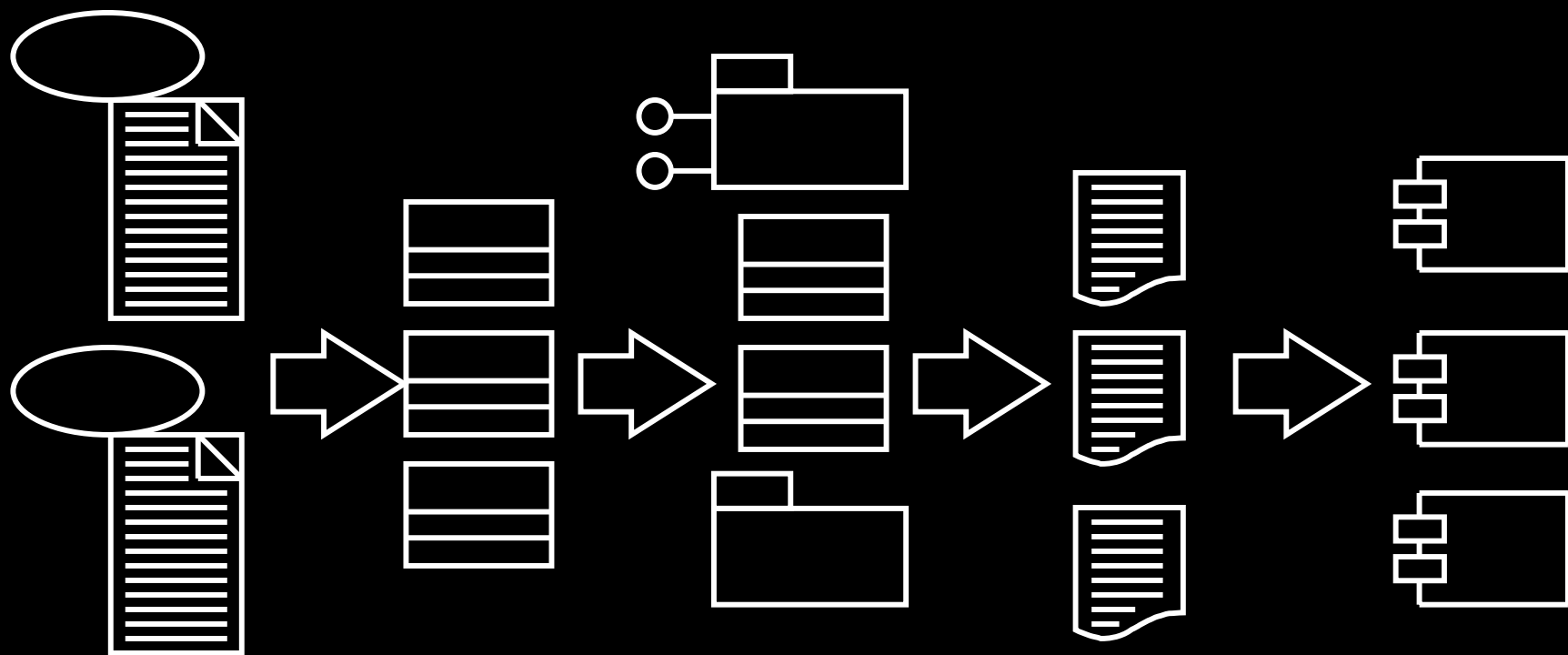
スループット
応答時間

サポートのしやすさ
(Supportability)

テスト性
拡張性
適応性
保守性
互換性

構成容易性
保守容易性
インストール容易性
ローカライズ容易性
頑強性

モデルの変遷



ユースケース

分析
クラス

設計
要素

コード

実行
形式

複雑さを管理：ユースケースモデルの構造化

◆ なぜ、ユースケースモデルを構造化するのか？

- ユースケースの理解を容易にするため
- 多くのユースケース間で共有される振る舞いを再利用するため
- ユースケースモデルの管理を容易にするため

◆ 手段

- パッケージ化→“ユースケースパッケージ”
- アクターの整理(役割の整理)
- ユースケースの構造化



ユースケースの関係

◆ 包含

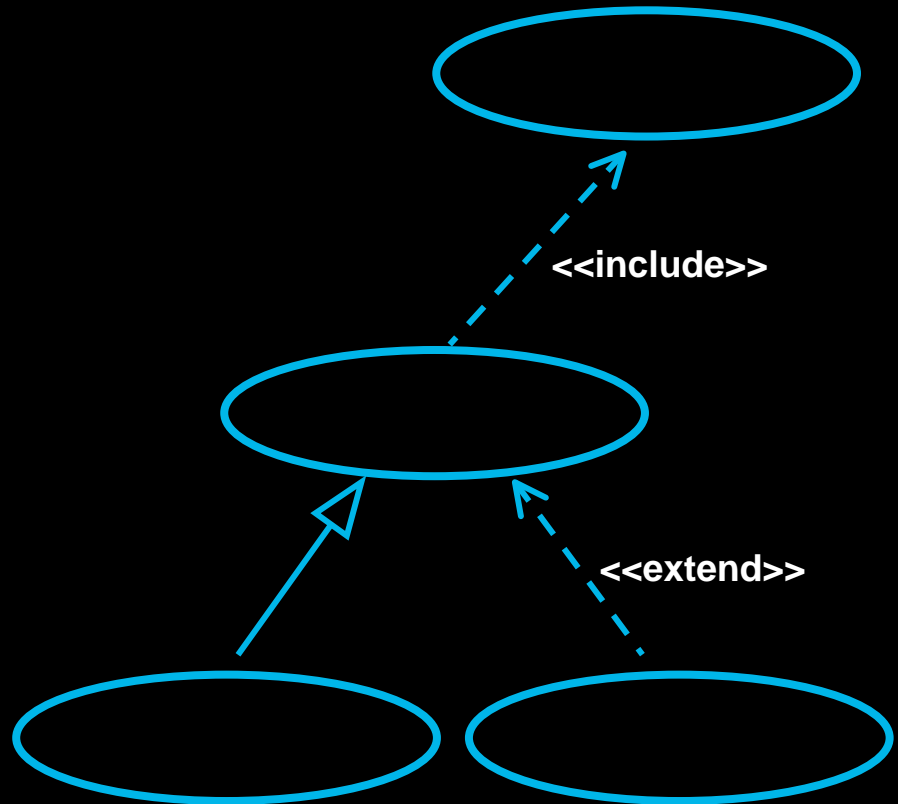
- ユースケースが**その結果に依存する**振る舞いだけを括り出し、結果を実現するための方法は括り出さない(カプセル化)。

◆ 拡張

- ユースケースの**オプション**または**例外的な部分**を括り出す。

◆ 汎化

- ユースケース (またはアクター) が**目的、構造、および振る舞い**を共有することを示す。

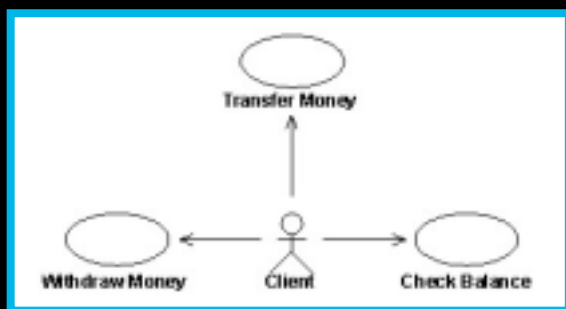


情報を“トレース”する:ユースケースの実現

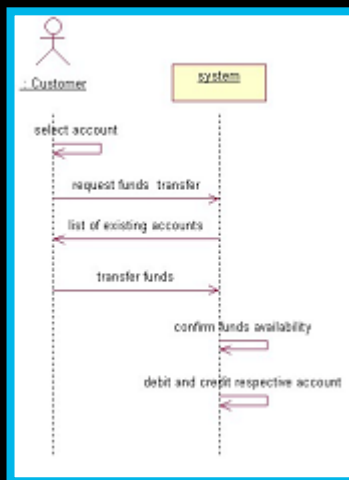
Use Case Model

Design Model

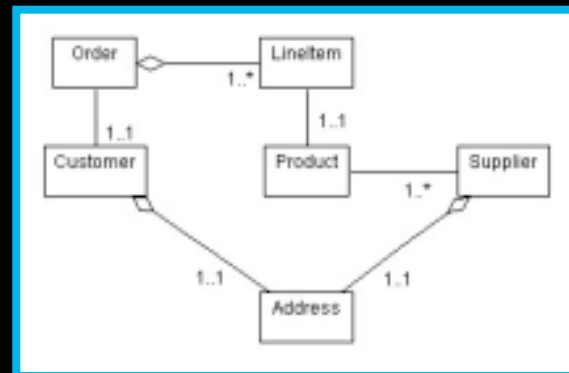
<<realize>>



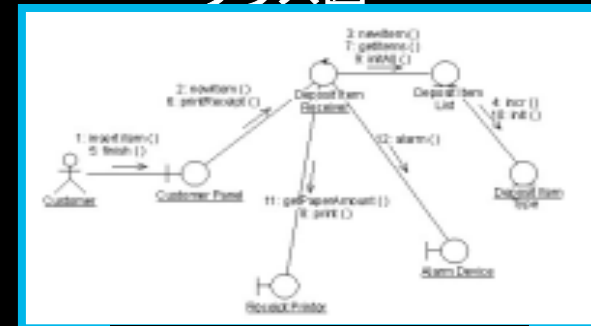
ユースケース図



シーケンス図

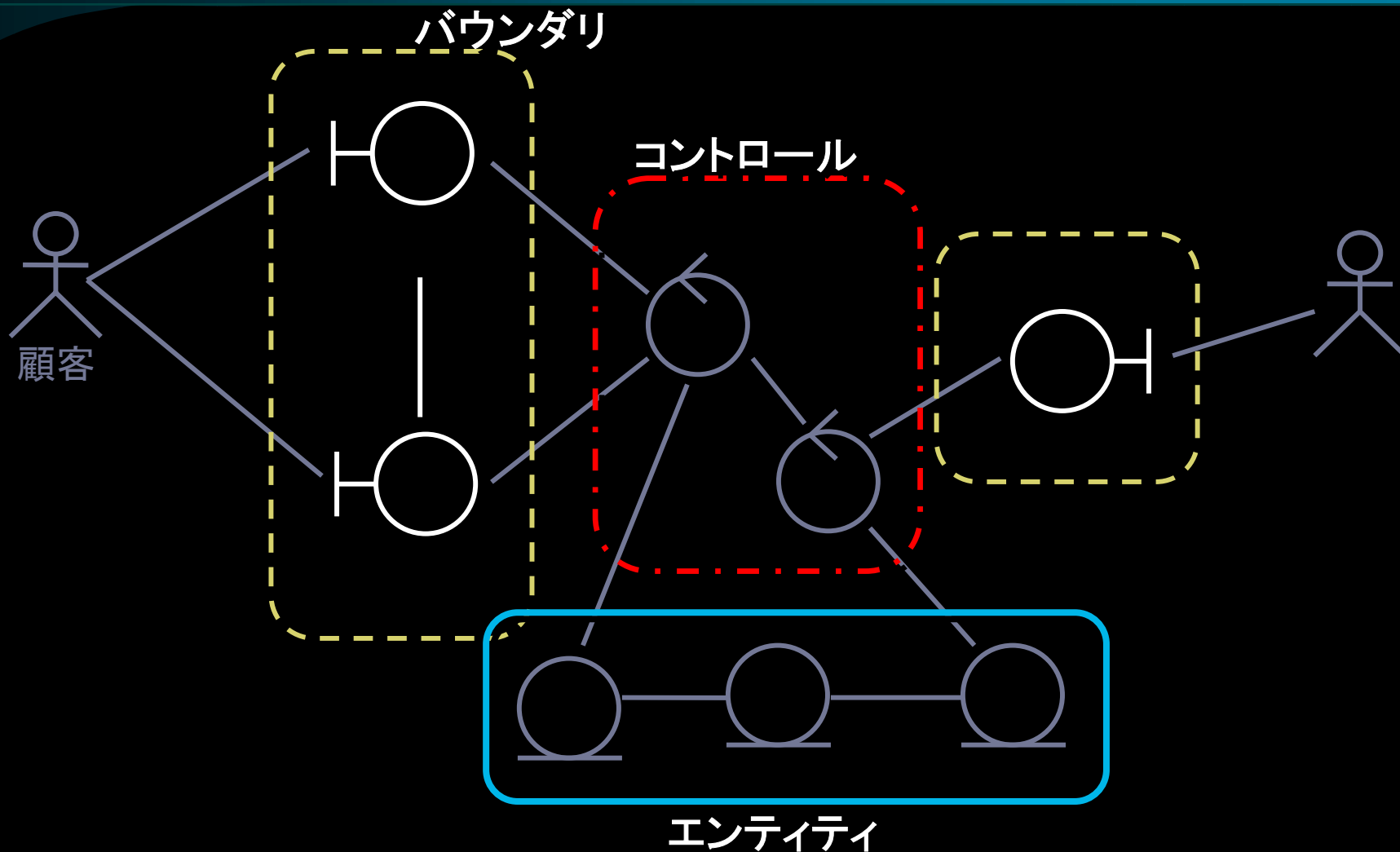


クラス図

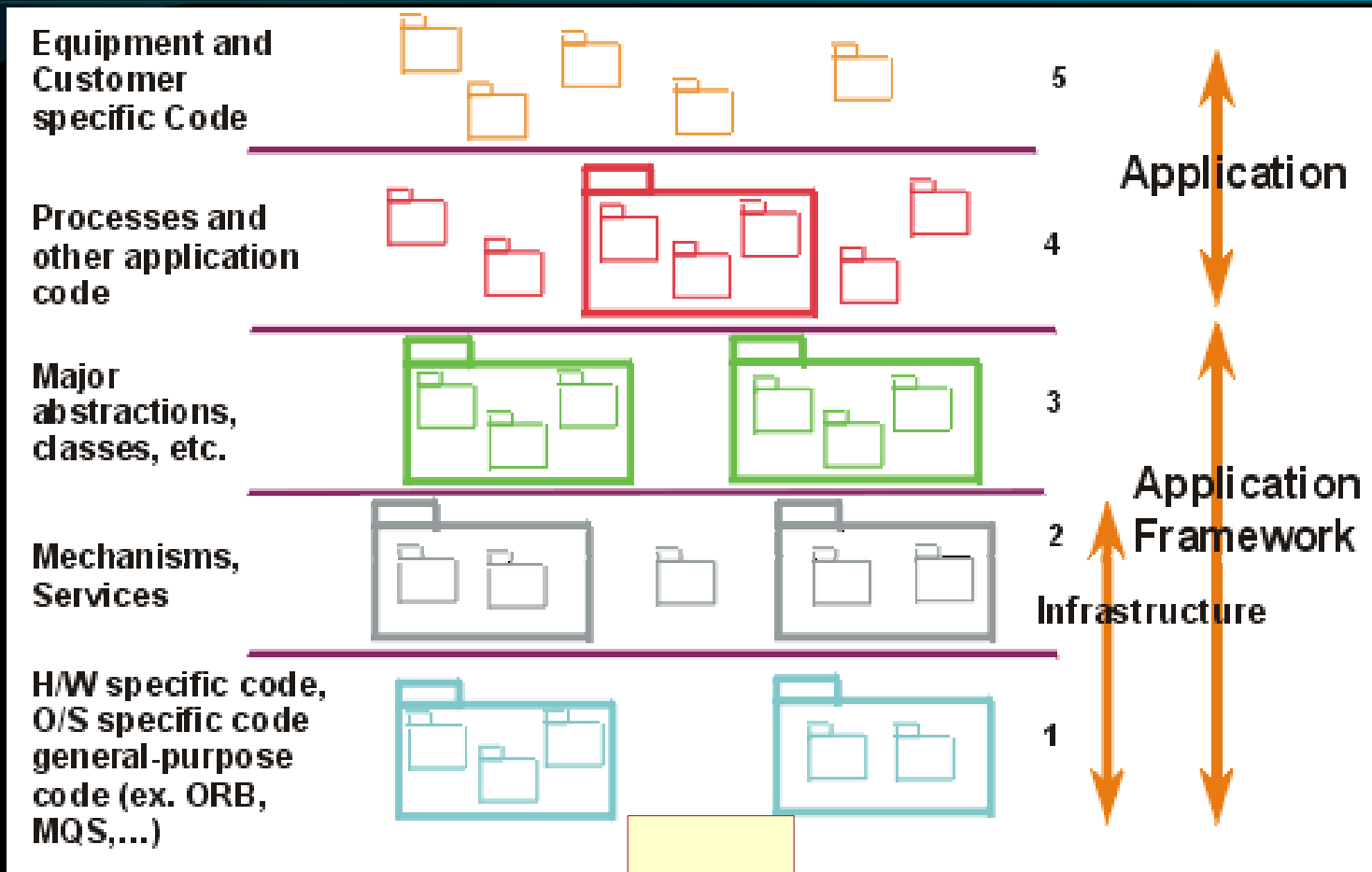


コラボレーション図

分析クラス



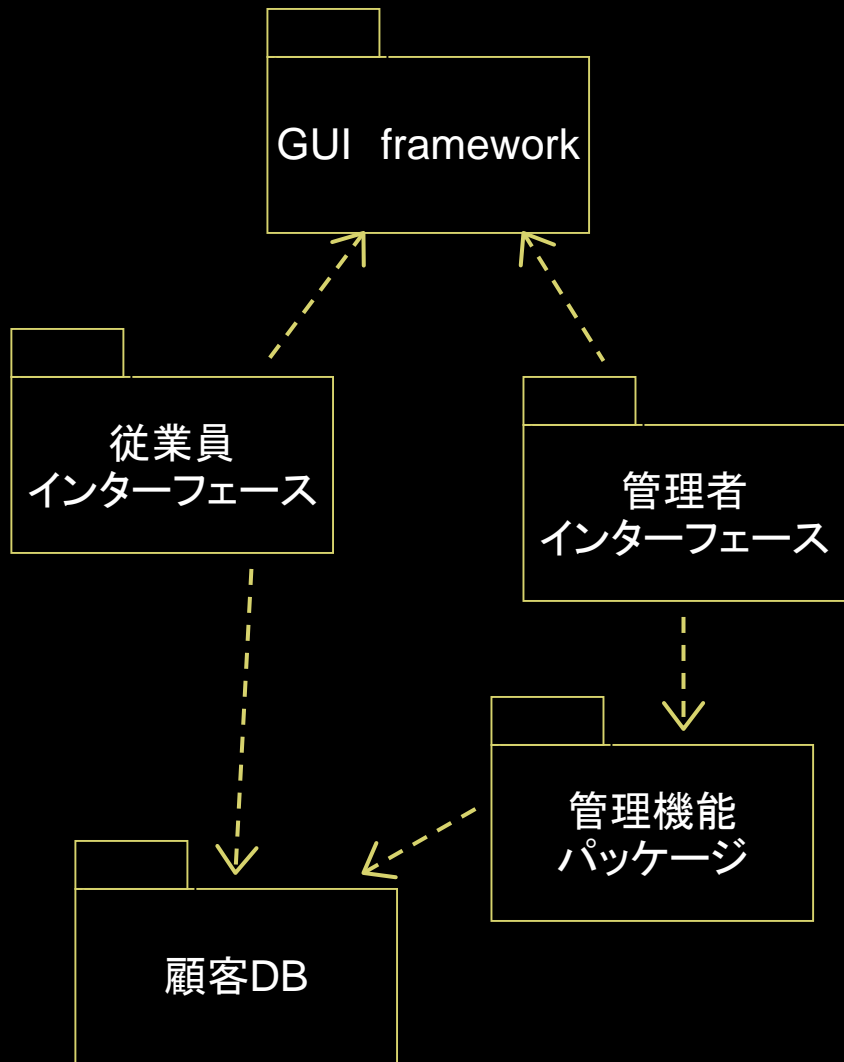
アーキテクチャパターン: Layers



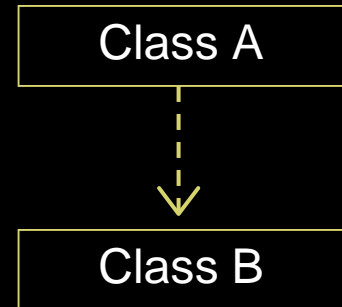
<<layer>>
Package Name

コンポーネント設計のキー: 依存関係

パッケージ間の依存



クラス間の依存

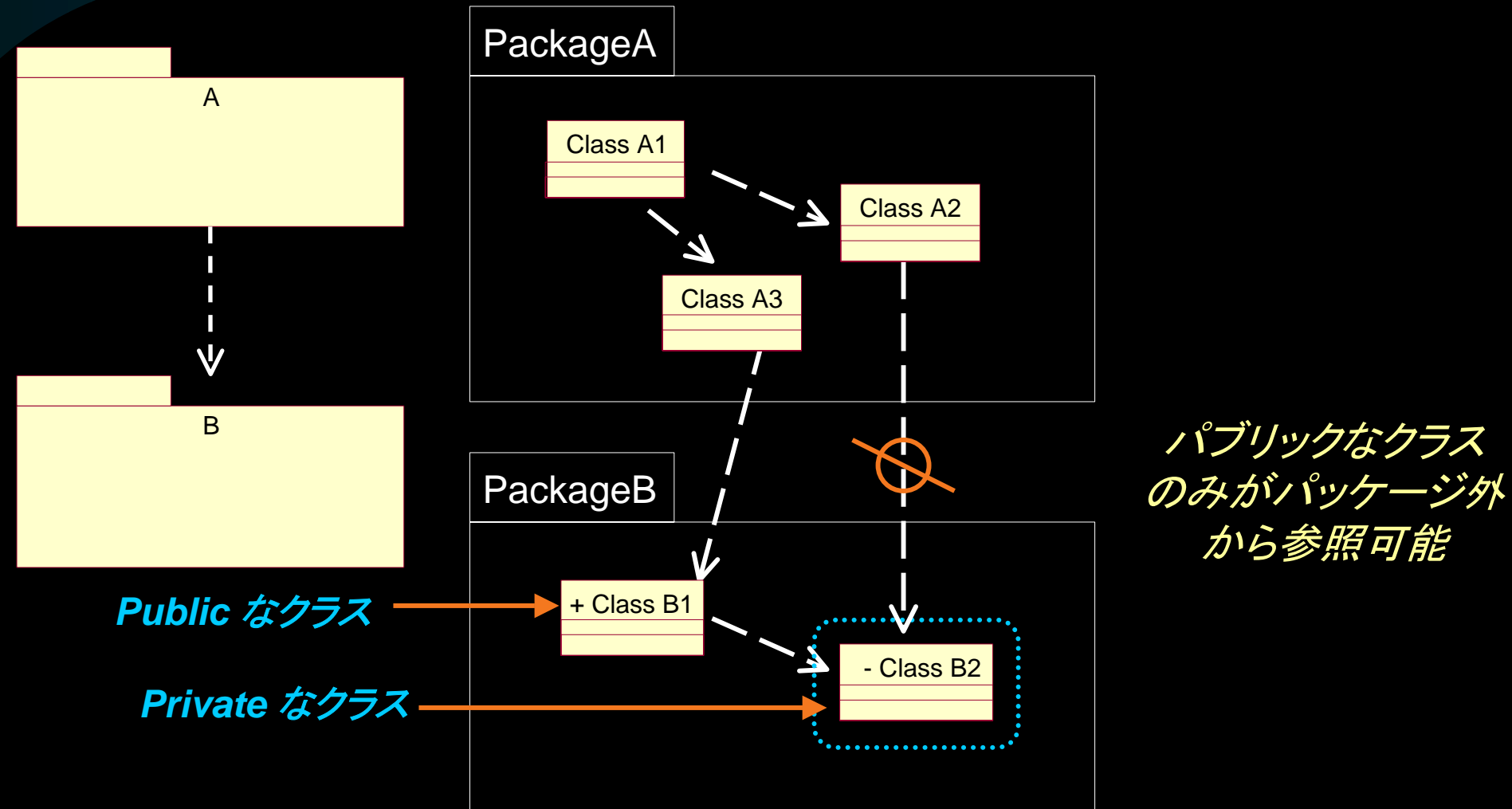


- ◆ クラスAのある操作で、クラスBのインスタンスをローカル変数として使う
- ◆ クラスAのある操作で、クラスBのインスタンスを引数として使う

仕様変更に対して、

- ◆ 短時間での確実な影響範囲の特定
- ◆ 柔軟な構造を設計

パッケージの依存関係: パッケージ要素の可視性



OO Principle: カプセル化 (Encapsulation)

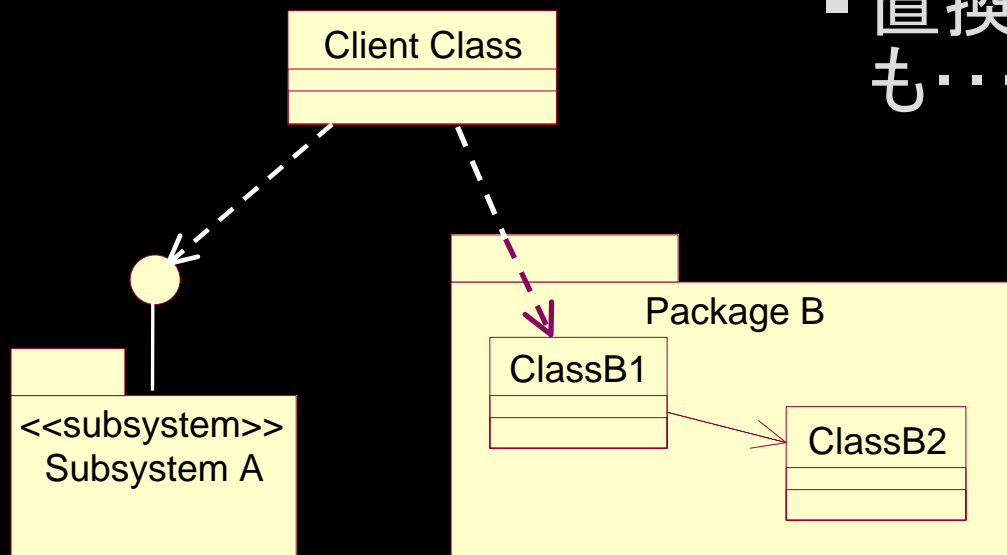
パッケージ vs サブシステム

サブシステム

- 振る舞いを提供
- 構成要素を完全にカプセル化
- 置換可能

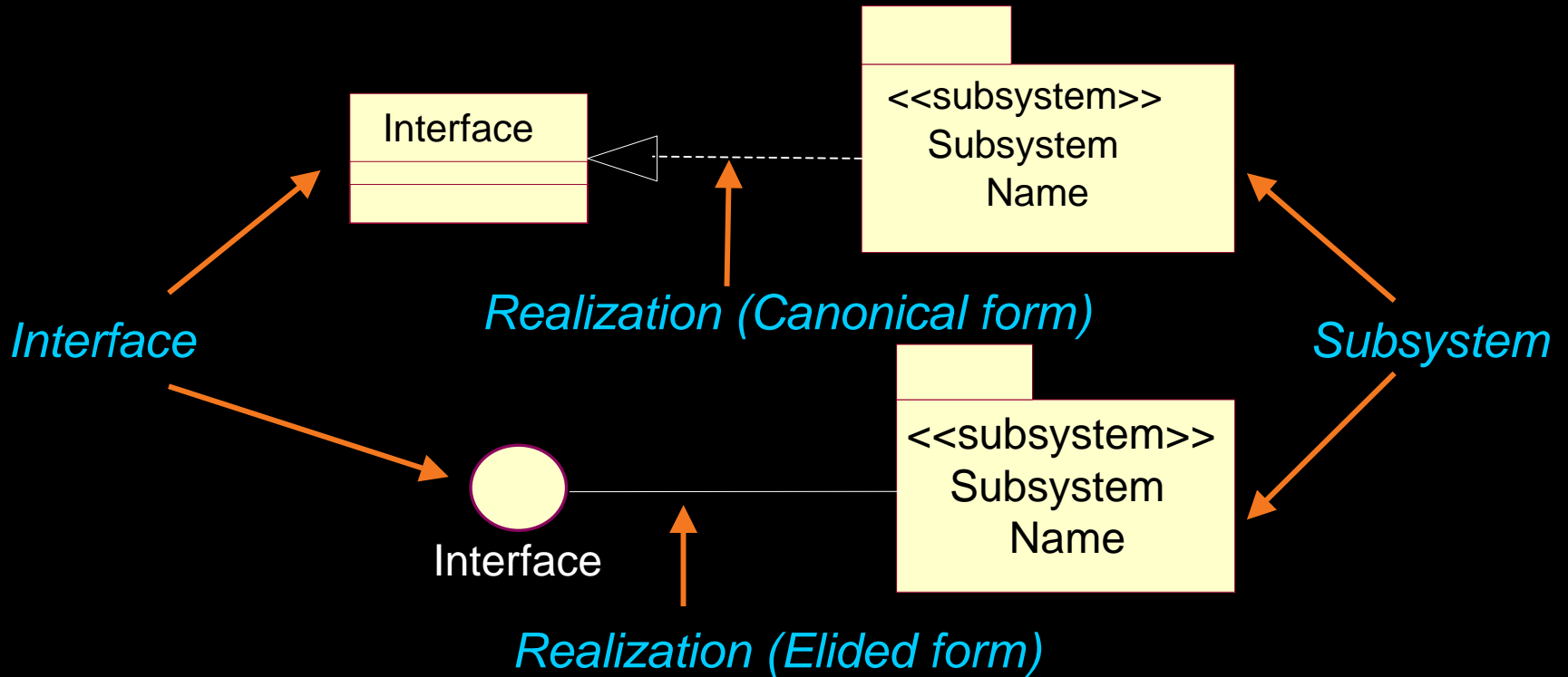
パッケージ

- 振る舞いを表現しない
- 構成要素を完全にはカプセル化しない
- 置換は難しいかも...



カプセル化がキー

サブシステムとインターフェース



機能外要求とは

- ◆ FURPSの“URPS”
 - 使いやすさ
 - 信頼性
 - 性能
 - サポートのしやすさ
- ◆ 法律または規制に関する要求への準拠
 - FCC
 - FDA
 - DOD
 - ISO
- ◆ 設計に対する制約
 - オペレーティングシステム
 - 環境
 - 互換性
 - アプリケーション標準

機能外要求の指定

- ◆ 機能外要求の多くは、個別のユースケースに適用される場合、およびユースケースのプロパティの中で言及される場合がある。
 1. ユースケースの“イベントフロー”節の中で
 - 基本フロー
 - 代替フロー
 2. ユースケースの“特殊要件”節の中で
- ◆ 機能外要求は、システム全体に適用されることが多い。
 3. このような要求は、“補足仕様書”の中で指定される。



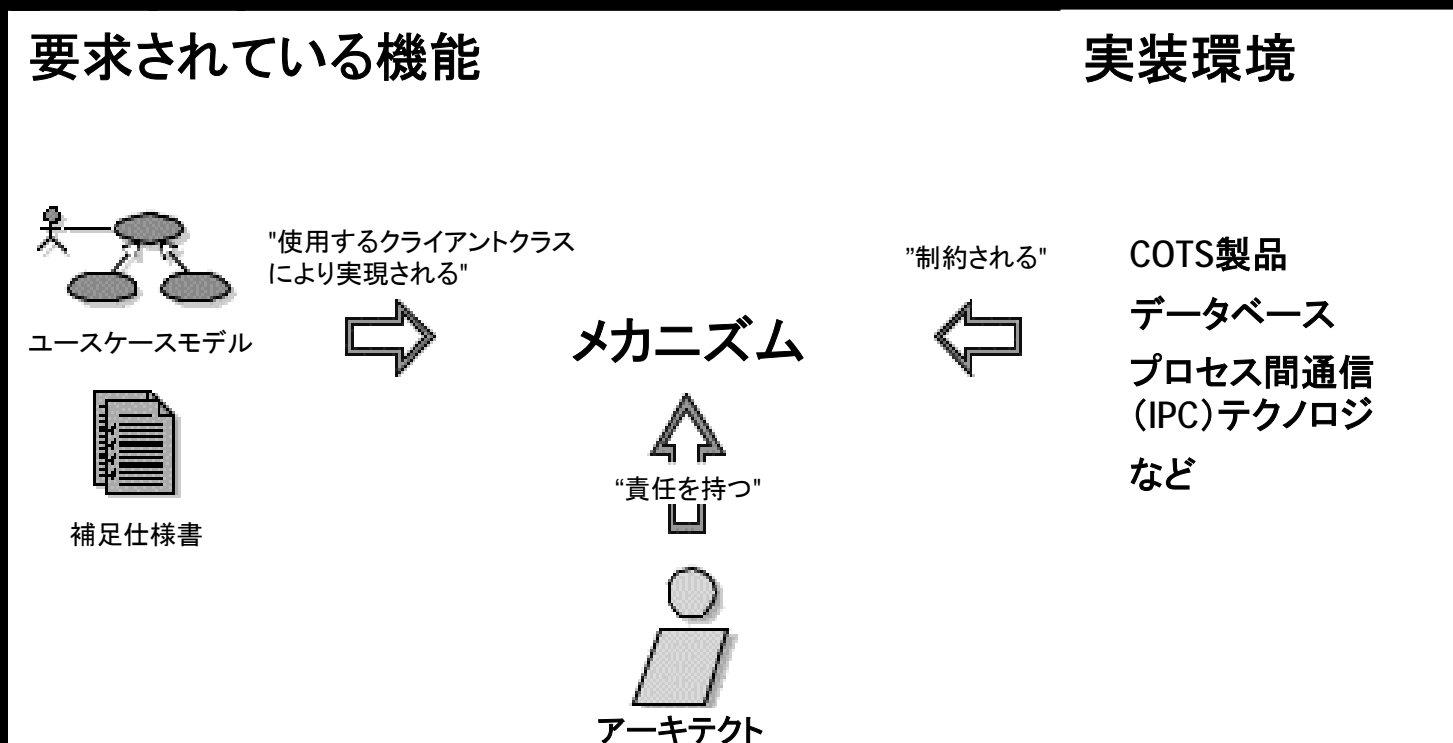
メカニズム：非機能要求を管理する

- ◆ ひとつの手として、非機能要求を顕在化・対応するための体系的アプローチに利用する
- ◆ RUPでの定義
 - パターンのインスタンスである。
 - 1つのコンテキストにおける（繰り返し起きる問題への）特定のソリューション
 - 繰り返し発生するソフトウェアアプリケーションの問題に対してソリューションを提供するコラボレーション
 - 例：永続性の処理
 - デザインパターンの適用

アーキテクチャのメカニズムとは

◆ アーキテクチャのメカニズム

- 分析メカニズム（概念的）
- 設計メカニズム（具体的）
- 実装メカニズム（現実的）



分析メカニズムの記述

- ◆ すべての分析メカニズムを一覧にまとめる
- ◆ 分析メカニズムに対するクライアントクラスの対応を記述する

分析クラス	分析メカニズム
学生	永続性、セキュリティ
教授	永続性、セキュリティ
コース科目	
コース	
成績評価提出コントローラ	分散性
登録コントローラ	分散性

- ◆ 分析メカニズムの特性を識別する

分析メカニズムの例

- ◆ 永続性
- ◆ コミュニケーション（IPCおよびRPC）
- ◆ メッセージルーティング
- ◆ 分散性
- ◆ トランザクション管理
- ◆ プロセス制御と同期化（リソースの競合）
- ◆ 情報交換、フォーマット変換
- ◆ セキュリティ
- ◆ エラーの検出/処理/報告
- ◆ 冗長性
- ◆ レガシーシステムとのインターフェイス

分析メカニズムの特性

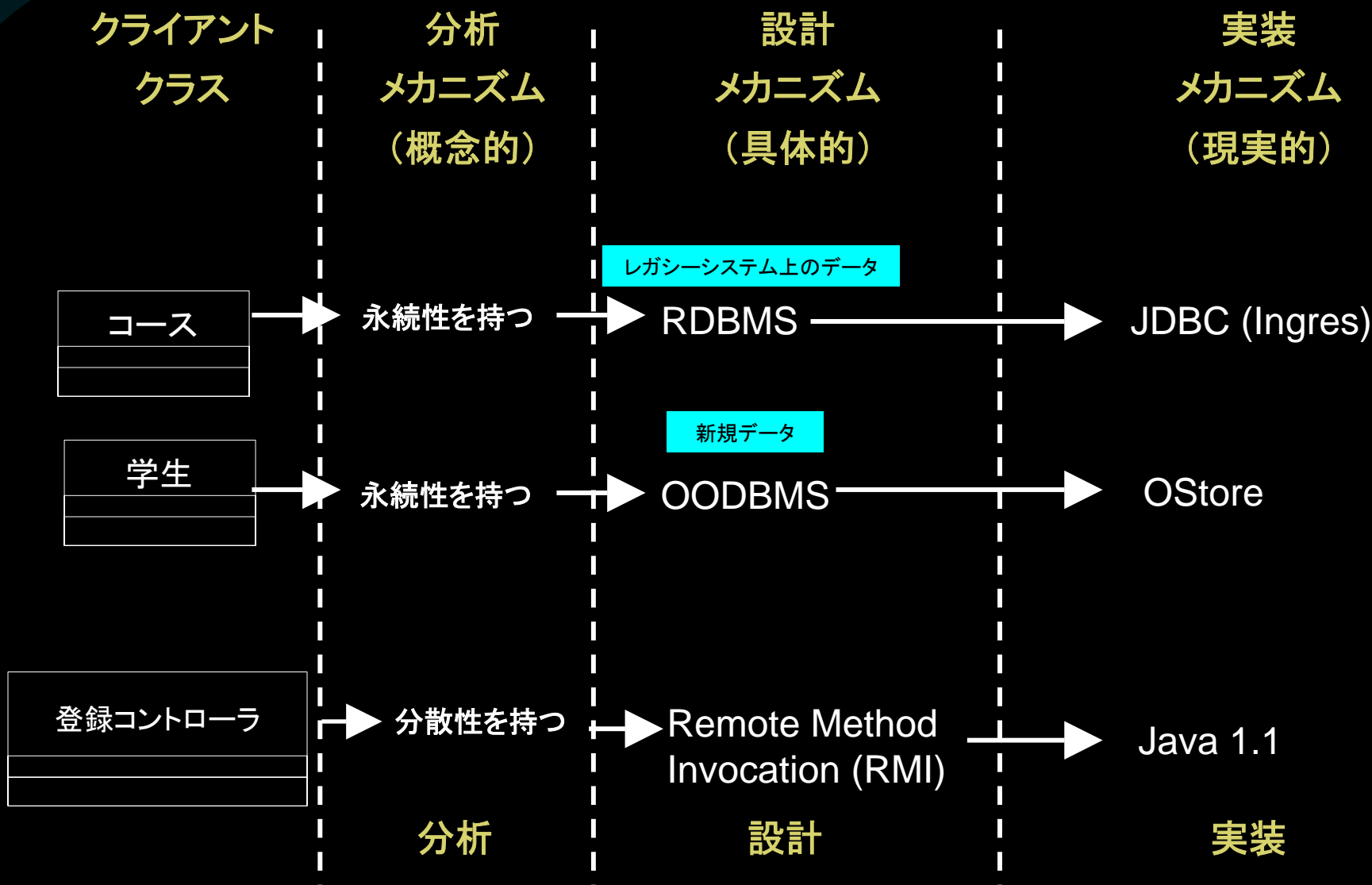
◆ 永続性

- 粒度
- 量
- 期間
- アクセスメカニズム
- アクセス頻度（作成/削除、更新、読み取り）
- 信頼性

◆ 通信

- 待ち時間
- 同時性
- メッセージサイズ
- プロトコル

設計/実装メカニズム



まとめ

- ◆ コンポーネント設計は、オブジェクト指向技術が利益をもたらすためのキー
- ◆ UMLによるビジュアルライズは、コンポーネント設計での、モデルの洗練に必要不可欠
 - 実現関係
 - 依存関係
- ◆ ビジネスモデルーシステムモデルーコンポーネントの追跡可能性に注目
 - ビジネスモデルからコンポーネント再利用可否を判断
- ◆ 設計の決定要因として非機能的要求のハンドリングが重要
 - デザインパターンは、“覚える”ものではなく、非機能的要求の“事前チェックリスト”として活用すべき

Thank

You

Rational Software Corporationについて

世界最大のソフトウェア企業の一社

- ◆ 設立：1981年
- ◆ 売上：\$ 8.15億 (2001年度)
(約978億円)
- ◆ 従業員：3,500+名
- ◆ 拠点：全世界に65箇所
- ◆ 研究開発費：\$1.6億
- ◆ ユーザー数：500,000以上



フォーチュン 100社のうち96社が
Rational社のソリューションを適用

日本ラショナルソフトウェア株式会社

- ◆ 設立 : 1997年7月
- ◆ 資本金 : 2億6000万円
- ◆ 従業員数 : 68名 (2001年11月現在)
- ◆ 日本国内における米国ラショナルソフトウェア社製品の販売およびサポート、トレーニング、技術コンサルティングの提供

