Java EEのためのモデリング入門
ユースケースからEJB、SOAへ

2007年11月2日

浅海智晴
日本Javaユーザ会
稚内北星学園大学東京サテライト校
浅海智晴事務所

日本Javaユーザ会

- 日本 Java ユーザグループ (JJUG) は、Java 技術の向上・発展、開発者の支援を目的とした任意団体です。
- http://www.java-users.jp/
- 会長：丸山不二夫
- 副会長：ひがやすお、浅海智晴
- 主な活動
  - Java基礎セミナー
  - クロスコミュニティ
  - JJUG クロスコミュニティカンファレンス 2007 Fall
    - 11月6日(火) 東京・有楽町国際フォーラム
    - http://www.java-users.jp/contents/events/ccc2007fall/

内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ

内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ

なぜモデリングなのか

- **ビジネスとITの融合**
- EoD
- **オフショア開発**

ビジネスとITの融合

- X-Over Development Conference
  - 2007年9月7日、日経BP社主催
  - http://itpro.nikkeibp.co.jp/ev/xdev/index.html
- **対談「【浅海智晴　×　細川努】これからのアーキテクチャの条件」**
  - **細川さん**
    - **(株)アーキテクタス代表、総務省CIO補佐官**
  - http://itpro.nikkeibp.co.jp/article/NEWS/20070910/281528/

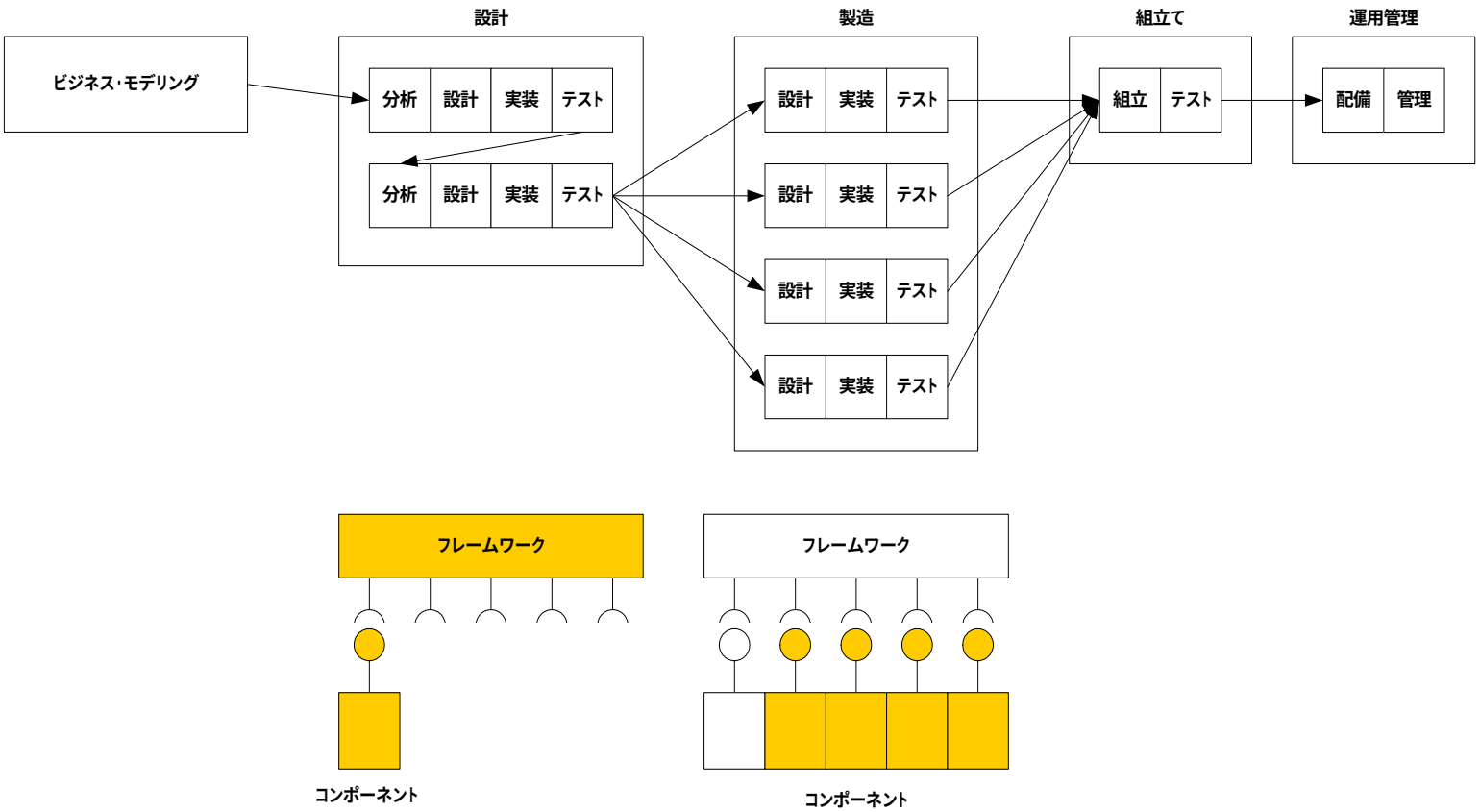
EoD(Ease of Development)

- EoDはJava開発者にとって必ずしも福音ではない
  - 簡単な開発はなくなるということ
- プログラマとツールの競争
  - ツールで自動生成できるレベルのプログラムを手作業で作成することで収入を得ているエンジニアには厳しい状況になる

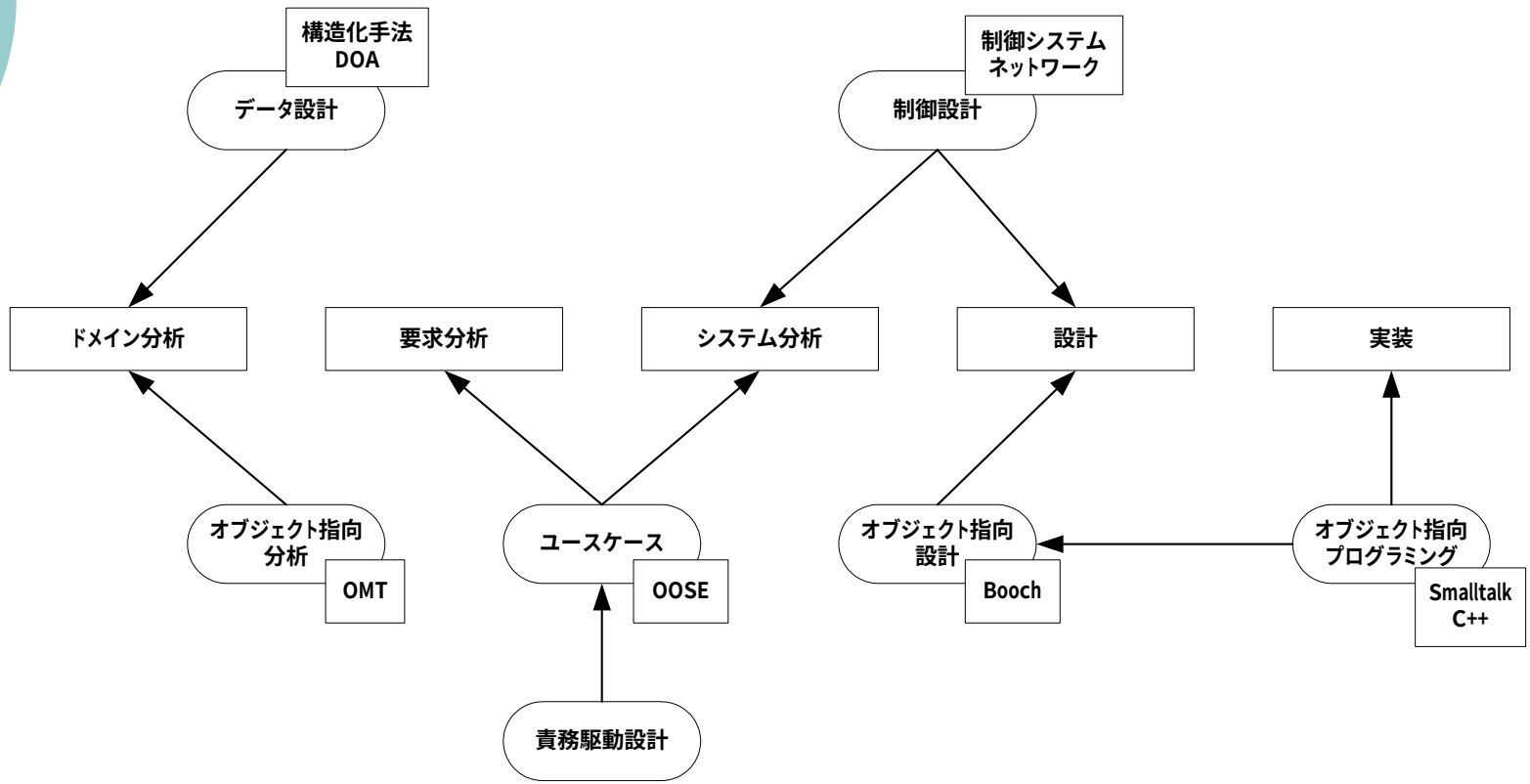
オフショア開発

- インタフェースが明確な一定のまとまりのある機能の”製造”はオフショア開発が有利
  - 分かりやすいプログラミングは国内に残らない
- 分析/設計/プログラミングをアドホックに混在させた開発スタイルは成立しない
  - 仕様を正確･精密に定義できることが必要条件
  - OOモデルを軸にした本格的なOOA/D(Object-Oriented Analysis/Design)によるシステム開発に移行

開発の流れ

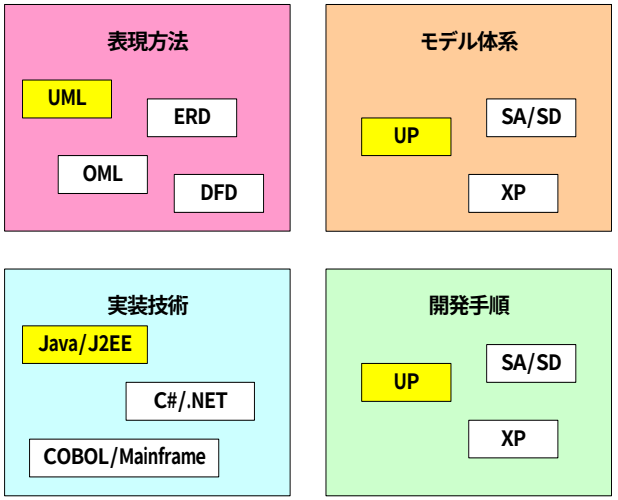

設計
製造
組立て
運用管理
ビジネス・モデリング
分析
設計
実装
テスト
分析
設計
実装
テスト
設計
実装
テスト
設計
実装
テスト
設計
実装
テスト
設計
実装
テスト
組立
テスト
配備
管理
フレームワーク
フレームワーク
コンポーネント
コンポーネント

モデリング技術の流れ

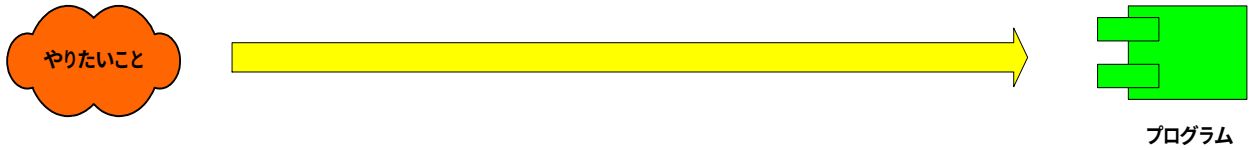

構造化手法
DOA
データ設計
制御システム
ネットワーク
制御設計
ドメイン分析
要求分析
システム分析
設計
実装
オブジェクト指向
分析
OMT
ユースケース
OOSE
オブジェクト指向
設計
Booch
オブジェクト指向
プログラミング
Smalltalk
C++
責務駆動設計

オブジェクト指向開発技術の構成

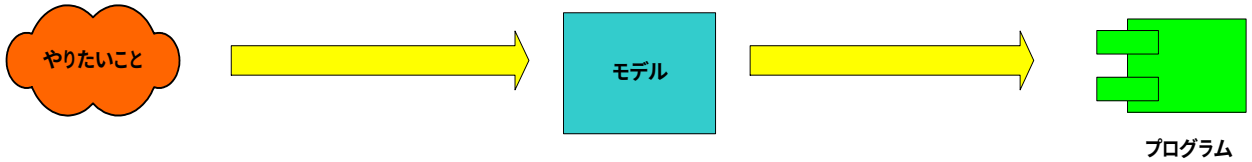
- **表現方法**
  - UML
  - DSL(Domain Specific Language)
- **モデル体系**
  - **オブジェクト・モデル**
- **開発手順**
  - **作業手順**
  - **プロジェクト管理**
- **実装技術**
  - Java＋JavaEE



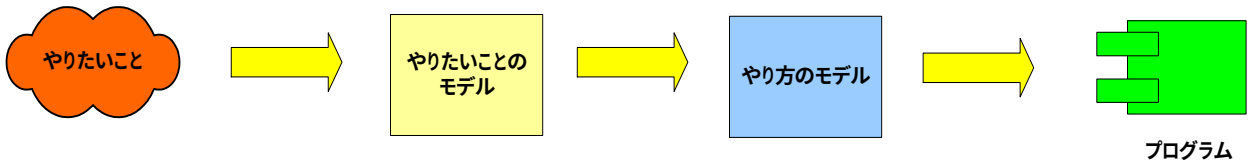
モデルの意義



(1) “やりたいこと”とプログラムの間の距離は長い

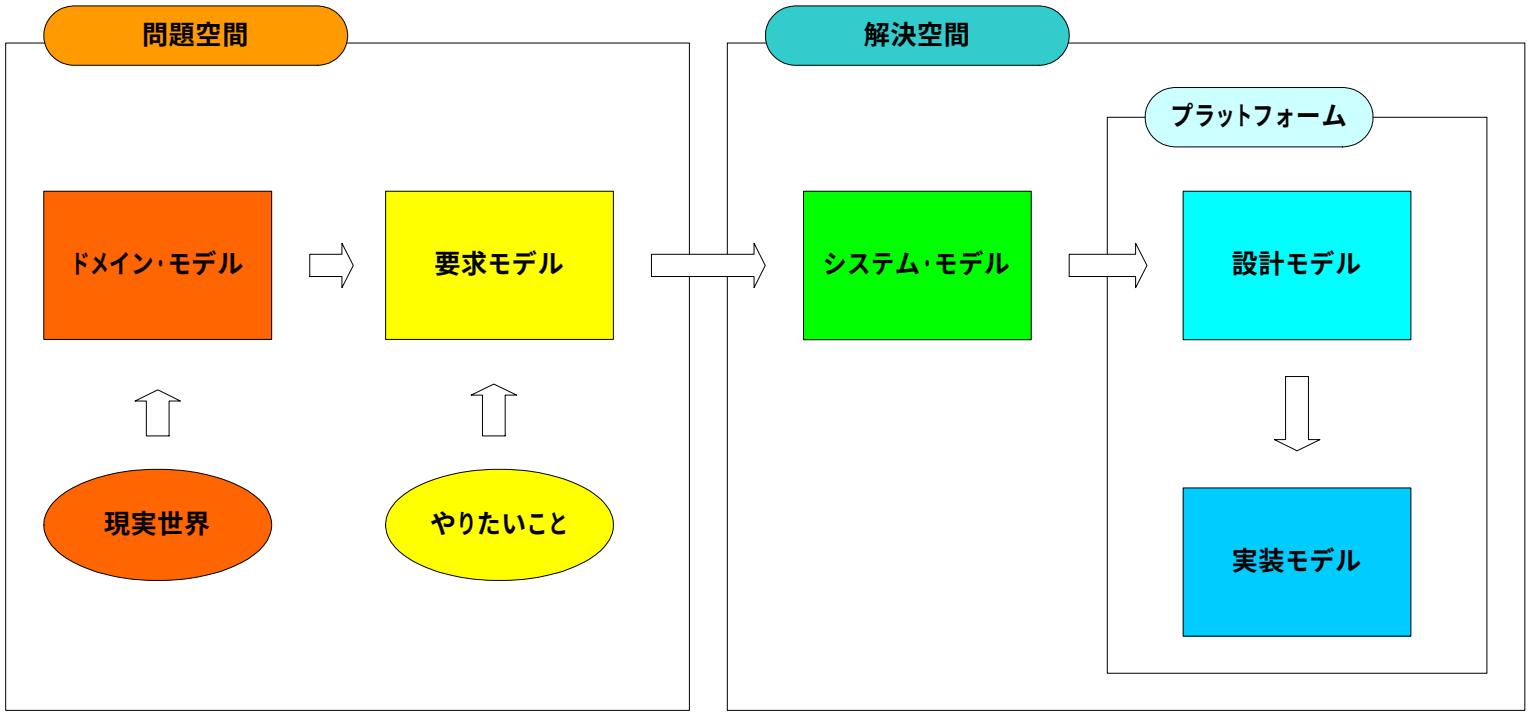


(2) “やりたいこと”とプログラムの間をモデルで中継

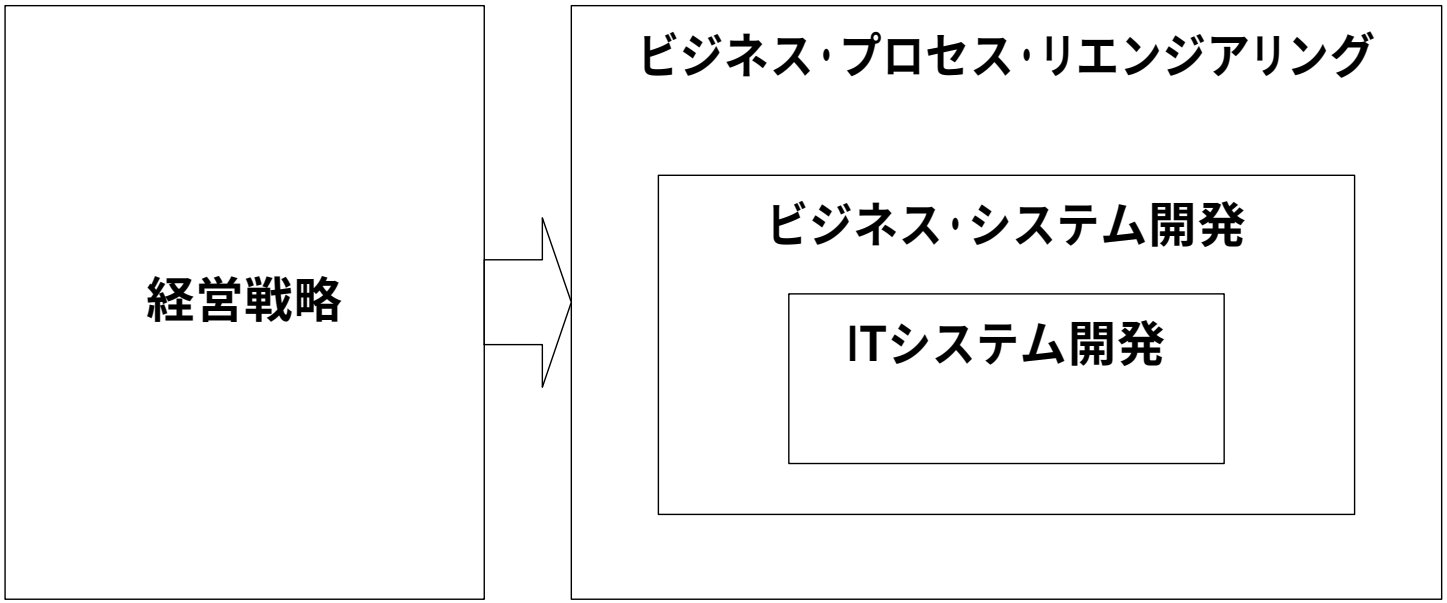


(3) “やりたいこと”のモデルとやり方のモデル

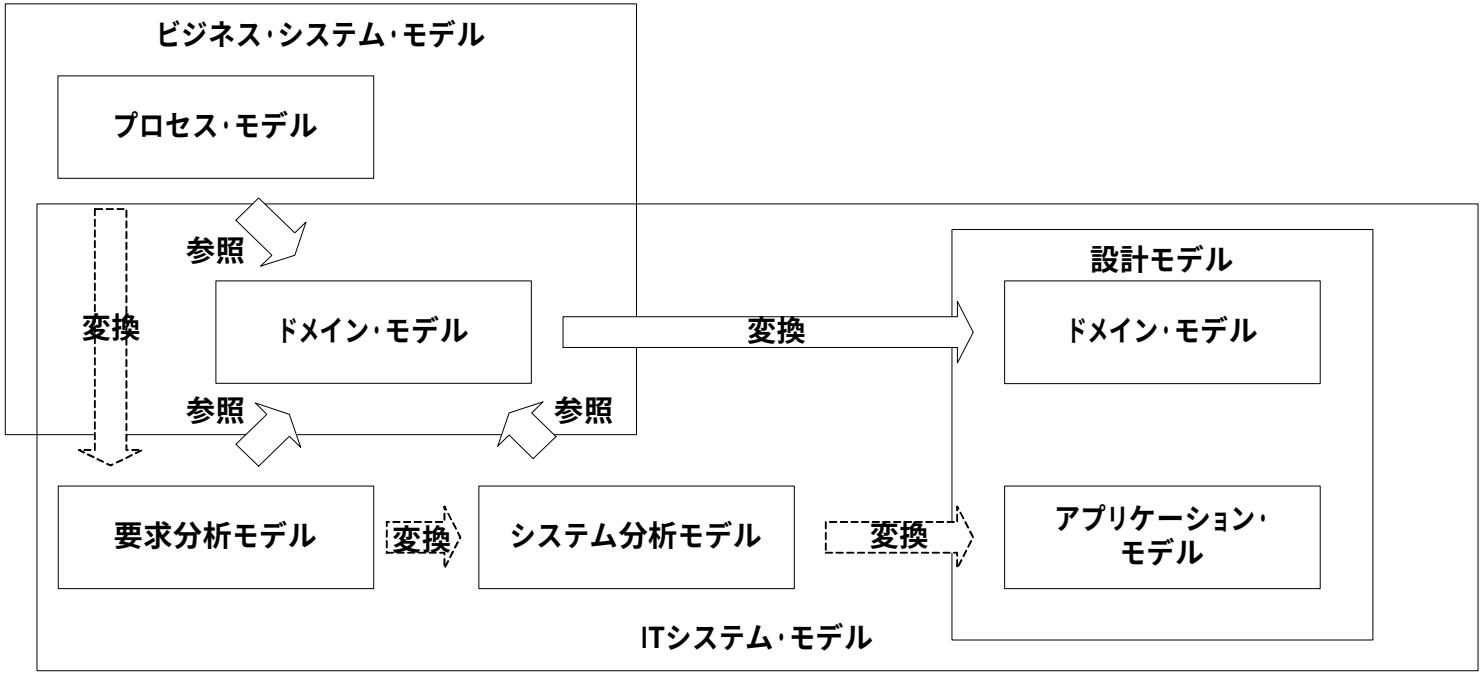
モデル体系リファレンス・モデル



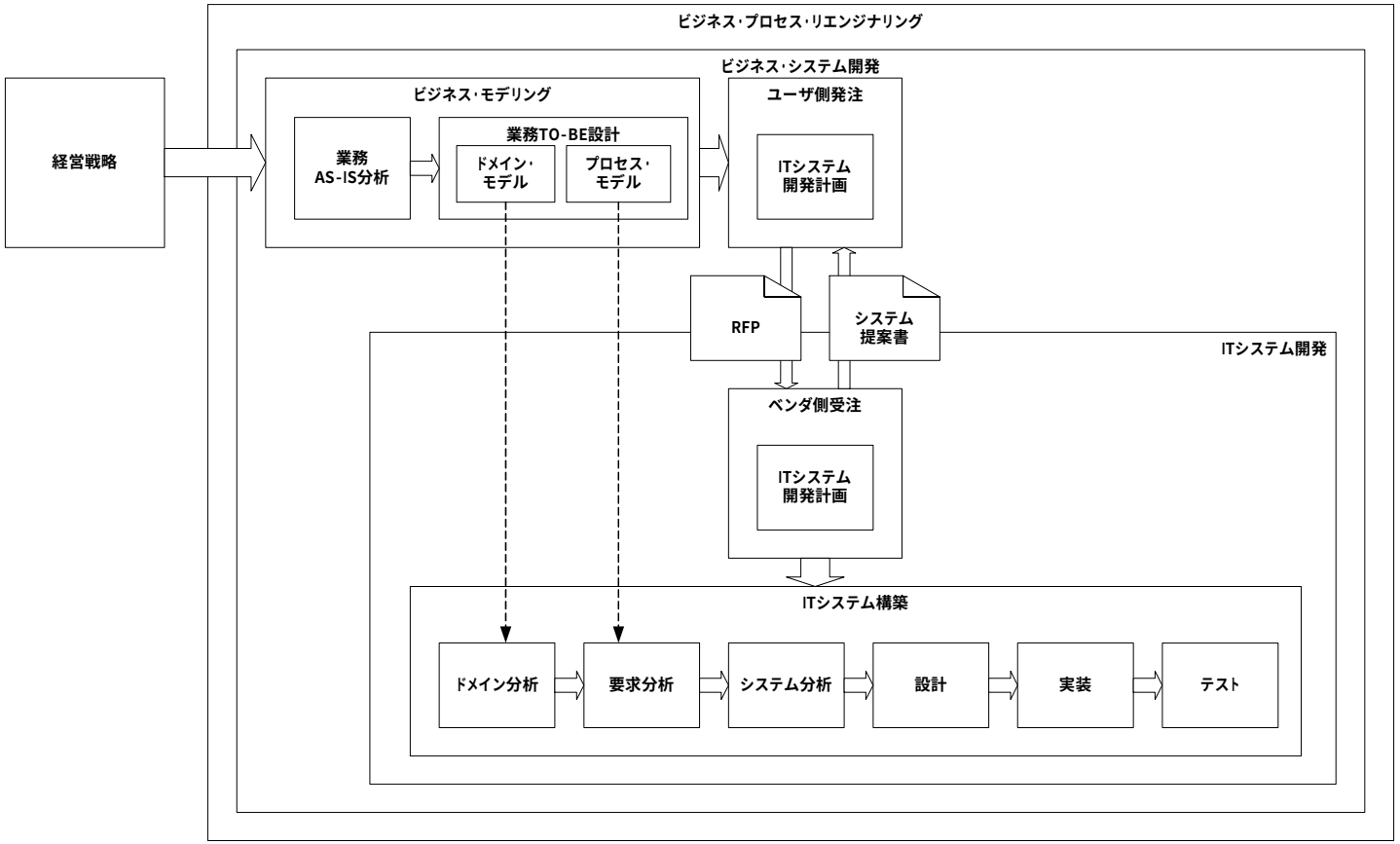
ビジネス･システム開発の階層



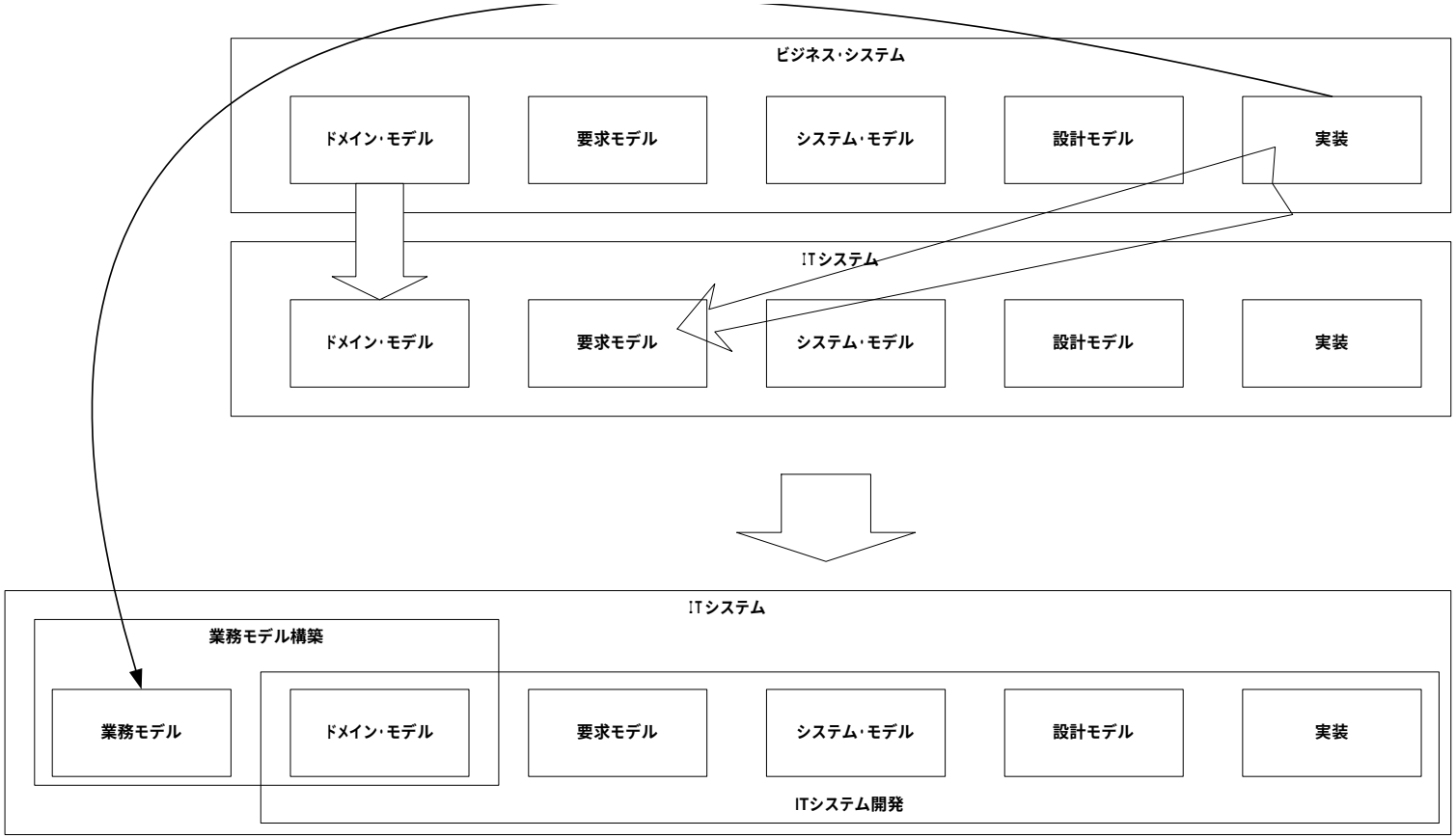
ビジネス・システム・モデルとITシステム・モデル



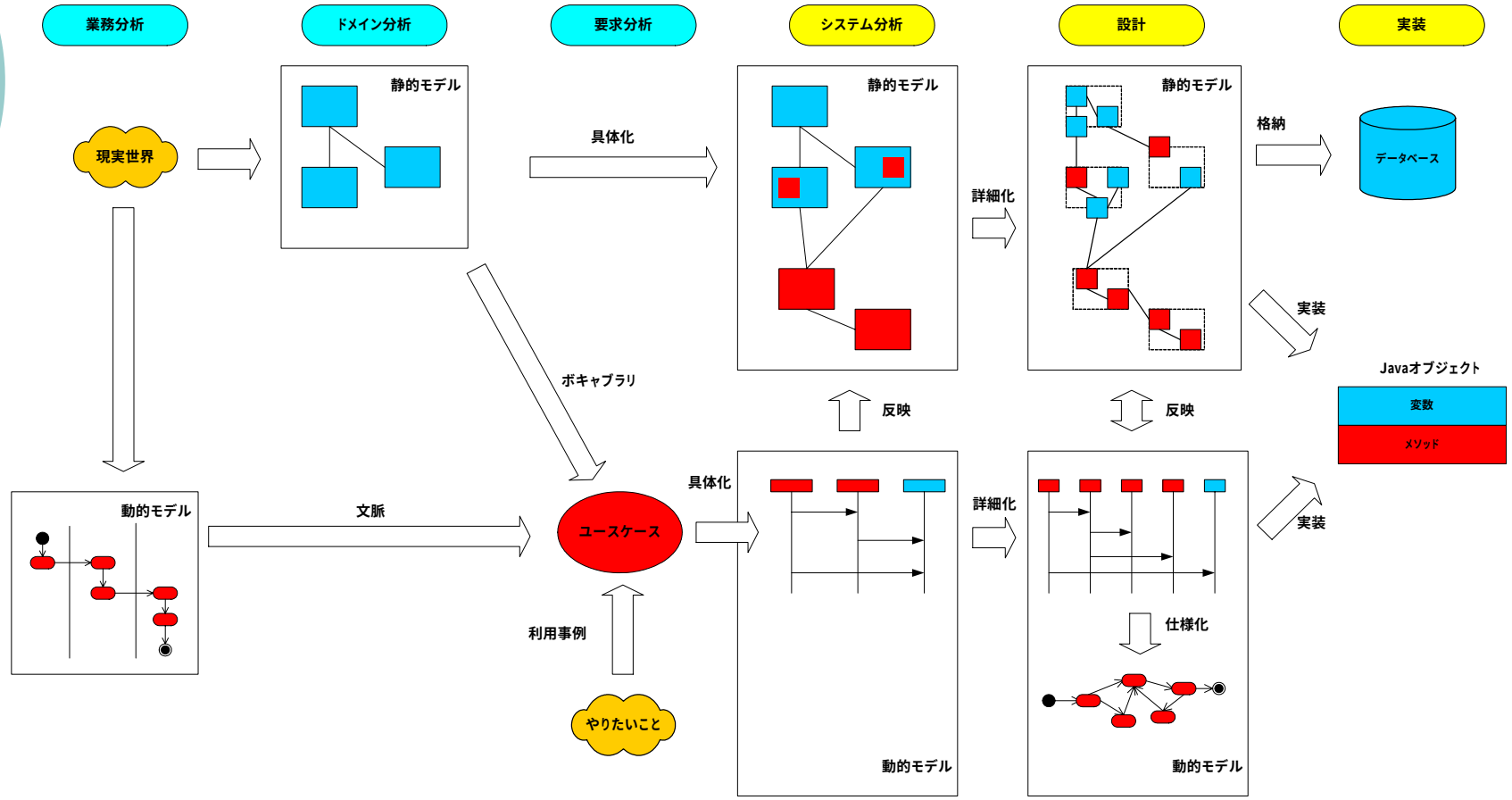
ビジネス・モデリングとITシステム開発の連携



開発手順の現実解


ビジネス・システム
ドメイン・モデル
要求モデル
システム・モデル
設計モデル
実装
ITシステム
ドメイン・モデル
要求モデル
システム・モデル
設計モデル
実装
ITシステム
業務モデル構築
業務モデル
ドメイン・モデル
要求モデル
システム・モデル
設計モデル
実装
ITシステム開発

開発の流れ


業務分析
ドメイン分析
要求分析
システム分析
設計
実装
現実世界
静的モデル
具体化
静的モデル
詳細化
静的モデル
格納
データベース
実装
ボキャブラリ
反映
反映
Javaオブジェクト
変数
メソッド
動的モデル
文脈
ユースケース
具体化
詳細化
実装
利用事例
仕様化
やりたいこと
動的モデル
動的モデル

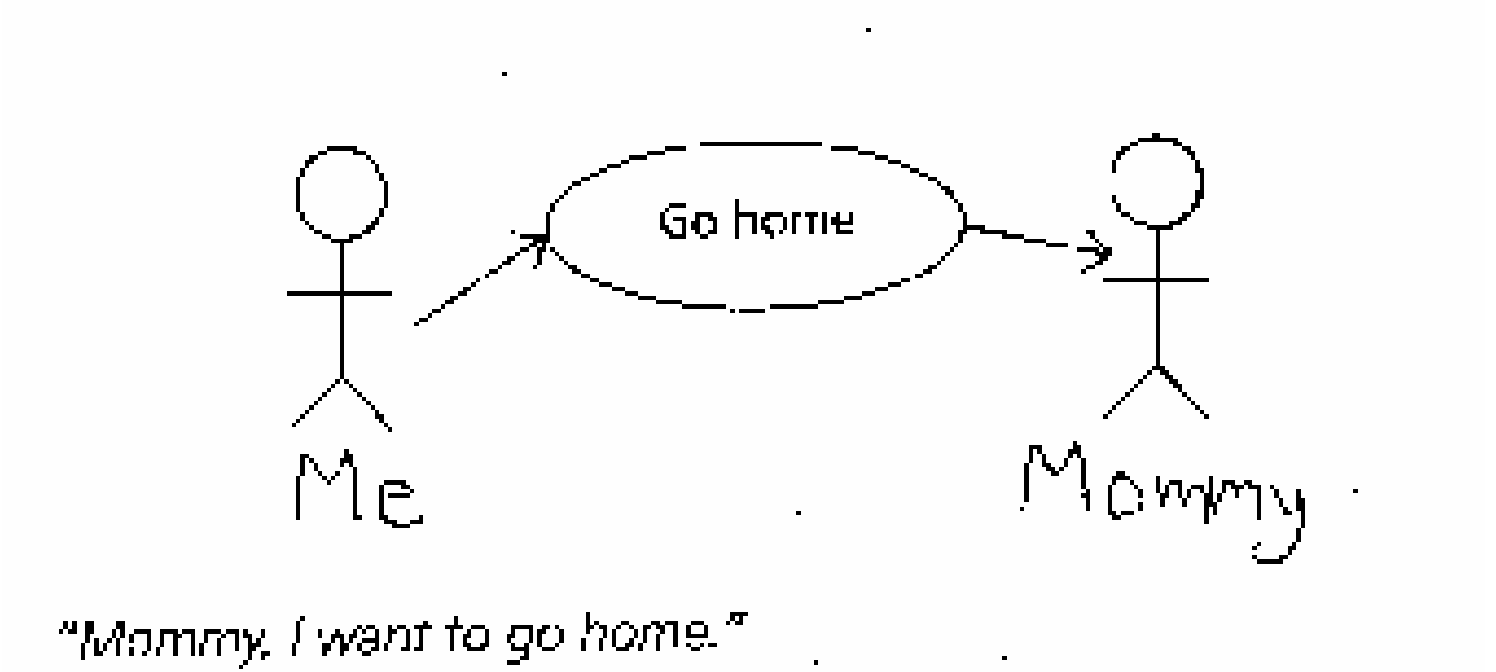
内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ

ユースケースとは

- 日本語訳”利用事例”
- アクターが目標を達成する利用事例の物語
  - アクターの要求を抽出することができる
- アクターとシステムのコラボレーション(協調)
  - システムの振舞いを抽出することができる
- システムの機能要求仕様
  - 脚本(イベント系列･シナリオ)が重要
  - ユースケース図は目次のようなもので、それほど重要ではない

Mommy, I want to go home

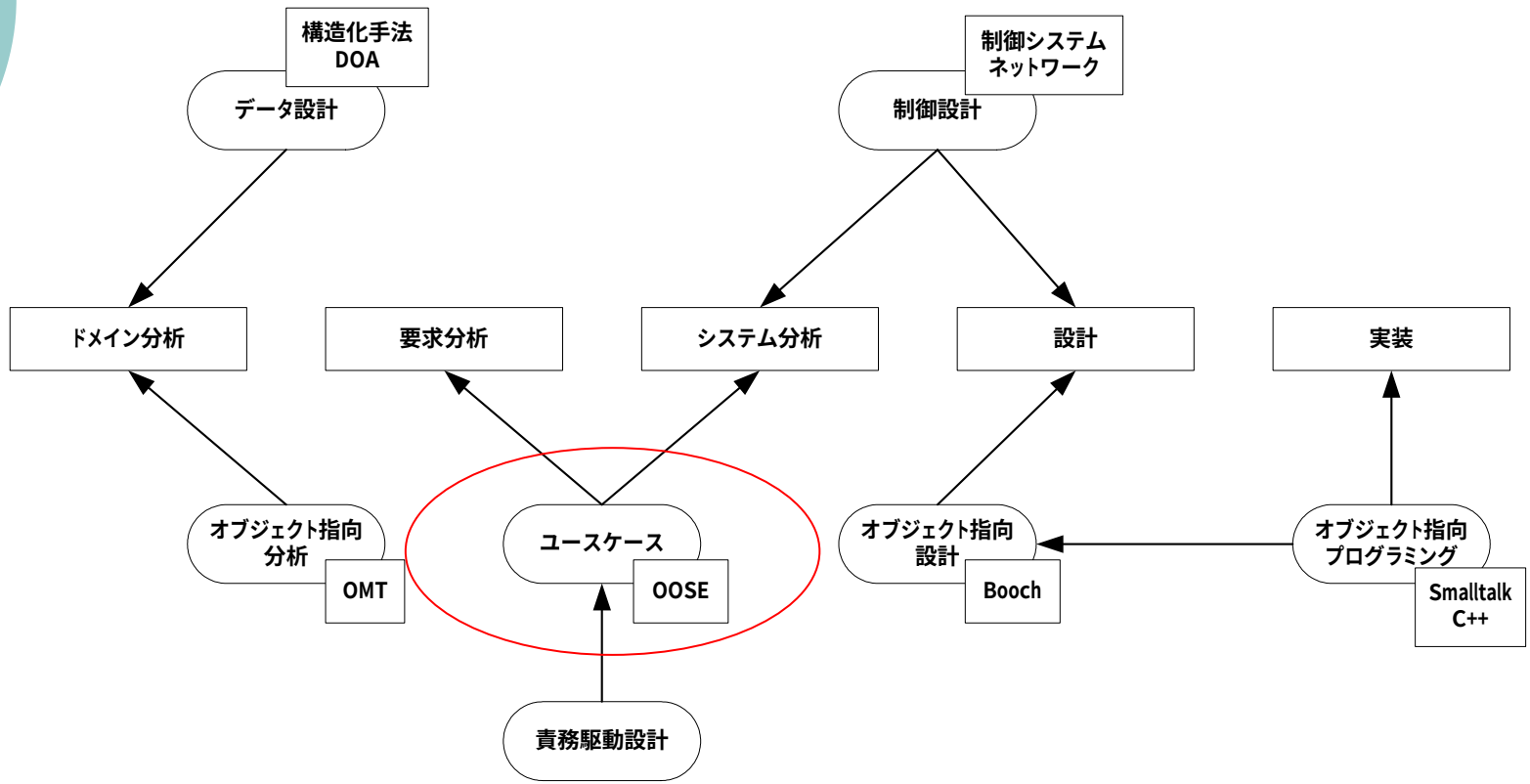


『Writing Effective Use Cases』より引用

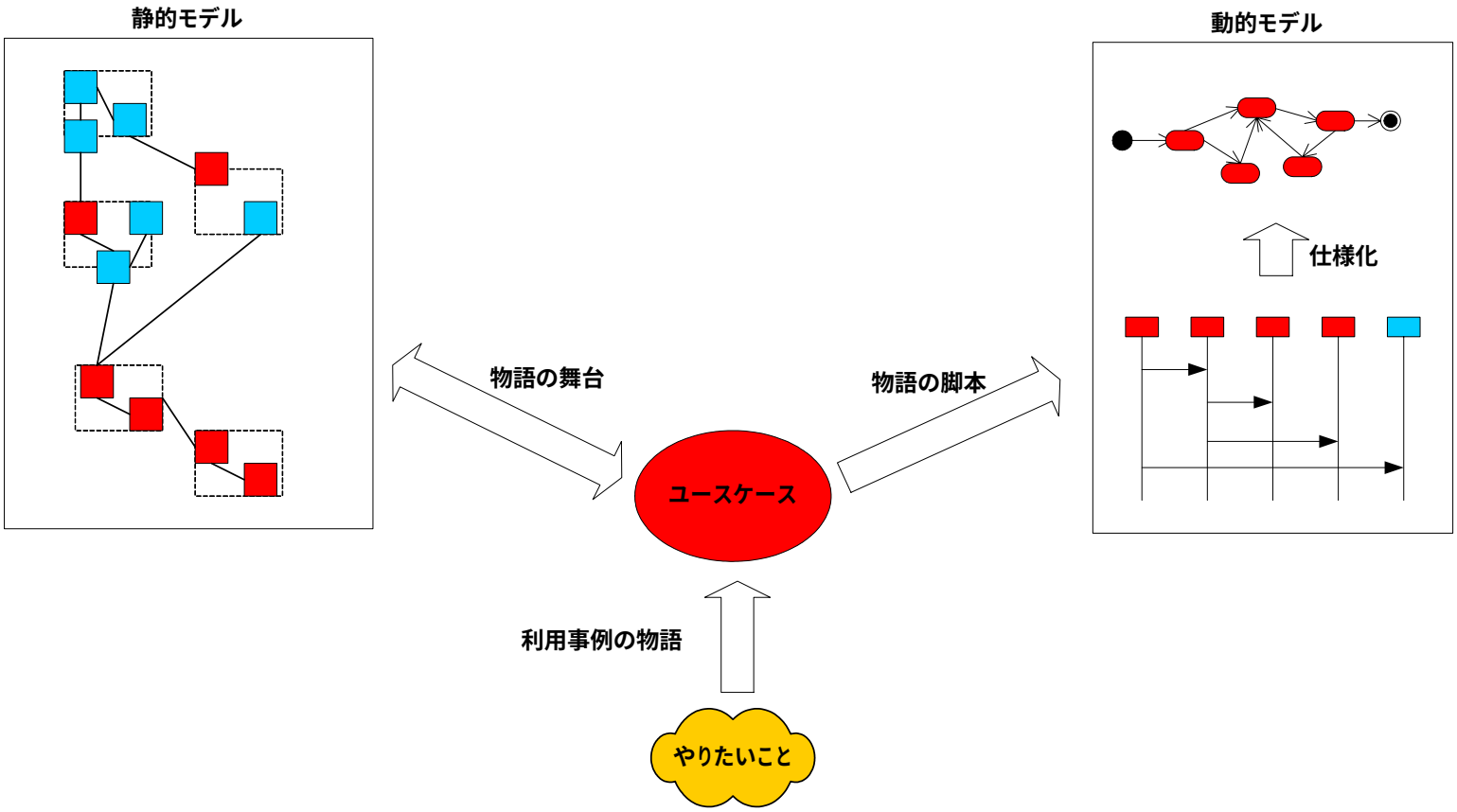
ユースケースを理解するコツ

ユースケース	物語
ユースケース名	タイトル
ユースケース概要	あらすじ
プライマリ・アクター	主役
セカンダリ・アクター	相手役、脇役
事前条件	物語開始前の状況
事後条件	物語終了時の状況
イベント系列	脚本
シナリオ	台本

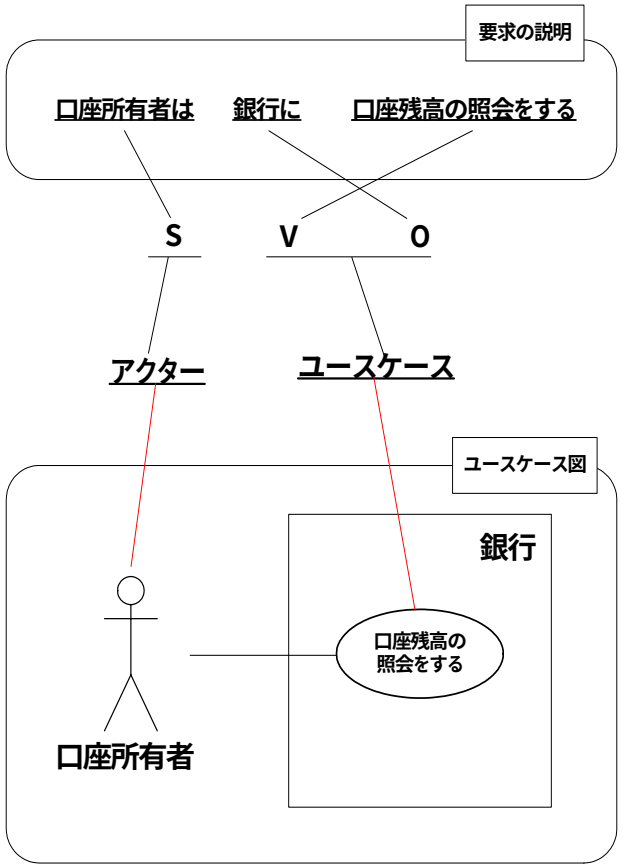
ユースケースはモデルを繋ぐハブ(1)



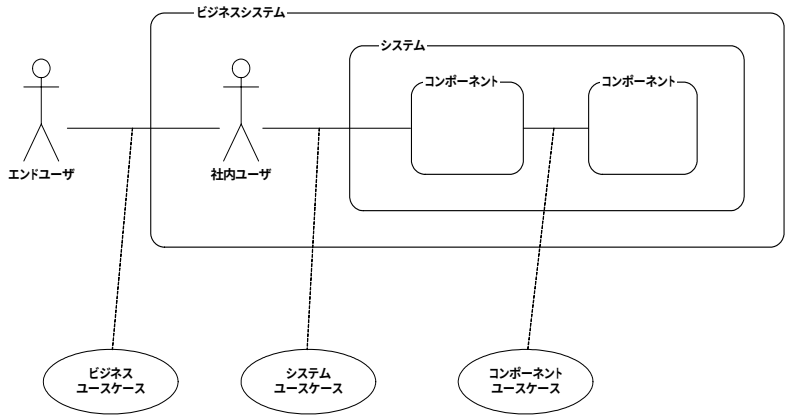
ユースケースはモデルを繋ぐハブ(2)



要求の説明とユースケース



ユースケースのレイヤ



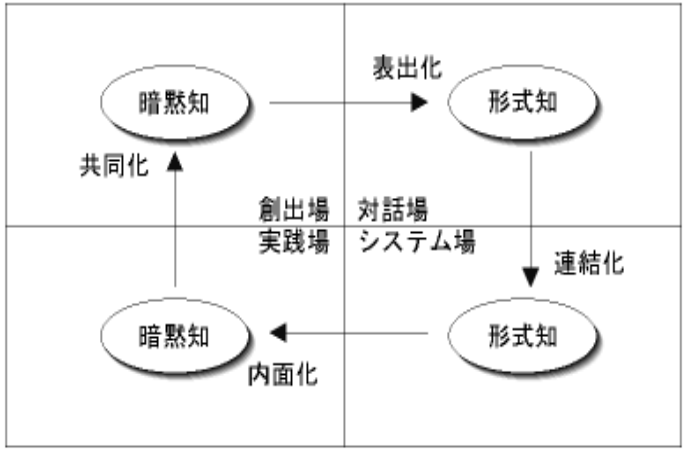
- ビジネス
  - 企業の目的
  - 大規模システムで補助的に用いられる
- システム
  - システムを使用するユーザをプライマリアクタとしたユースケース
  - 中心となるユースケース
- コンポーネント
  - システムの内部動作を記述したユースケース
  - 大規模システムで補助的に用いられる

ユースケースの例

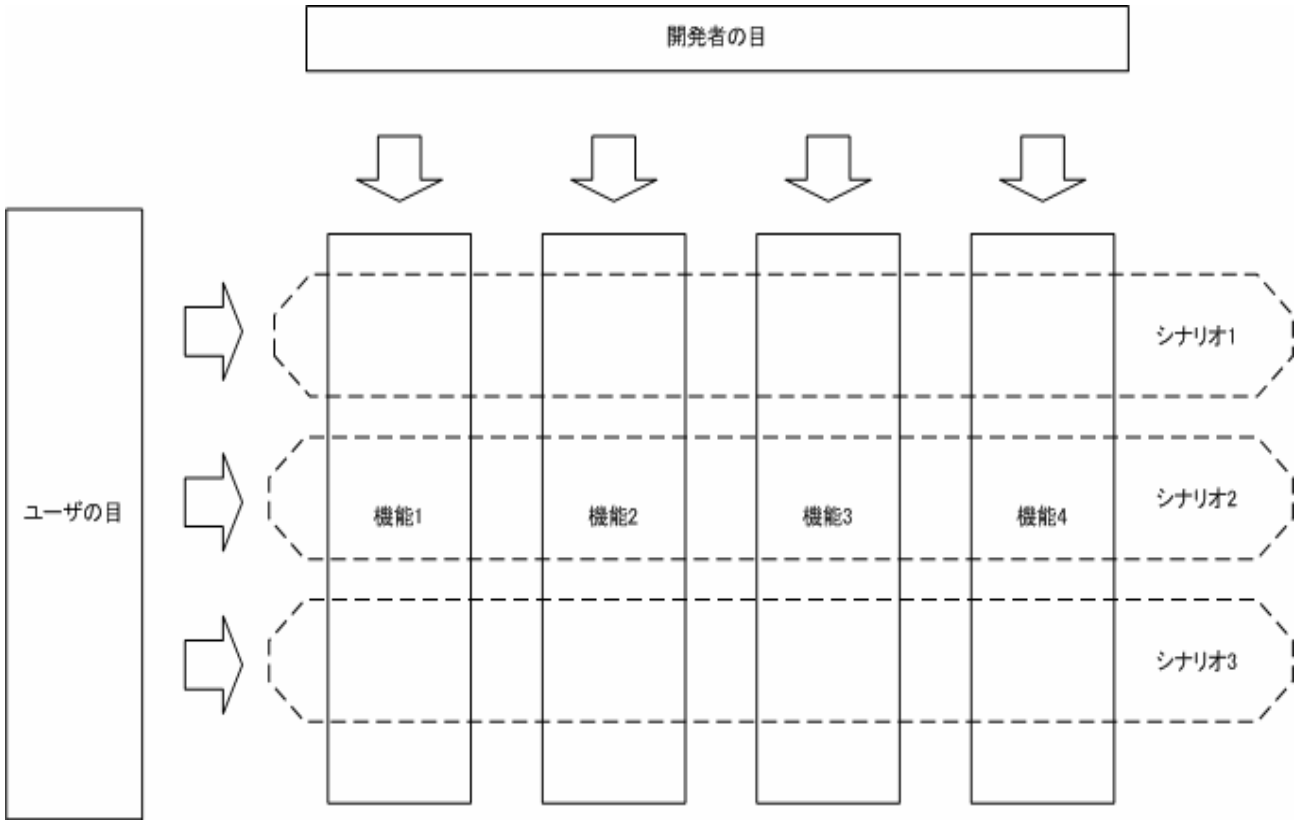
- ユースケース名:ATMから預金をおろす
- 基本パス
  - 1. 口座所有者がATMのカードスロットにカードを入れる
  - 2. 口座所有者がATMの入力画面に暗証番号を入れる
  - 3. ATMが口座に対して暗証番号の照会を行い正しい暗証番号であることを確認する
  - 4. ATMは口座所有者に引き出し金額の入力を促す
  - 5. 口座所有者はATMの入力画面に引き出し金額を入れる
  - 6. ATMが口座に預金残高の問合わせを行い十分な預金残高があることを確認する
  - 7. ATMの現金払い出し装置は口座所有者に現金を支払う
  - 8. ATMのカードスロットは口座所有者にカードを返却する
- 代替パス
  - 1. 口座所有者がATMのカードスロットにカードを入れる
  - 2. 口座所有者がATMの入力画面に暗証番号を入れる
  - 3. ATMが口座に対して暗証番号の照会を行ったが、誤った暗証番号であることが判明した
  - 4. ATMは口座所有者に暗証番号が誤りであったことを通知する
  - 5. ATMは口座所有者にカードを返却する

シナリオの役割1

- ユースケース技術で使用されるシナリオは暗黙値を形式知に表出化させるために、とても有効な技術。
- 要求仕様定義ワークフローは知識経営SECIモデルにおける対話場の活動として考えることができる。



シナリオの役割2



シナリオの役割3

- 設計との連動
  - 要求分析の次の作業分野であるシステム分析で、シナリオを使用した責務駆動設計の技法を利用したロバストネス分析
- 責務駆動設計 (Responsibility-Driven Design)
  - シナリオから、各オブジェクトに配置する責務を抽出し割り当てる設計方法
  - オブジェクト指向における動的モデルの基本
- ロバストネス分析
  - ユースケース記述のシナリオから、ロバストネス図を用いてシナリオを実現するオブジェクトの構造と振る舞いを抽出する

アクタとは何か

- システムの外部にあり、システムと相互作用するエンティティ
- システムが直接操作できないオブジェクト
- 人
  - ユーザ、顧客、取引先
- システム
  - 装置、外部システム
- 自然現象
  - 時間

アクタの種類(1)

- プライマリアクタ
  - ユースケースを起動するアクタ
  - ユースケースはプライマリアクタがシステムに求める目的を表現したもの
- サポーティングアクタ
  - ユースケースから呼ばれるアクタ
  - ユースケースを実現するために利用される人や外部システム

アクタの種類(2)

- ステークホルダ
  - ユースケースに対して利害関係を持っているアクタ
  - 通常ユースケース図には現れない
- インターナルアクタ
  - システム内でユースケースとコラボレーションするアクタ
  - 通常ユースケース図には現れない
  - ユースケースを複数レイヤに分けて作成する場合に登場
- SuD(System Under Discussion)
  - ユースケースが動作するシステム
  - ユースケースが動作する文脈
  - ユースケース図では「システム」として現れる

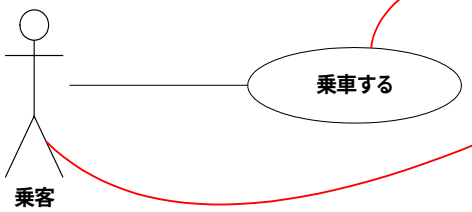
ユースケースのフォーマット

- 状況
  - UMLの仕様外
  - 方法論によってフォーマットはまちまち
- ユースケース名、基本フロー(main flow)、代替フロー(alternative flow)、例外フロー(exception flow)が最小構成
- アクター、前提条件、事後条件などを加えたものが基本構成

ユースケースのフォーマット

ユースケース名	乗車する	**ユースケース番号**	UC0001
プライマリアクター	乗客		
セカンダリアクター	N/A		
事前条件	シートに乗客は座っていない		
事後条件	シートに乗客が座っている		
基本フロー	1. 乗客がドアを開ける 2. 乗客が室内に入る 3. 乗客がドアを閉める 4. 乗客がシートに座る		
代替フロー	1. a. すでにドアが開いていたので、ドアは開けない 3. a. 後から乗ってくる人がいるのでドアは閉めない		
例外フロー	1. a. ドアの鍵が閉まっているのでドアを開けることができない		

ユースケース図とユースケース記述

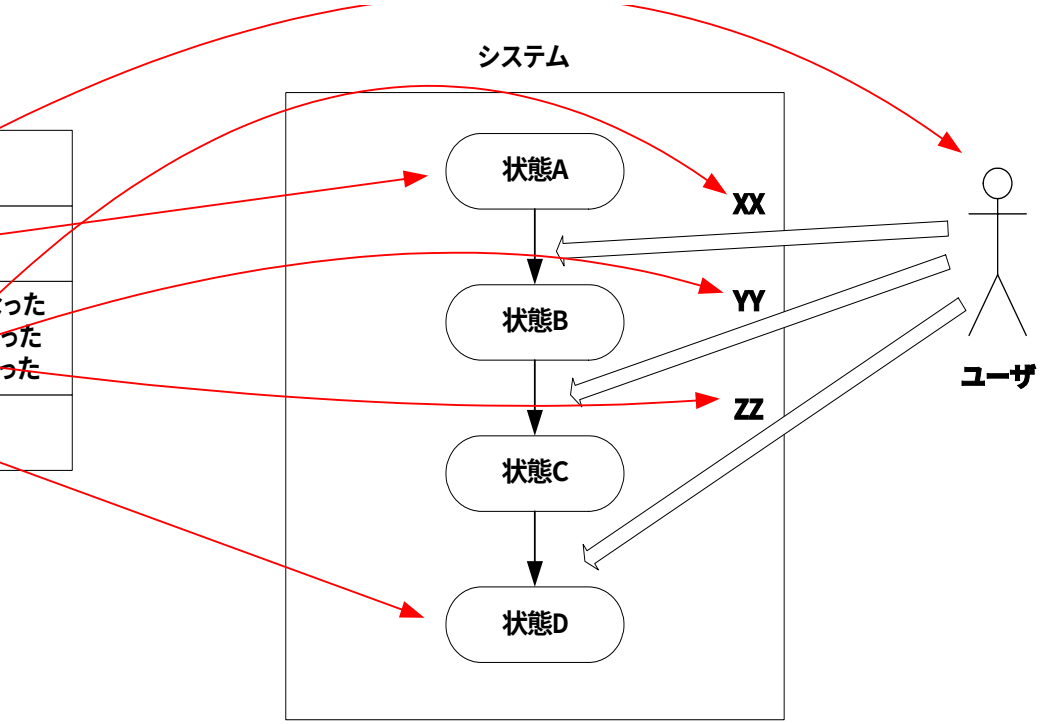


ユースケース名	乗車する	**ユースケース番号**	UC0001
プライマリアクター	乗客		
セカンダリアクター	N/A		
事前条件	シートに乗客は座っていない		
事後条件	シートに乗客が座っている		
基本フロー	1. 乗客がドアを開ける 2. 乗客が室内に入る 3. 乗客がドアを閉める 4. 乗客がシートに座る		
代替フロー	1. a. すでにドアが開いていたので、ドアは開けない 3. a. 後から乗ってくる人がいるのでドアは閉めない		
例外フロー	1. a. ドアの鍵が閉まっているのでドアを開けることができない		

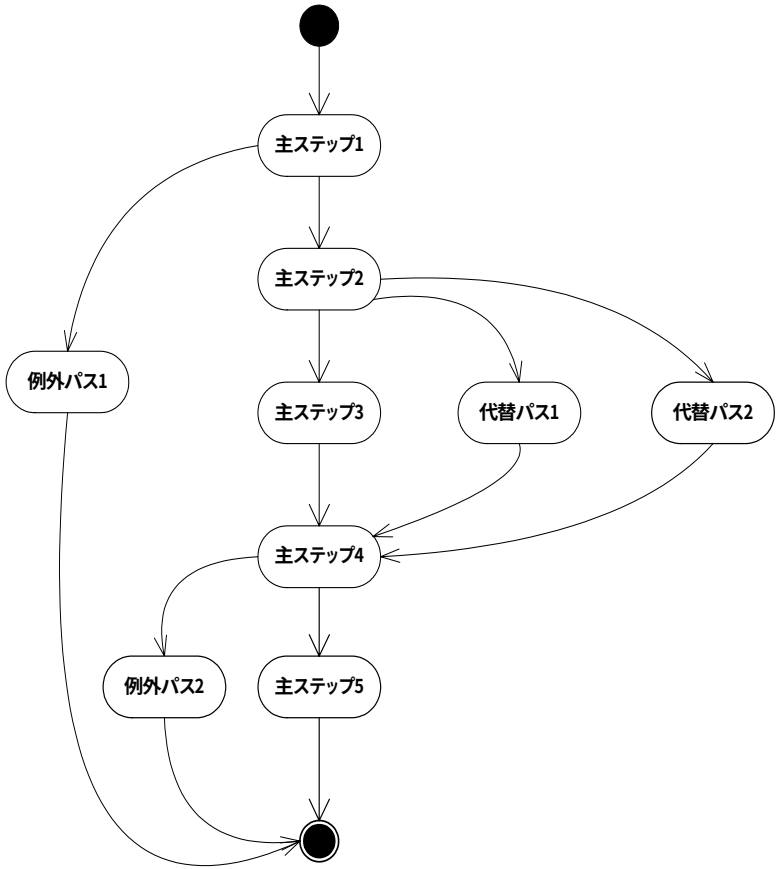
ユースケースとシステムの状態

ユースケース

アクター	ユーザ
事前条件	状態A
基本フロー	1. ユーザがＸＸして状態Bになった 2. ユーザがYYして状態Cになった 3. ユーザがZZして状態Dになった
事後条件	状態D



ユースケース･フローの記述方法

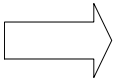


基本フロー	1. 主ステップ1 2. 主ステップ2 3. 主ステップ3 4. 主ステップ4 5. 主ステップ5
代替フロー	主ステップ3で代替パス1でもよい 主ステップ3で代替パス2でもよい
例外フロー	主ステップ1でXXしたら例外パス1 主ステップ4でYYしたら例外パス2

ユースケースとテストケース

ユースケース

ユースケース名	ユースケースABC
アクター	ユーザ
入力データ	データAA
事前条件	状態A
基本フロー	1. ユーザがデータAAをＸＸして状態Bになった 2. ユーザがYYして状態Cになった 3. ユーザがZZして状態Dになりデータ BBを通知
事後条件	状態D
出力データ	データBB



テストケース

テストケース名	テストケースABC
準備	入力データAAを用意
	システムを状態Aにする
テスト手順	1. ＸＸを実行 (状態Bを確認) 2. YYを実行 (状態Cを確認) 3. ZZを実行
確認	システムが状態Dに なっていることを確認
	出力データBBを確認

ユースケースの誤用

- フローチャート
  - ユースケースはアクションを表すものではなく…
  - 利用者(主役)が目標を達成するためのシステムの利用事例の物語
  - 主役と脇役とシステムのコラボレーション(協調)
- 機能一覧
  - ユースケースはシステムが提供する機能ではなく…
  - 利用者(主役)が目標を達成するためのシステムの利用事例の物語
  - 主役と脇役とシステムのコラボレーション(協調)
- 画面遷移
  - ユースケースは入出力画面ではなく…
  - 利用者(主役)が目標を達成するためのシステムの利用事例の物語
  - 主役と脇役とシステムのコラボレーション(協調)

内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ
  - ユースケースから論理コンポーネントへ
  - 論理コンポーネントからEJB&SOAへ

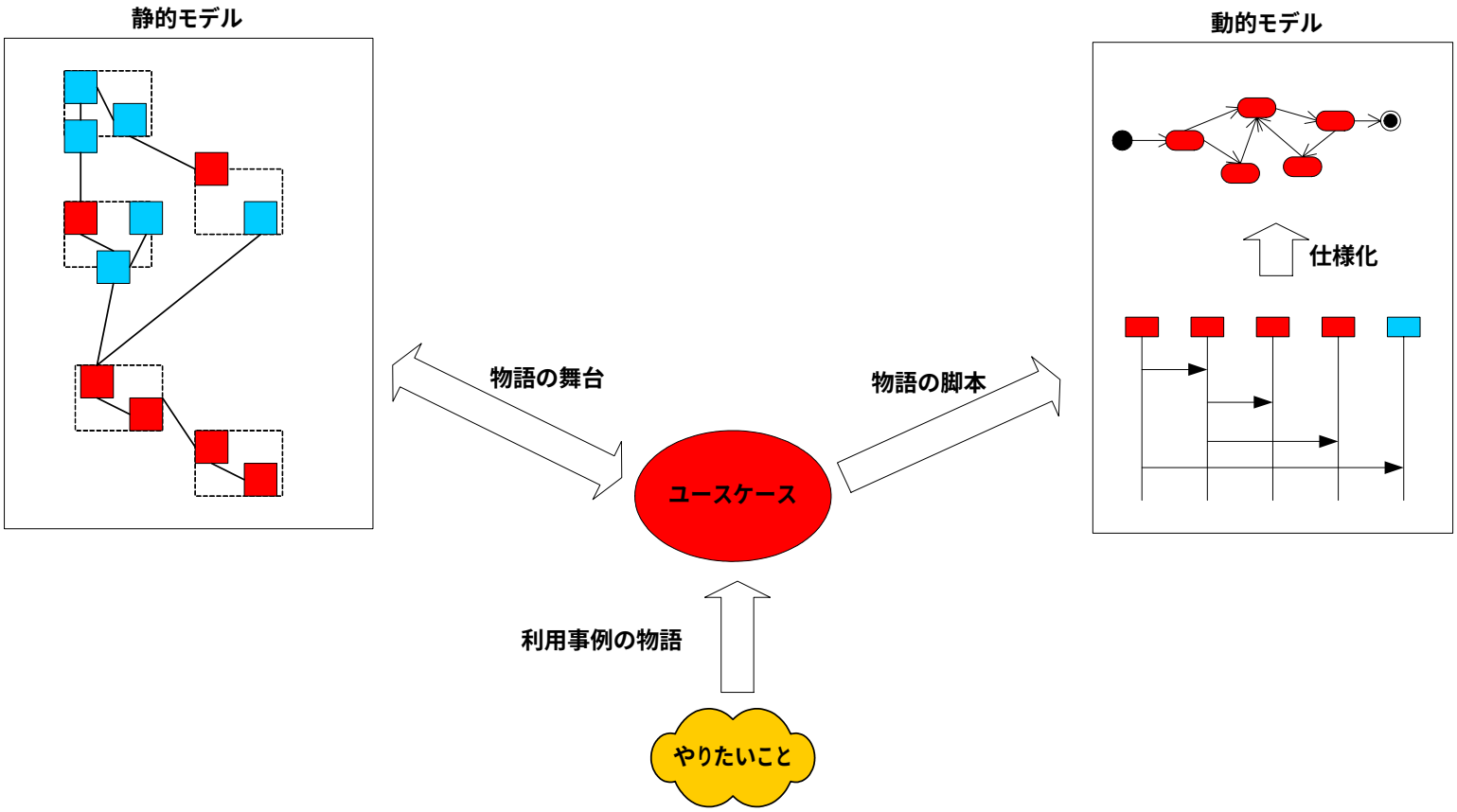
内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ
  - ユースケースから論理コンポーネントへ
  - 論理コンポーネントからEJB&SOAへ

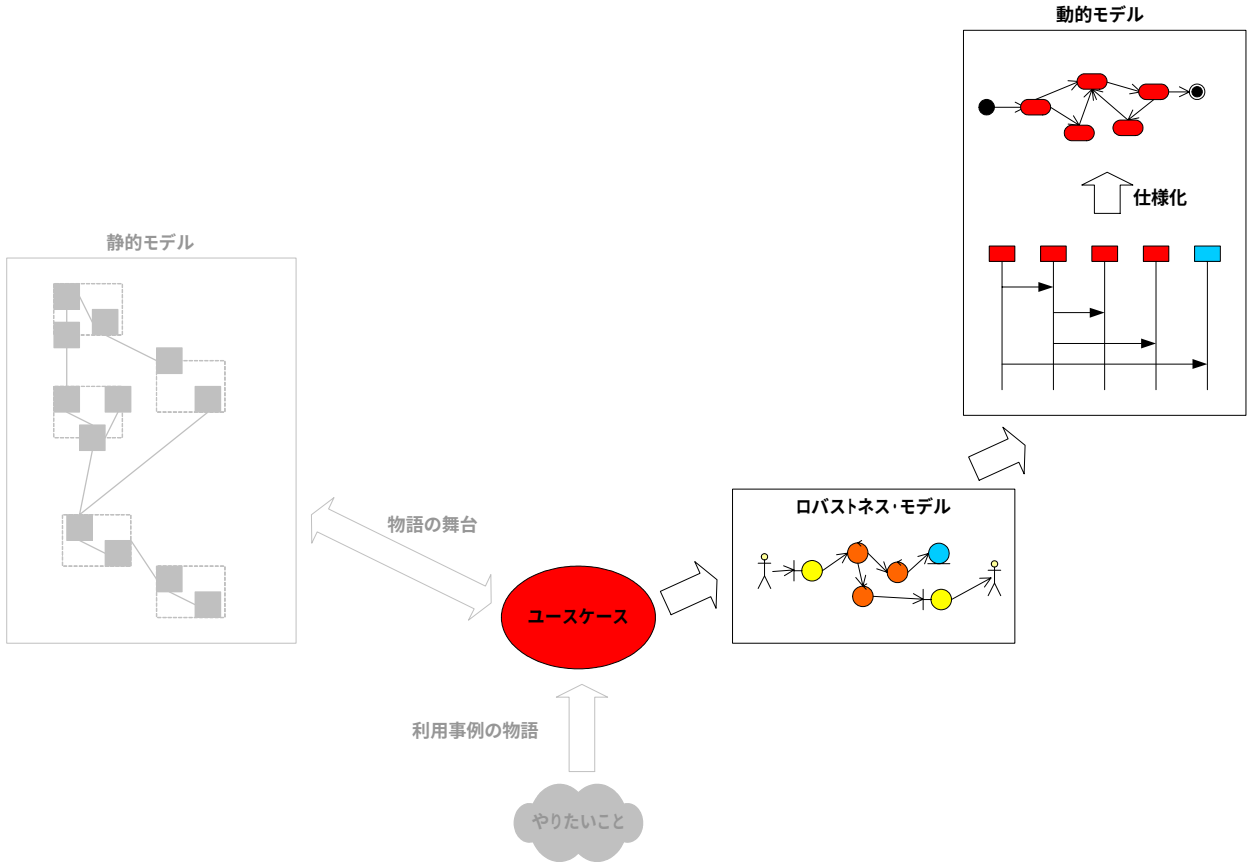
ユースケースとオブジェクト・モデル

- ユースケース
  - 要求仕様を”物語”として記述したもの
- 静的モデルとの連携
  - ドメイン・モデル(用語集)を脚本の語彙として用いる
- 動的モデルとの連携
  - 脚本からコラボレーションを責務駆動設計の技法を用いて抽出する
  - ⇒ロバストネス分析

ロバストネス分析の役割(1)



ロバストネス分析の役割(2)

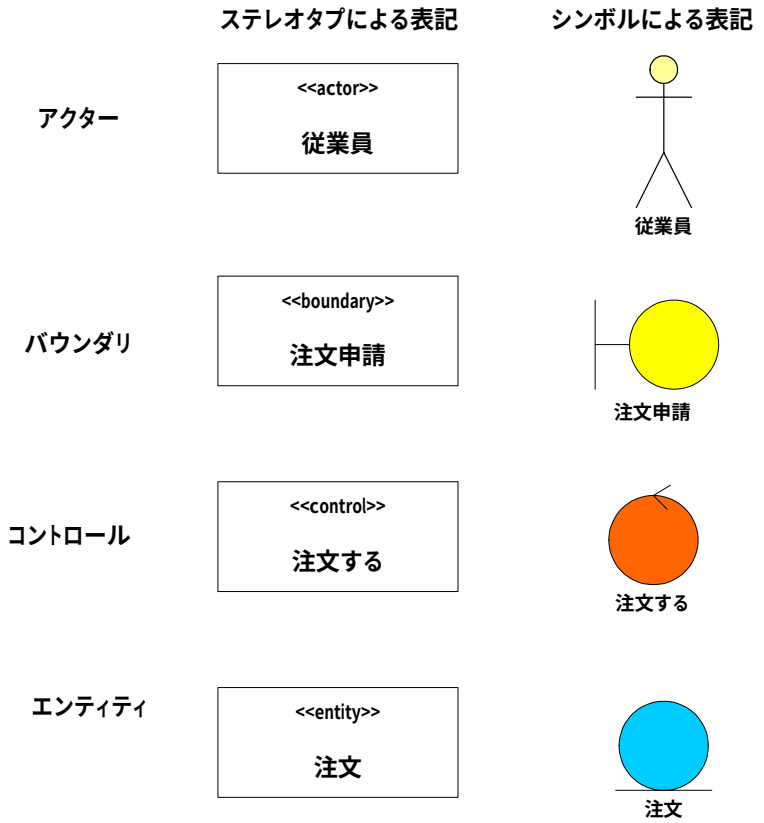


ロバストネス分析

- 本格的な分析モデルを作成するための予備モデル
- ユースケースのフロー(シナリオ)から変化に強いシステム構造を抽出する
  - 責務駆動設計の技法を用いる
- 4つの分析オブジェクト
  - アクター、バウンダリ、コントロール、エンティティ

ロバストネス・モデルの構成要素

- **アクター** (actor)
  - システム外部にあるオブジェクト
- **バウンダリ・オブジェクト** (boundary object)
  - アクターと通信するオブジェクト
- **コントロール・オブジェクト** (control object)
  - システムの責務を表現するオブジェクト
- **エンティティ・オブジェクト** (entity object)
  - (システムの管理下にある)ドメイン・オブジェクト(現実世界)に対応するオブジェクト



ロバストネス分析の考え方

- ドメインモデルとユースケースから、オブジェクトによるシステムを構築する
  - オブジェクトを抽出する
  - オブジェクト間の関係を抽出する
  - オブジェクト間の相互作用を抽出する
- システムの理想的なオブジェクトモデルを抽出する
  - 実装非依存の本質的なオブジェクトモデル
  - 変化に強いオブジェクトモデル

変化に強いオブジェクトモデル

- 機能の種類によって変化の度合に違いがある
  - ユーザインタフェース
    - 頻繁に変更がある
    - 異なったユーザインタフェースが必要となるケースもある(例:GUI，Web)
  - 機能
    - 仕様追加や仕様変更は比較的多い
  - データ
    - 比較的安定している
    - さまざまな機能間での共有ができる
- 「ユーザインタフェース」は分離した方が明らかに良い
- 「機能」と「データ」を1つのオブジェクトにまとめると、仕様変更の多い「機能」にひきづられて、オブジェクト仕様が安定しない
  - 「機能」と「データ」を分離することで、「機能」の仕様変更が「データ」に影響しなくなる
  - 純粋な「データ」として扱うことで、さまざまな「機能」間での共有が行いやすくなる

バウンダリ・オブジェクト

- アクター(システム外のオブジェクト)との通信を行うオブジェクト
- 設計時には：
  - ユーザインタフェース
    - CLI (Command Line Interface)
    - CUI (Character User Interface)
    - GUI (Graphical User Interface)
    - Web UI (Web User Interface)
  - 外部システム
    - API
    - 通信プロトコル

コントロール・オブジェクト

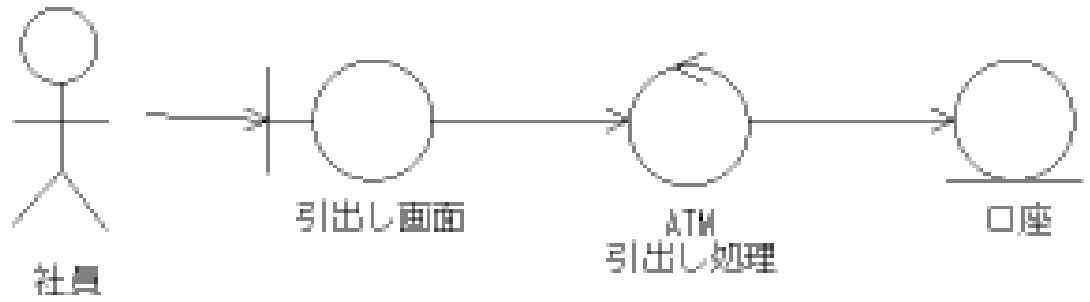
- システムの振舞いを表現するオブジェクト
- アプリケーション・ロジックを抽象化
- 設計時の実現方法はケースバイケース
  - 必ずしも1コントロールオブジェクトが設計時の1クラスにマップされるわけではない点に注意
    - 複数のコントロールオブジェクトをまとめて1オブジェクト
    - バウンダリオブジェクトに実装
    - エンティティオブジェクトに実装

エンティティ・オブジェクト

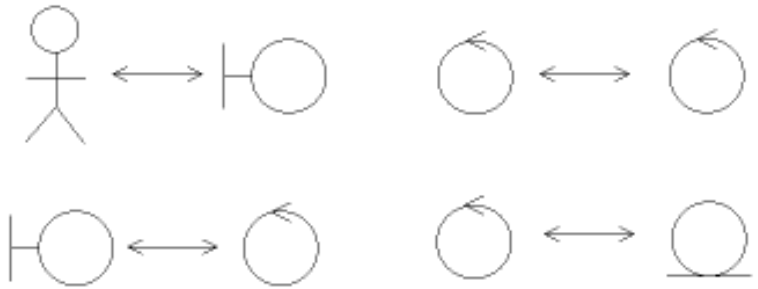
- 現実世界を抽象化したオブジェクト
- ドメイン分析における静的構造からの流れ
  - ドメイン分析の結果抽出されたオブジェクトであり安定している
    - ユースケースのライフサイクルを超えて長期的に存在し続けるオブジェクト
    - 設計時には、主にデータベースに格納されるデータとして実現

ロバストネス図の描き方

- 設計に入る前の予備設計
- 3つのオブジェクトの抽出が目的
- 3つのオブジェクト間の関係を抽出
- 特に外部アクタとの接点になるバウンダリオブジェクトと、データベースに格納するデータの候補であるエンティティオブジェクトが重要
- コントロールオブジェクトは、システムの機能や振る舞いのためのプレースホルダ



分析オブジェクト間の通信



- アクターはバウンダリとしか通信しない
- エンティティはコントロールとしか通信しない

主語(S)	**動詞**(V)	**目的語**(O)
バウンダリ	コントロール	エンティティ
バウンダリ	コントロール	バウンダリ
コントロール	コントロール	エンティティ
コントロール	コントロール	バウンダリ
コントロール	コントロール	コントロール

アーキテクチャ分析

- ロバストネス・モデルを基にして、論理コンポーネントによって構成される論理アーキテクチャを作成する。
- 論理コンポーネントは、UMLのコンポーネント図レベルの抽象モデル。
- 論理コンポーネントの外部仕様を定義する。

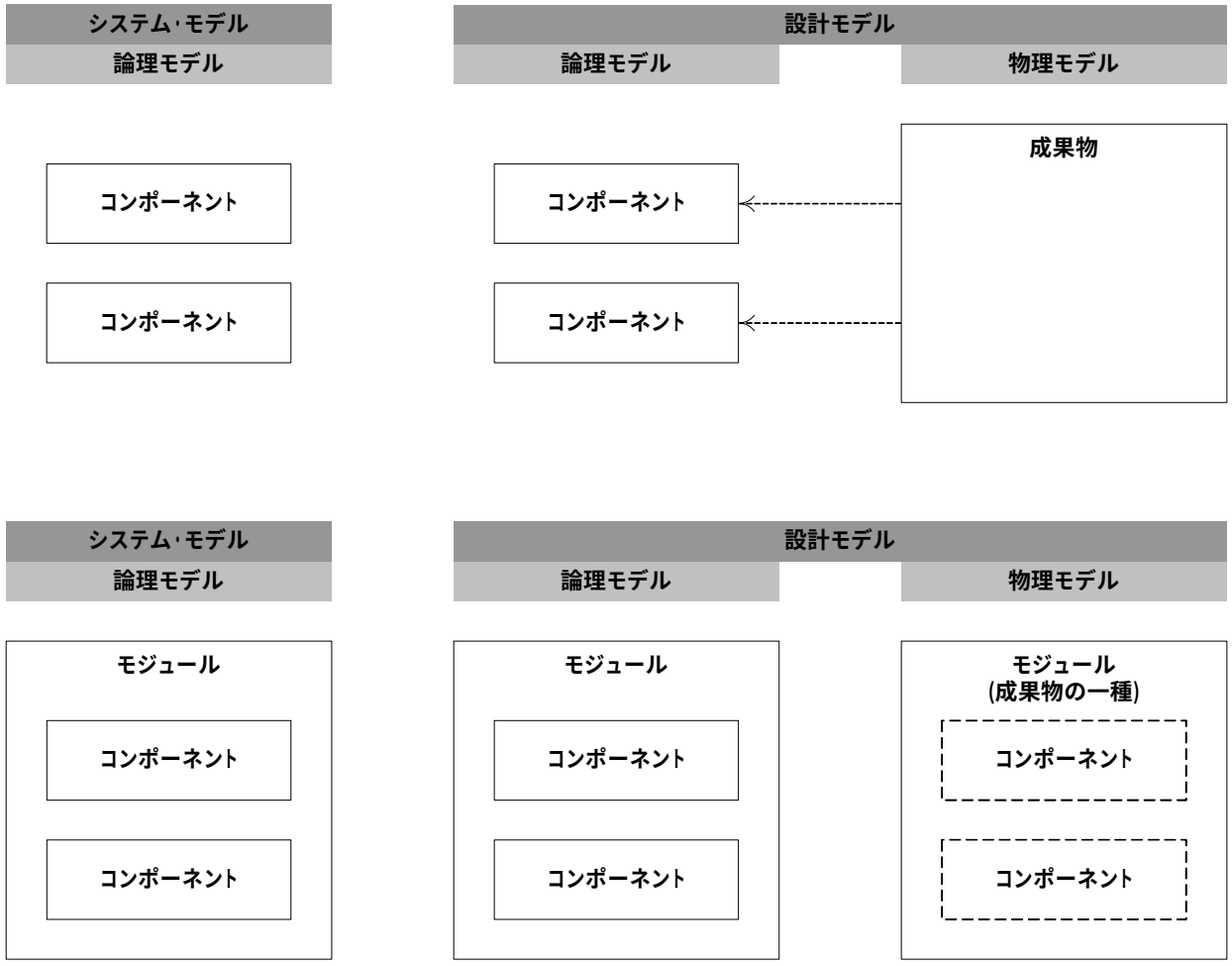
アプリケーション・アーキテクチャ

- プレゼンテーション層
  - 人間(＋その他の外部システム)にUI(＋その他のインタフェース)を提供する層
- アプリケーション層
  - アプリケーション・ロジックを提供する層
- ドメイン層
  - アプリケーションから操作するドメイン・オブジェクトを提供する層
- 統合層(永続層)
  - ドメイン層を外部リソース(データベース、ERP、他システム)と統合するための層
  - データベースを対象とする場合には永続層と呼ぶ

ソフトウェア部品を表現するモデル要素

- **コンポーネント** (component)
  - UMLで置換可能なソフトウェア部品を示す分類子
  - システム・モデル、設計モデルで利用
- **成果物** (artifact)
  - UMLで物理的な構成要素を示す分類子
  - 設計モデルで利用
- **モジュール** (module)
  - UMLでは用いられていないモデル要素
    - 本講義(SimpleModeling)の提案
  - 論理モデルの中でコンポーネントを束ね、物理モデルの成果物に対応する
  - システム・モデル、設計モデルで利用

モジュール、コンポーネント、成果物



コンポーネント

- UI Component
  - UIを実現するコンポーネント
- Service Component
  - 他システムに提供するサービスを実現するコンポーネント
- Application Component
  - アプリケーション・ロジックを実現するコンポーネント
- Entity Component
  - ドメイン・オブジェクトを実現するコンポーネント
- Comunication Component
  - 外部リソースの代理オブジェクトを実現するコンポーネント

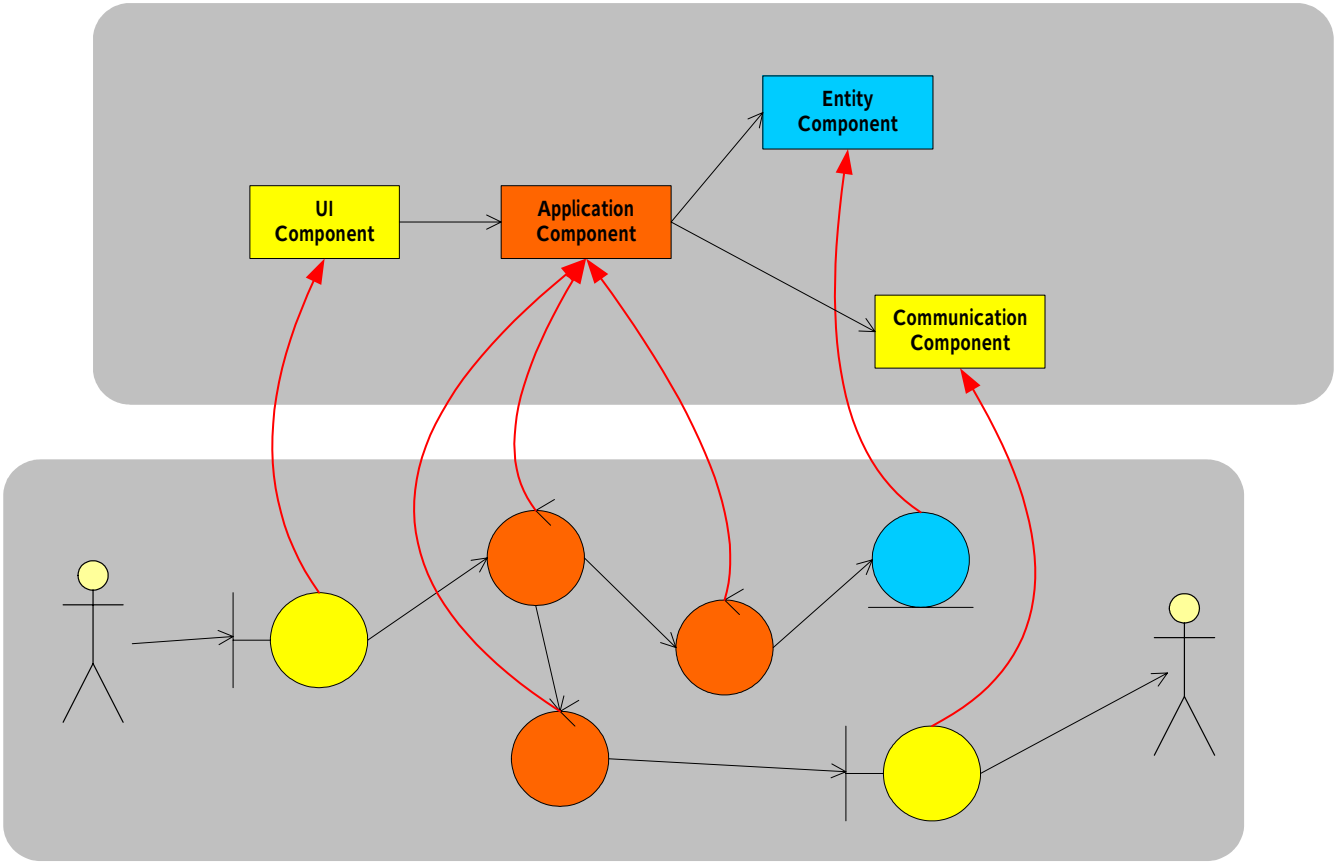
モジュール

- Presentation Module
  - UI Component
  - Service Component
- Application Module
  - Application Component
- Domain Module
  - Entity Component
  - Comunication Component

ロバストネス・モデルからアーキテクチャ・モデルへの変換

- プライマリ・アクターから操作されるバウンダリ・オブジェクト
  - プライマリ・アクターが人間の場合
    - ⇒UI Component
  - プライマリ・アクターがシステムの場合
    - ⇒Service Component
- コントロール・オブジェクト
  - ⇒Application Component
- エンティティ・オブジェクト
  - ⇒Entity Component
- セカンダリ・アクターを操作するバウンダリ・オブジェクト
  - ⇒Commnication Component

ロバストネス・モデルからアーキテクチャ・モデルへの変換



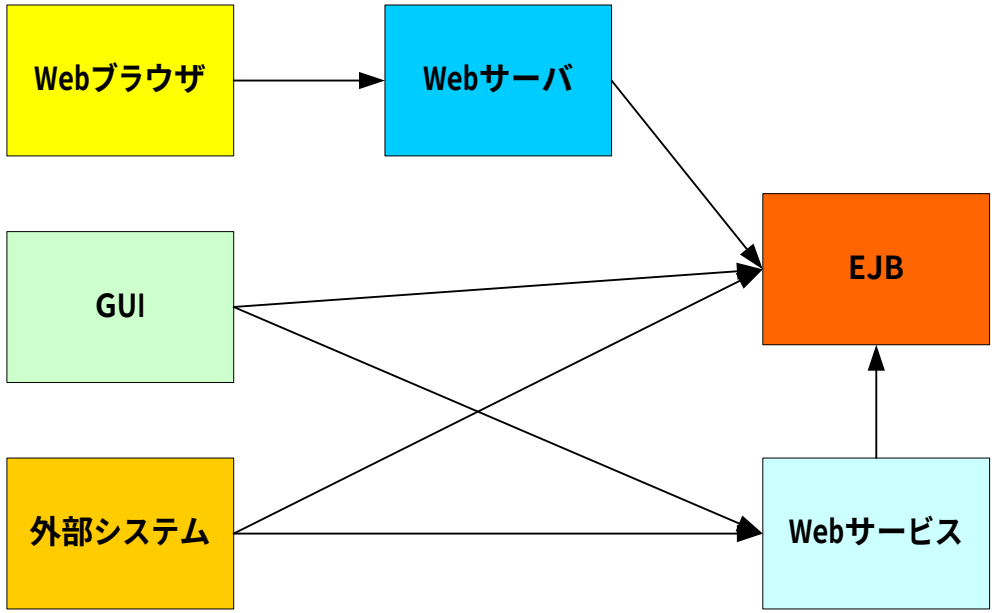
内容

- モデリング
- ユースケース
- ユースケースからEJB、SOAへ
  - ユースケースから論理コンポーネントへ
  - 論理コンポーネントからEJB&SOAへ

SOAとは

- Service Oriented Architecture
  - オブジェクトでもなくコンポーネントでもなくサービス
- ビジネスとの関係
  - ビジネス･プロセス
- 実現方法
  - BPEL
  - Webサービス

EJB&Webサービスのシステム･アーキテクチャ



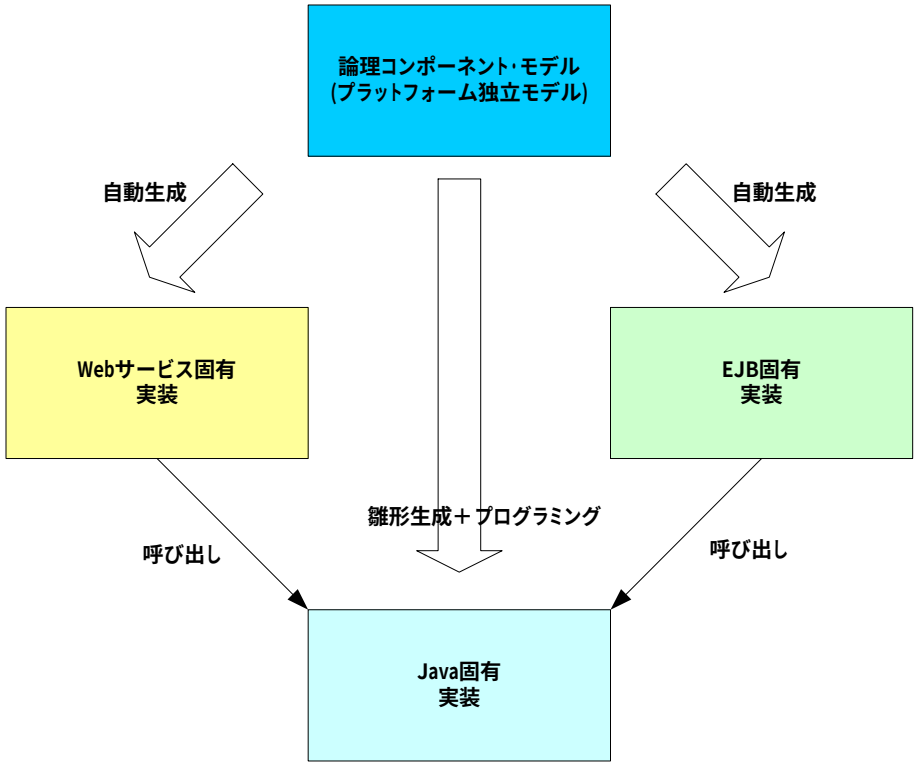
Javaでの実現・現状

- 色々なプラットフォームが存在
- プラットフォームの例(オープンソースを中心として)
  - JAX-WS
    - Glassfishとの連携
  - Apache Axis
    - 広く利用されている
  - JBoss WS
    - これから…
- サービス・インタフェースからプラットフォーム固有部分を自動生成するものが多い
  - WSDL(Web Service Description Language)から生成
  - Java(POJO-Plain Old Java Object)から生成

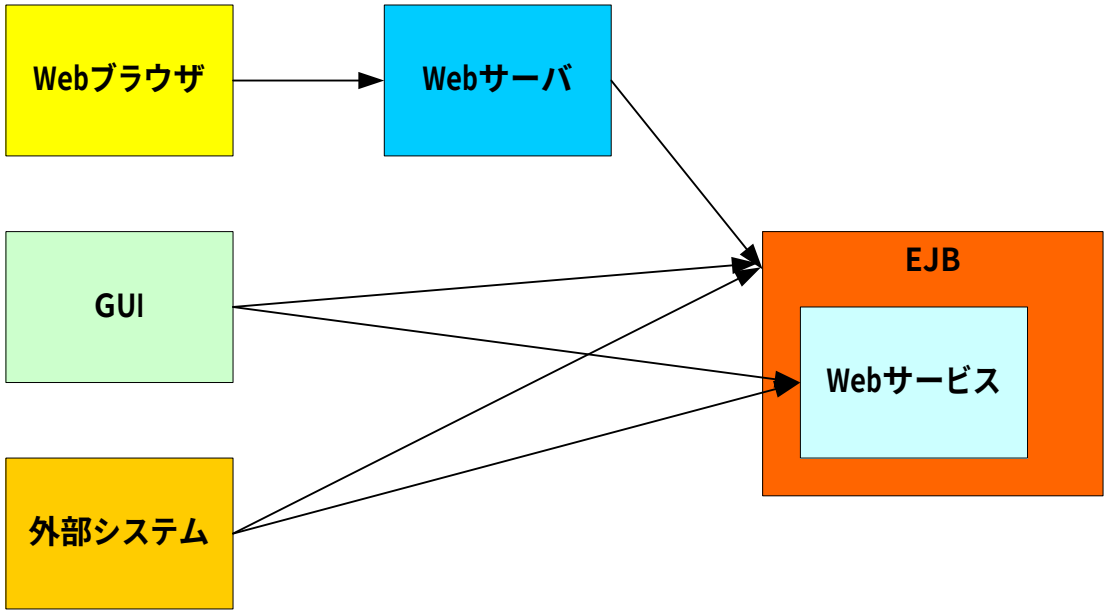
Javaでの実現・最終形

- 実現方法1：プラットフォーム独立モデルからプラットフォーム固有部分を自動生成
  - WSDLやJava(POJO)から自動生成するものとは、対象モデルの抽象度が異なる
- 実現方法2：EJBでハンドリング
  - EJBを開発すると、自動的にWebサービスとしても機能
  - XML系の技術も開発中に意識する必要はある

実現方法1:自動生成


論理コンポーネント・モデル
(プラットフォーム独立モデル)
自動生成
自動生成
Webサービス固有
実装
EJB固有
実装
雛形生成＋プログラミング
呼び出し
呼び出し
Java固有
実装

実現方法2:EJBでハンドリング



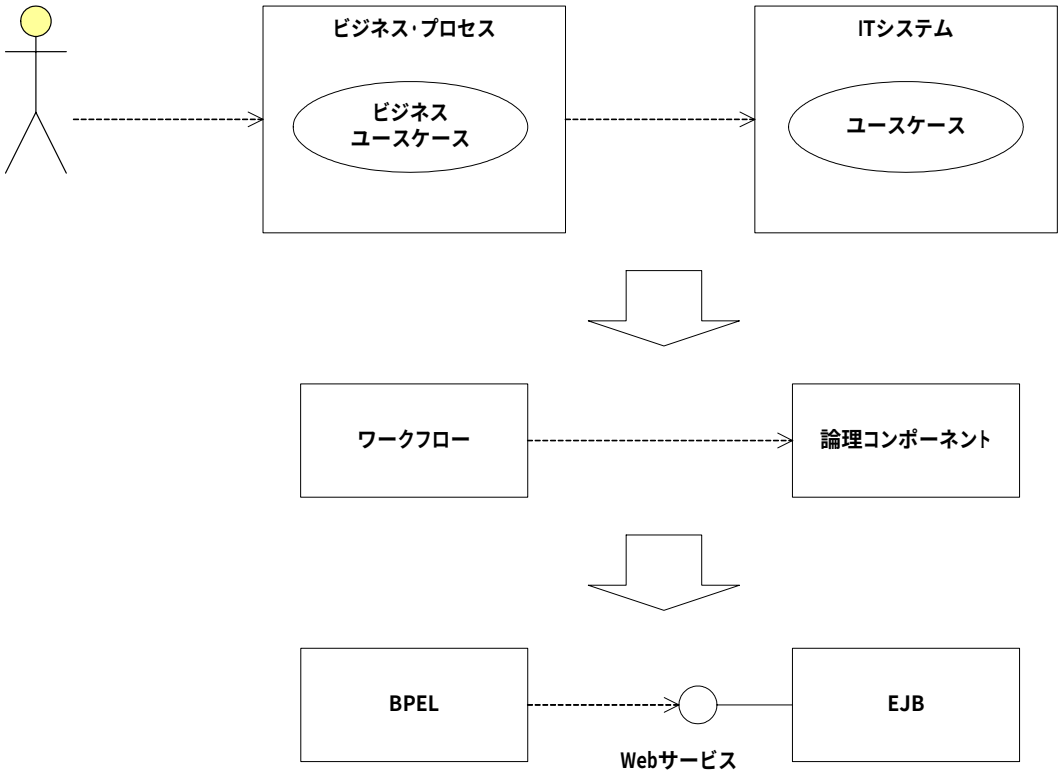
EJB&Webサービスの実現

- 論理コンポーネントを実装に落とし込む
  - 実現方法1：自動生成
  - 実現方法2：EJBでハンドリング
- いずれにしても、EJB＋αの範囲のコンポーネント・モデルを対象に考えていけばよい
  - Webサービスそのものは下位プロトコルの位置付けのイメージ(e.g. TCP/IP)

モデリングからEJB、SOAへ

- ビジネス・プロセスを論理コンポーネント・モデルに変換する技術
  - 2つのユースケースが中核技術となる
    - ビジネス・ユースケース
    - システム・ユースケース
- 論理コンポーネント・モデルをEJB、SOAへ変換する技術
  - プラットフォームに依存
  - XMLを意識しつつEJBを中心に取り組んでいくとよい

モデリングからEJB、SOAへ



まとめ

- モデリングの役割
  - ユースケース
  - 論理コンポーネント
- JavaでのSOAはEJBを中心に考える
  - ユースケースから論理コンポーネントへ
  - 論理コンポーネントからEJB＋SOAへ

付録

- モデリングからJavaEEへ
- モデリングの流れ

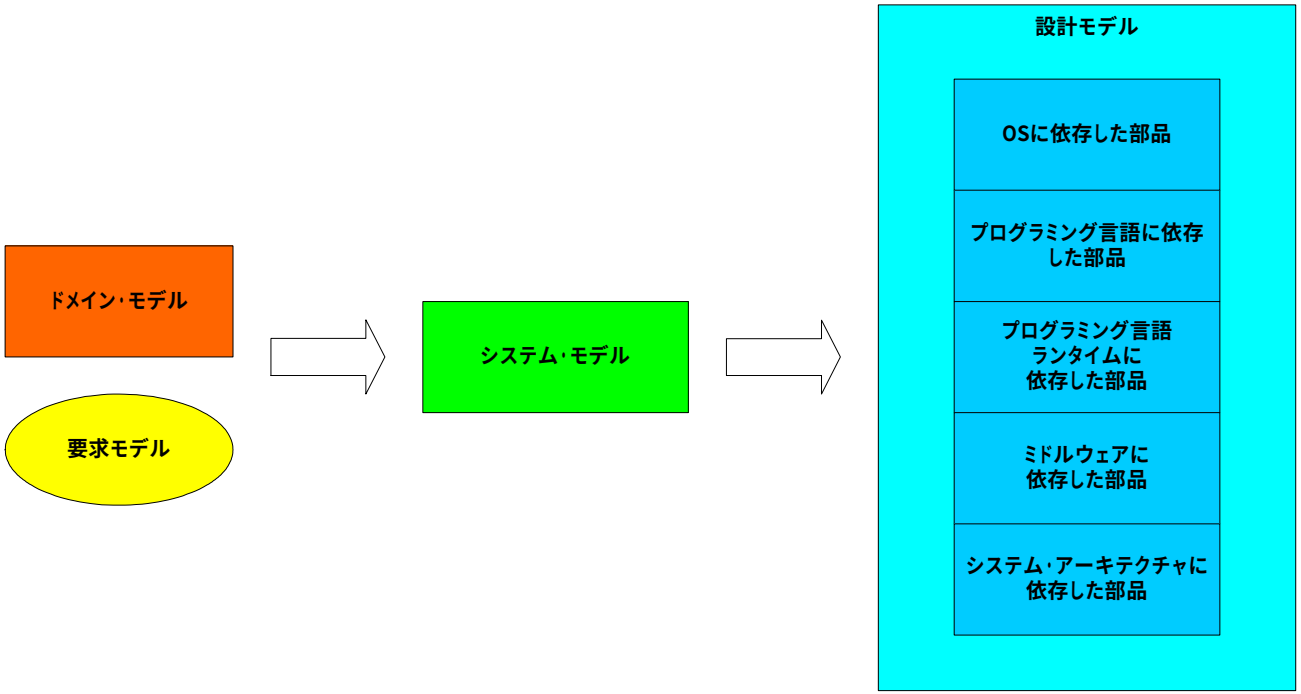
付録

- モデリングからJavaEEへ
- モデリングの流れ

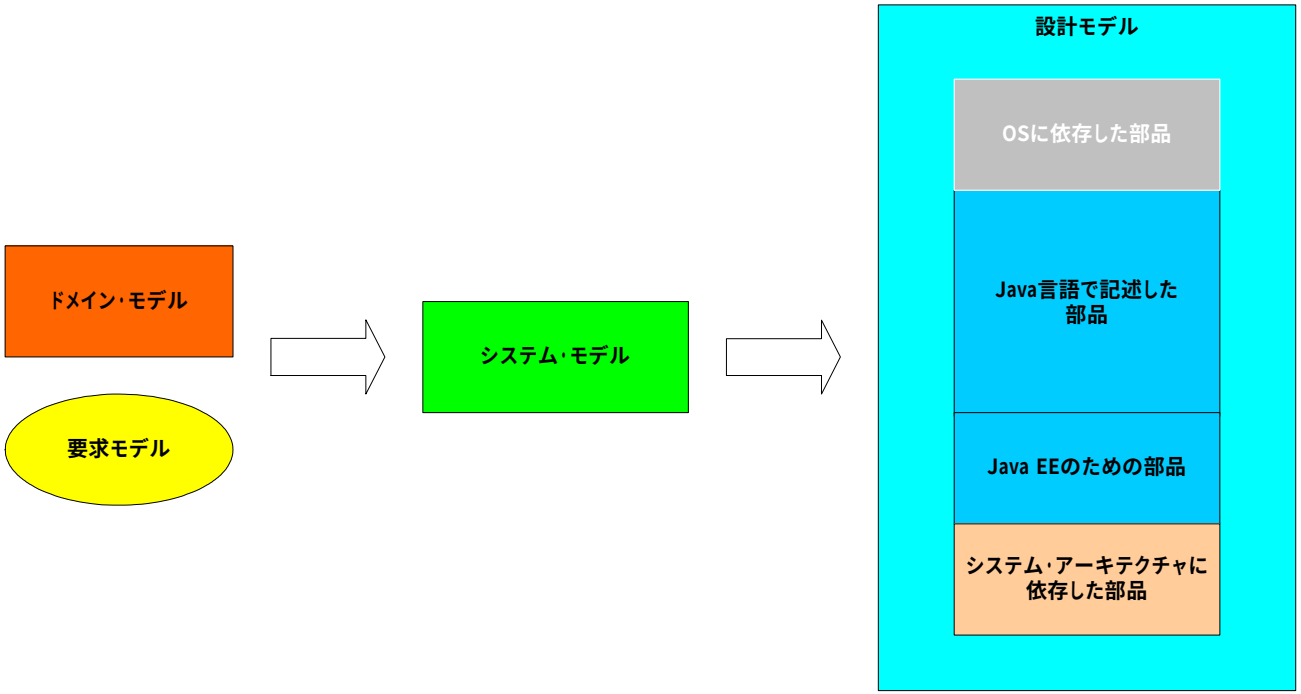
Java EE

- Java Enterprise Edition (JEE)
  - Java 2 Enterprise Edition 1.4**の次は**Java Enterprise Edition 5 (J2EE→JEE)
- Web**アプリケーションサーバ**
  - Web
    - Servlet, JSP, JSF
  - **分散コンポーネント**
    - EJB (Session Bean, Entity Bean, Message Bean)
  - **TP**(Transaction Processing)**モニタ**
    - **データベース･トランザクション**
    - **負荷分散**
    - **可用性(クラスタによる多重化など)**
    - **運用管理**

設計とプラットフォーム

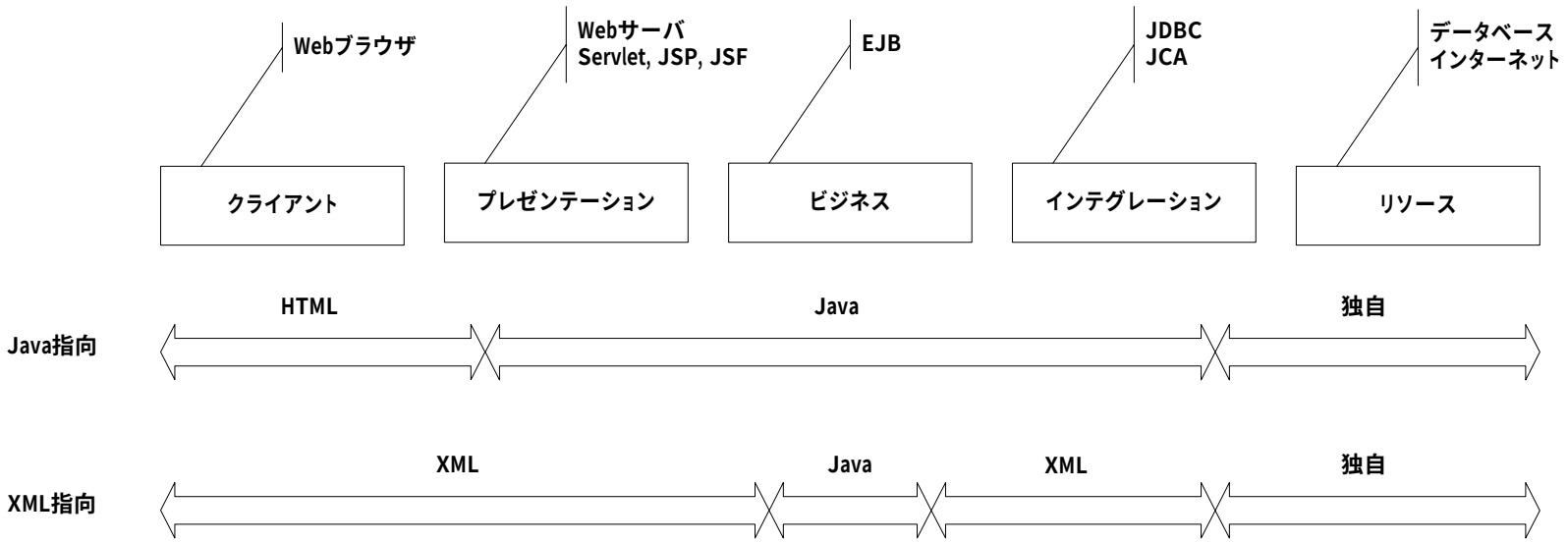


設計とプラットフォーム/Java+Java EEの場合



JavaEEアーキテクチャ

JavaEEデータフロー



JavaEEの構成要素(1)

- クライアント・ティア
  - GUIアプリケーション
  - アプレット
  - MIDlet
- プレゼンテーション・ティア
  - Webページ
  - サーブレット
  - JSP
  - JSF
  - Webフレームワーク(Strutsなど)
  - テンプレート・エンジン(Velocityなど)

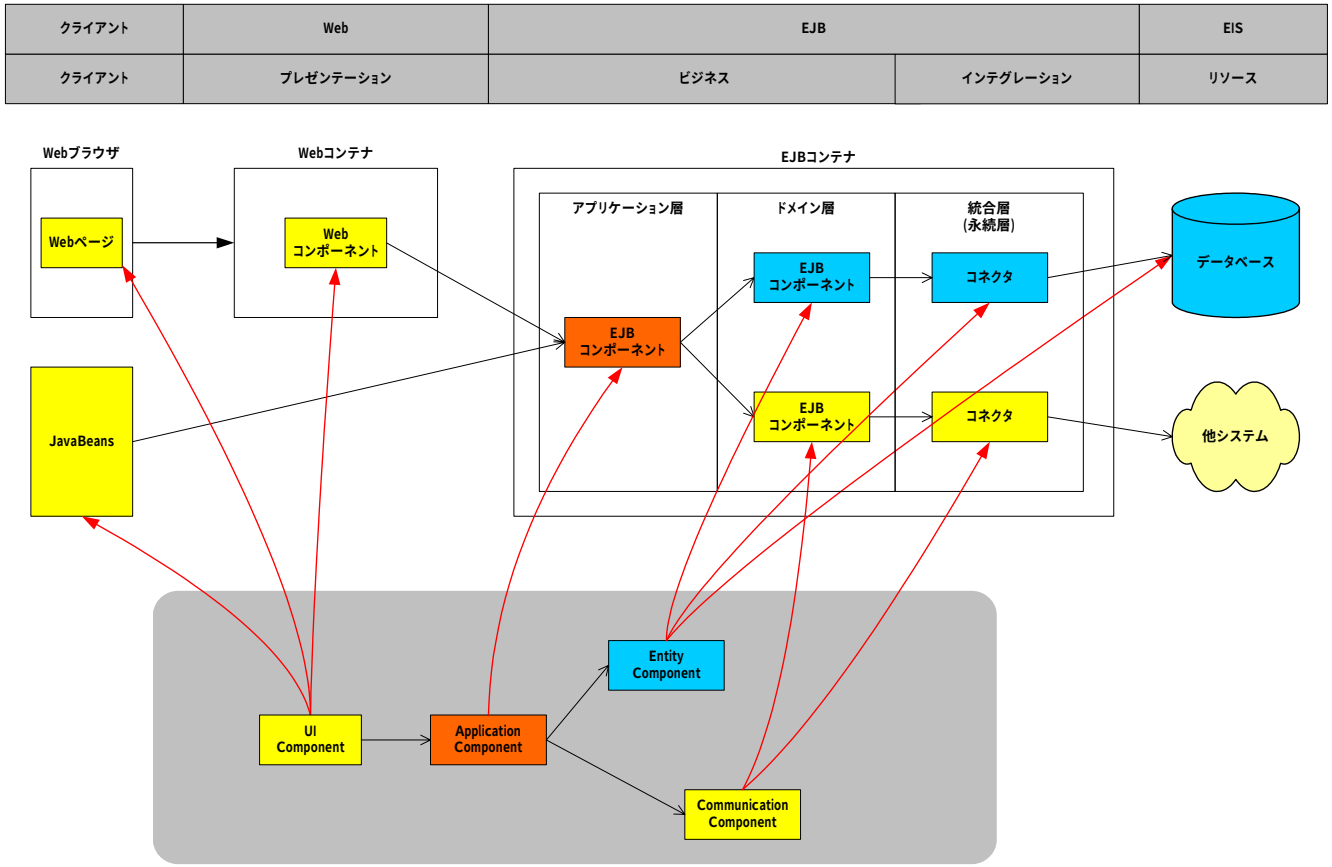
JavaEEの構成要素(2)

- **ビジネス・ティア**
  - EJB
    - Session Bean (Stateless/Stateful)
    - Entity Bean
    - Message Bean
- **インテグレーション・ティア**
  - JDBC
  - JCA(Java Connector Architecture)
- **リソース・ティア**
  - **データベース**
    - **リレーショナルデータベース**
    - **オブジェクト・リレーショナルデータベース**
    - **オブジェクト指向データベース**
    - XMLデータベース
  - ERP
  - Web

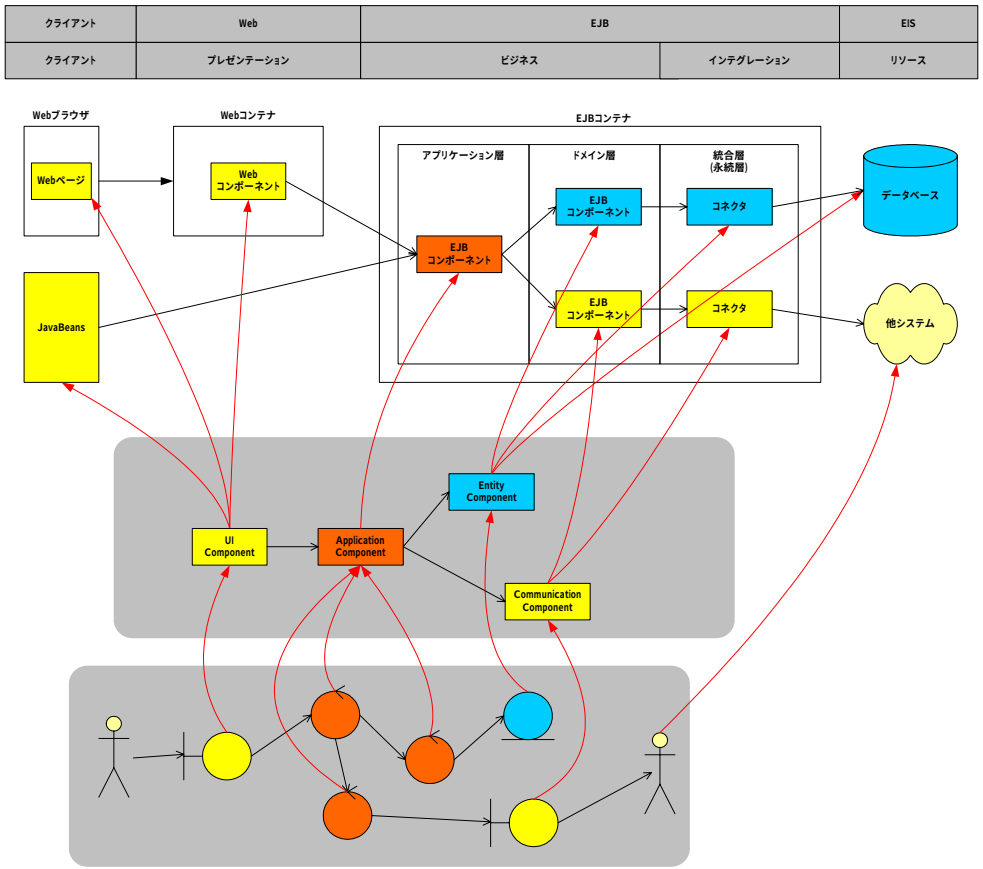
アプリケーション・アーキテクチャ

- プレゼンテーション層
  - 人間(＋その他の外部システム)にUI(＋その他のインタフェース)を提供する層
- アプリケーション層
  - アプリケーション・ロジックを提供する層
- ドメイン層
  - アプリケーションから操作するドメイン・オブジェクトを提供する層
- 統合層(永続層)
  - ドメイン層を外部リソース(データベース、ERP、他システム)と統合するための層
  - データベースを対象とする場合には永続層と呼ぶ

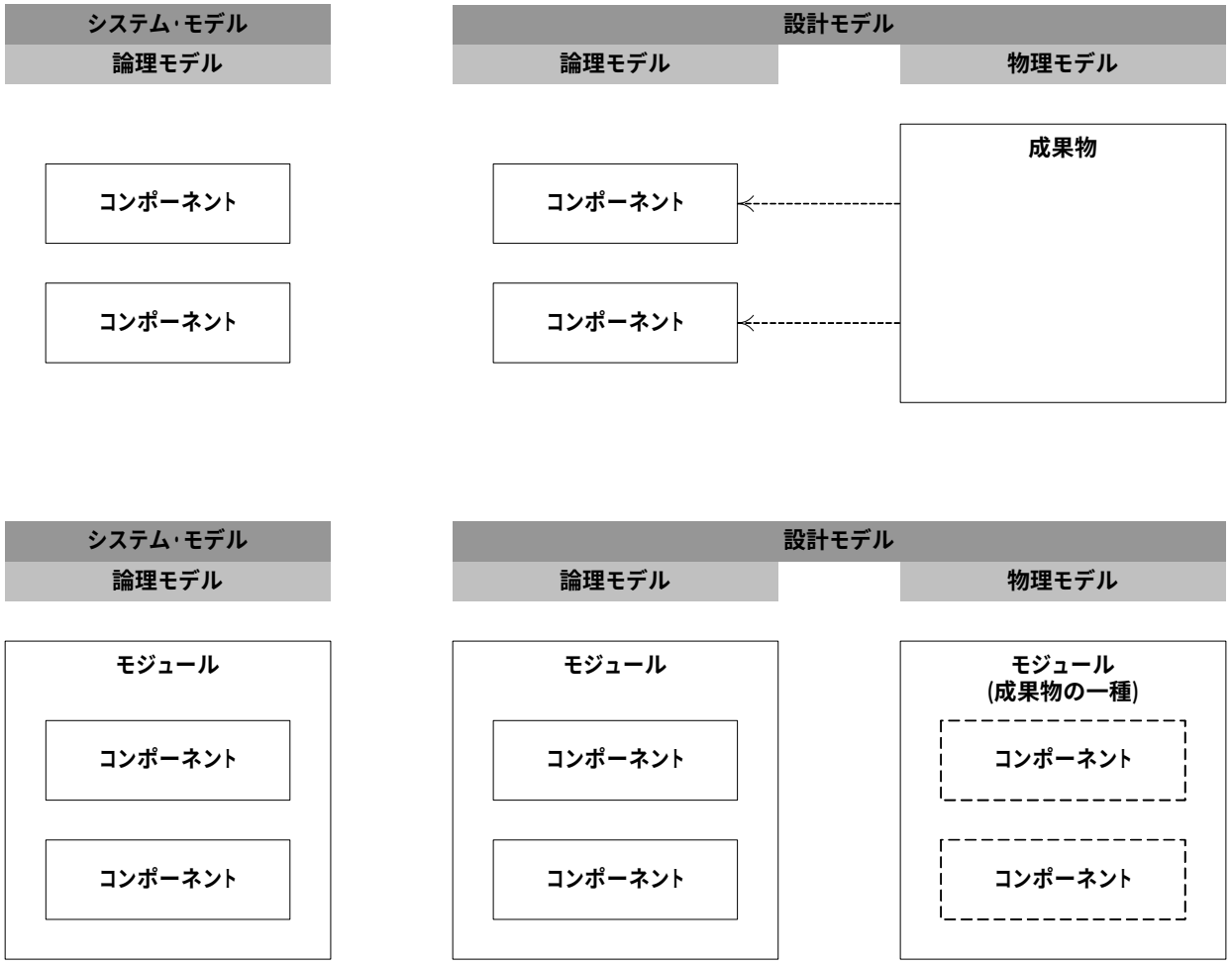
分析モデルとJavaEE



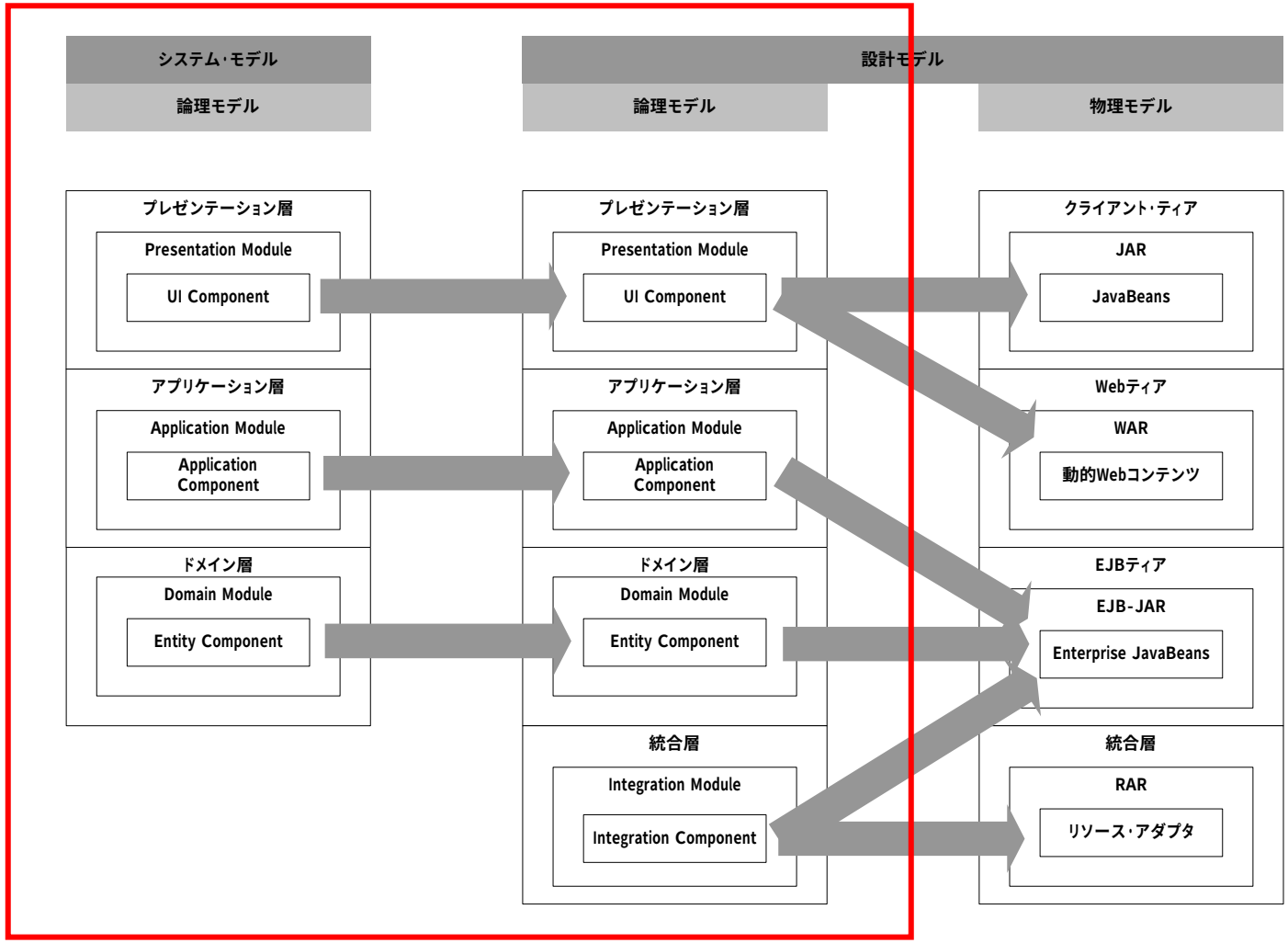
参考：ロバストネス・モデル/アーキテクチャ・モデルとJavaEE



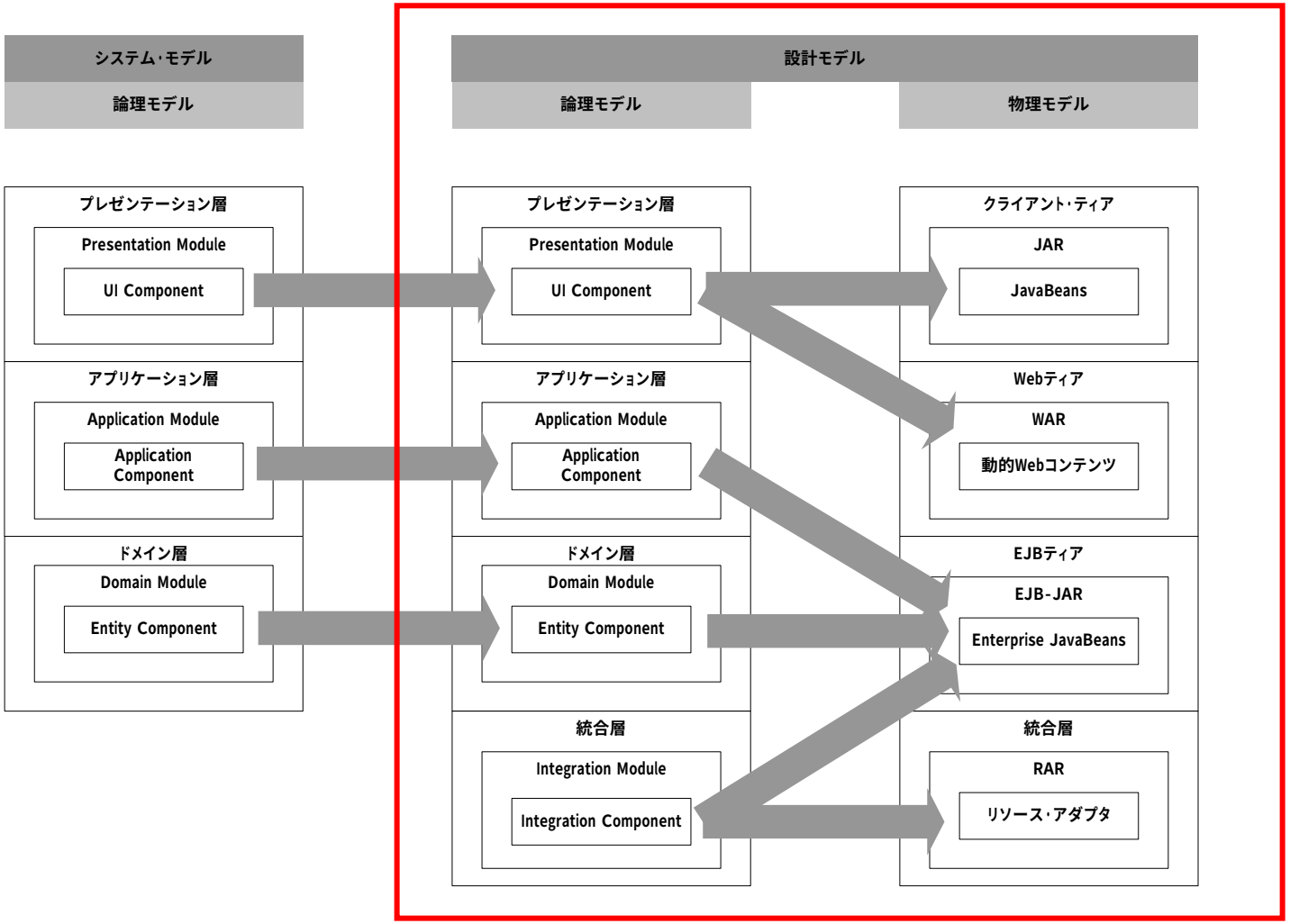
モジュール、コンポーネント、成果物



システム・モデルと設計モデルの対応



アプリケーション・アーキテクチャとWebアーキテクチャ



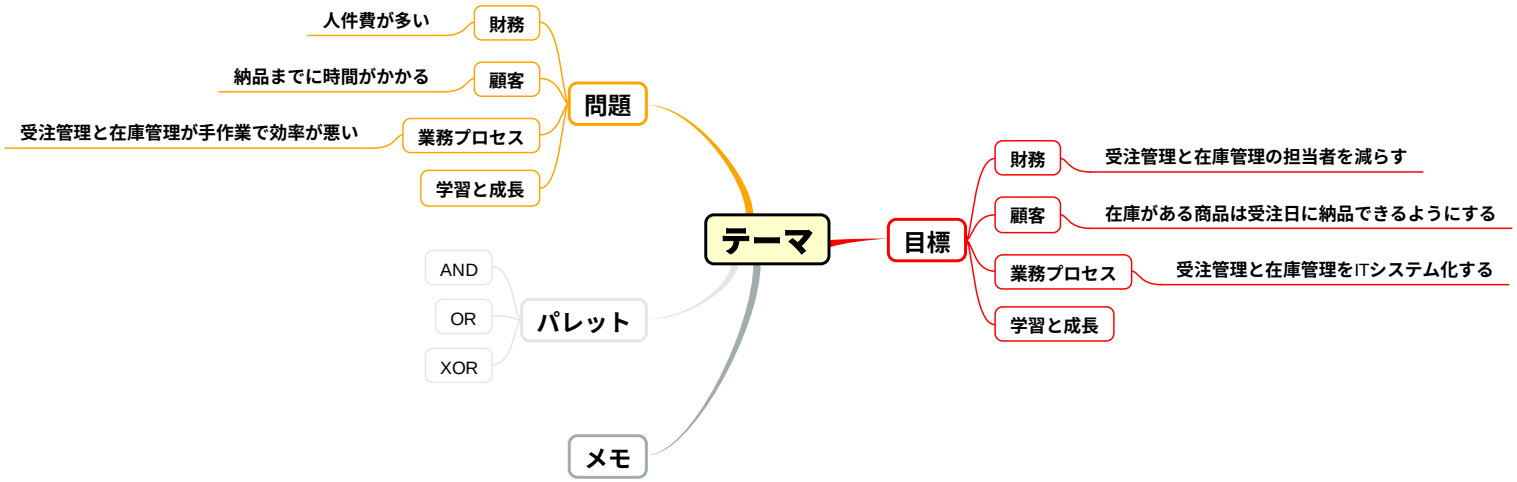
付録

- モデリングからJavaEEへ
- モデリングの流れ

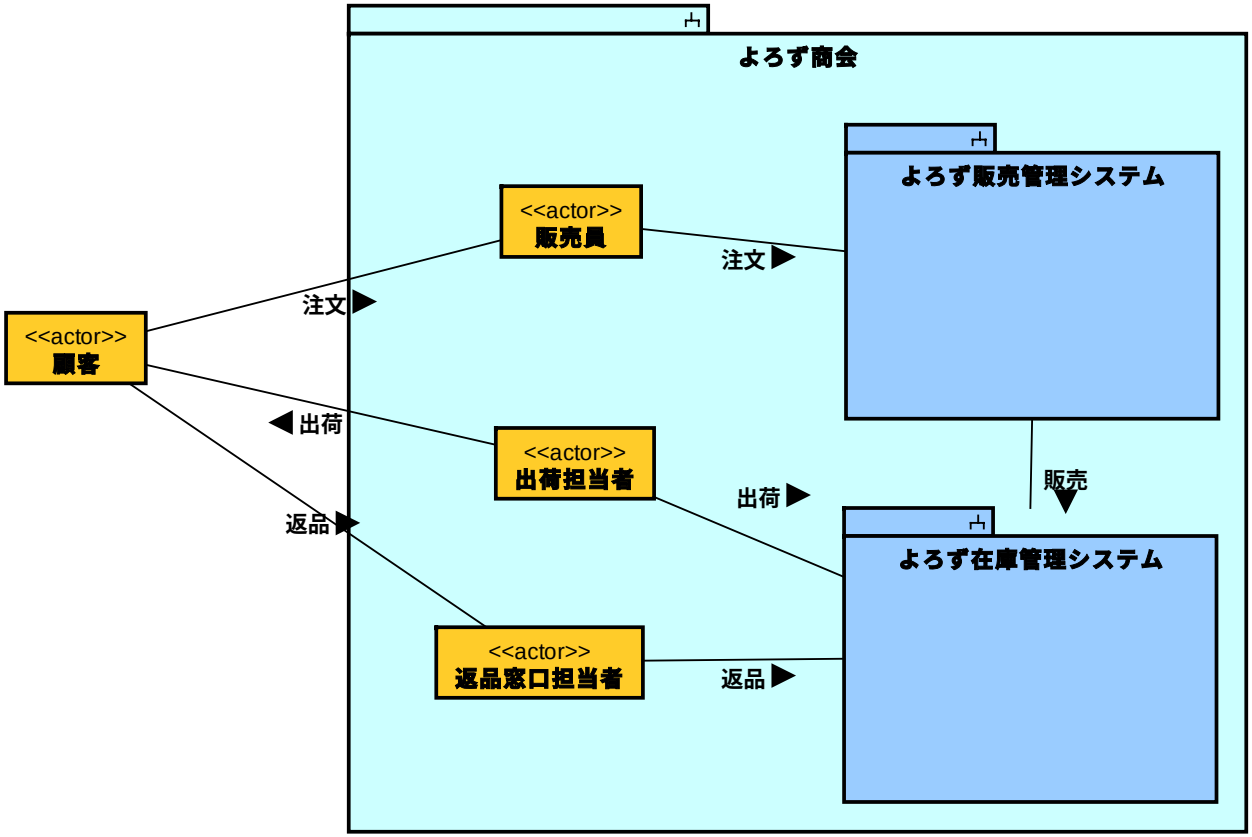
例題：よろず商会
業務ビジョン

- For [**美術品を購入する顧客**]
- Who [**安価でセンスのよい美術品を手軽に入手したい**]
- The [**よろず商会**] is a [**古美術販売業**]
- That [**カタログで古美術を通販する**]
- Unlike [**成金商会**]
- Our product [**気に入らない商品は無料で引き取り**]

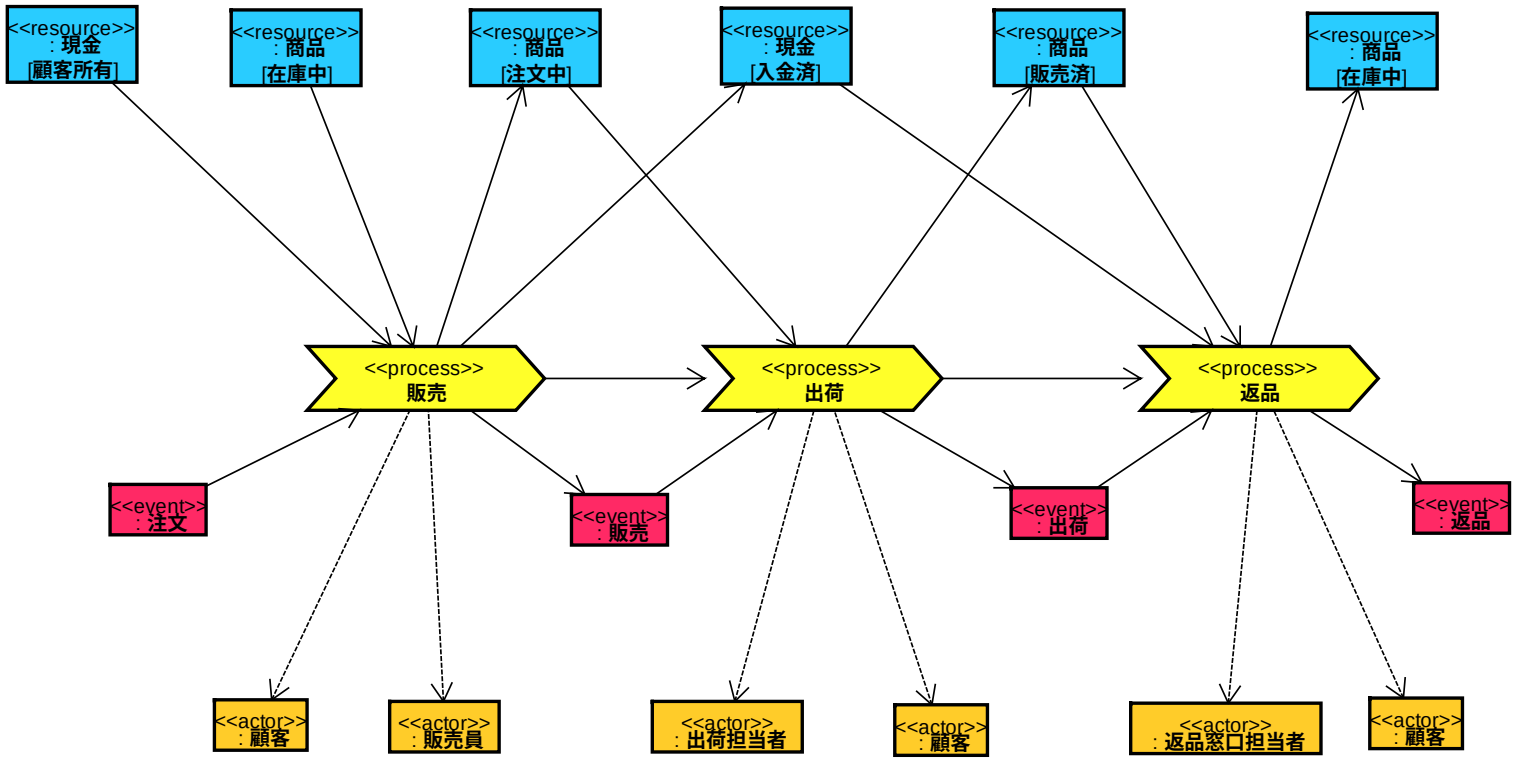
例題：よろず商会
業務ゴールモデル



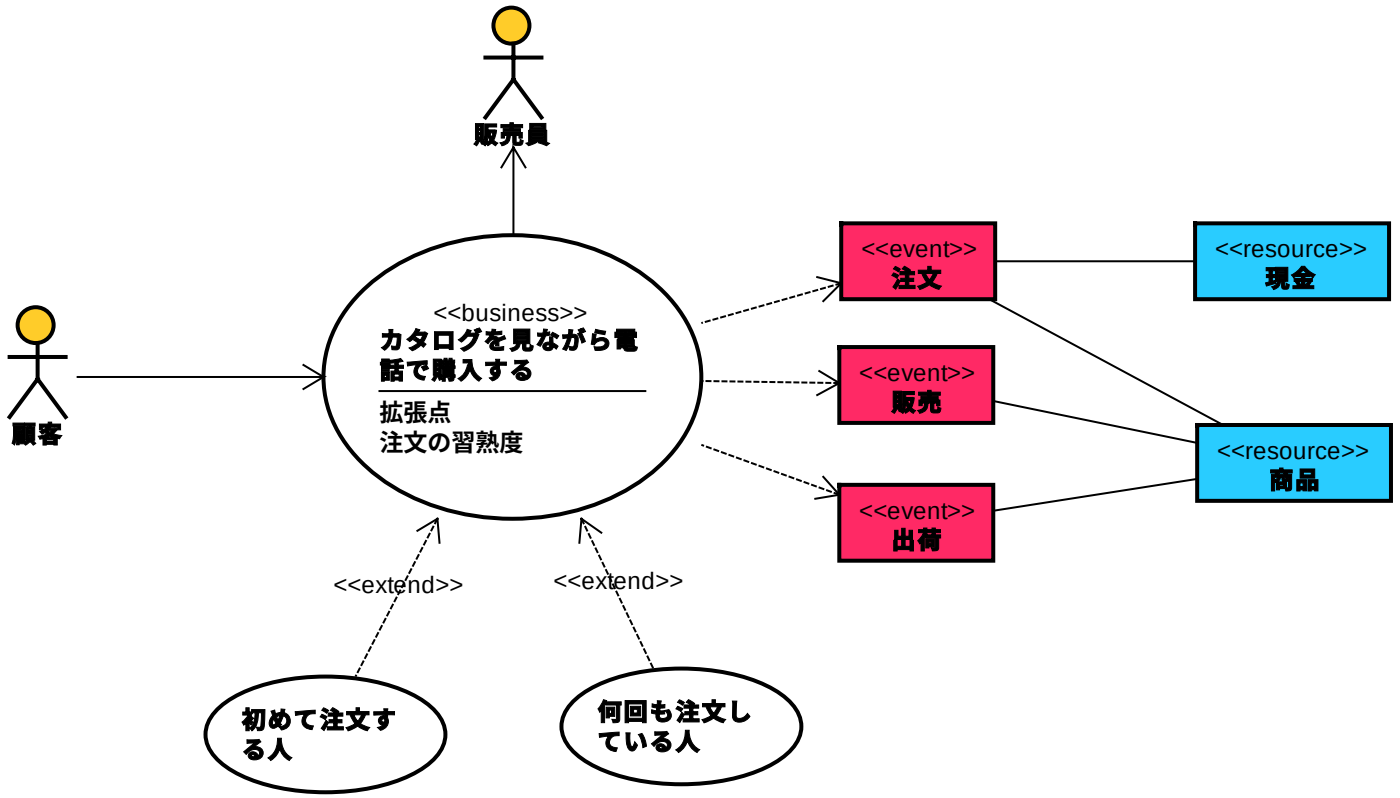
例題：よろず商会
業務コンテキスト



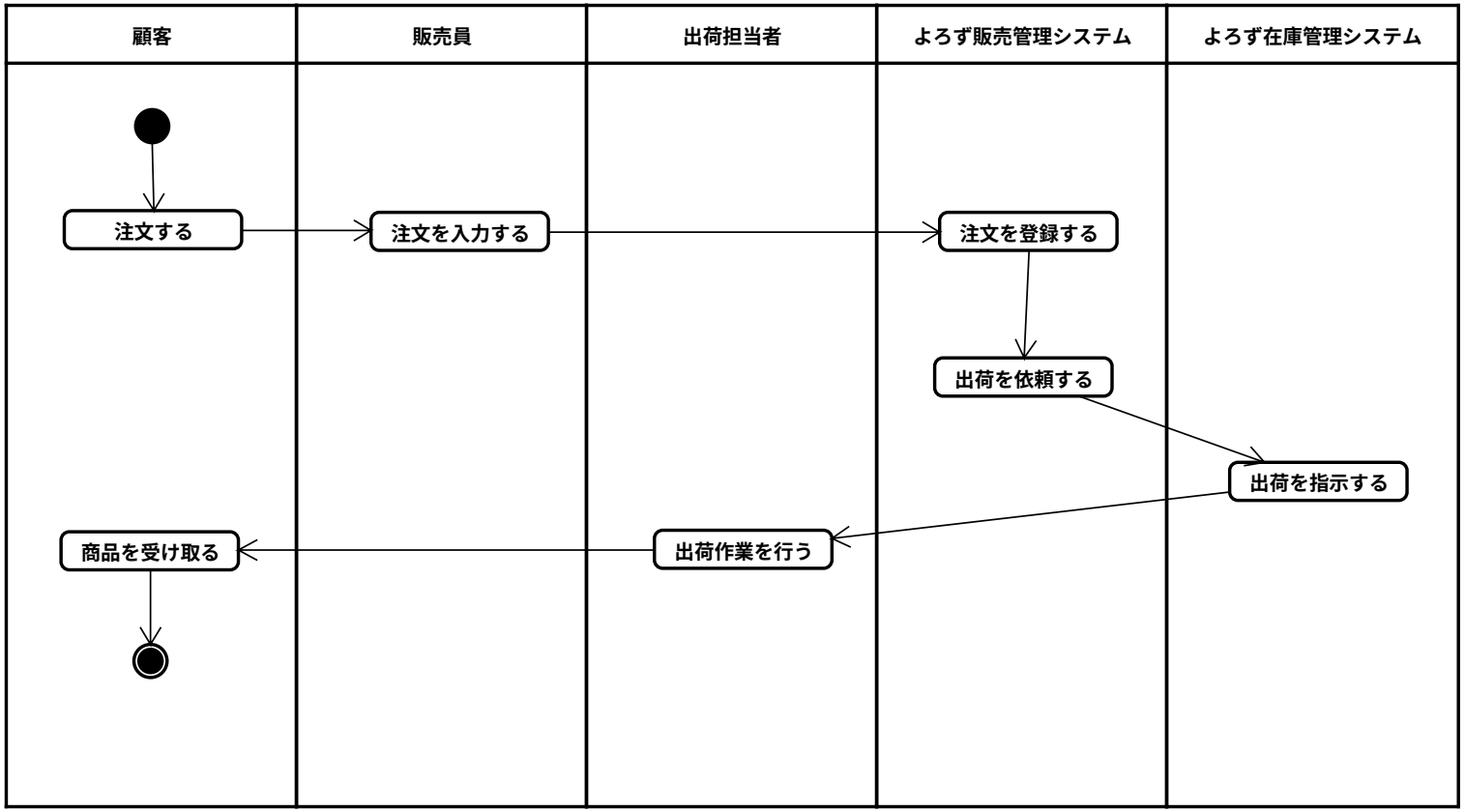
例題：よろず商会
業務モデル/業務プロセス



例題：よろず商会
業務モデル/業務ユースケース(+ドメイン・モデル)

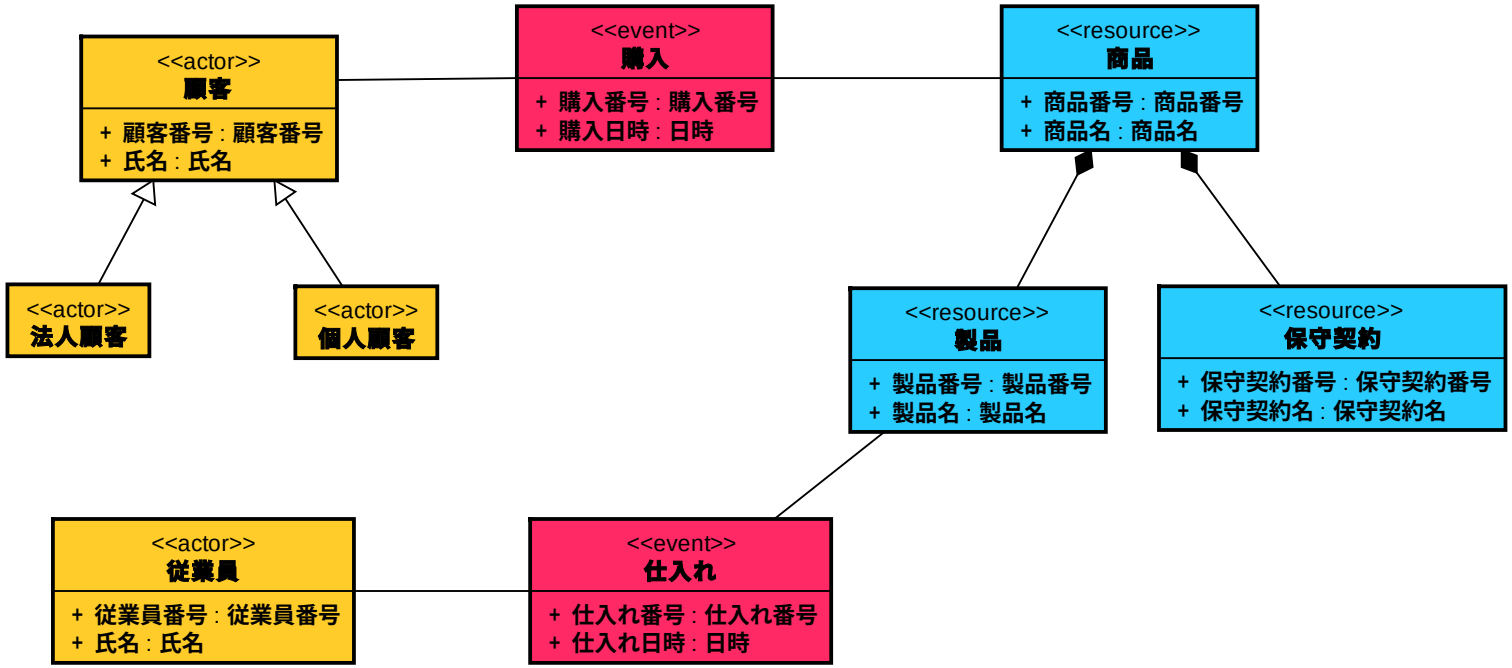


例題：よろず商会
業務モデル/業務フロー



例題：よろず商会
ドメイン・モデル：情報モデル

顧客は商品を購入する。
従業員は商品を仕入れる。
顧客には個人顧客と法人顧客がある。
商品は製品と保守契約から構成されている。
顧客は顧客番号と氏名を持っている。
従業員は従業員番号と氏名を持っている。



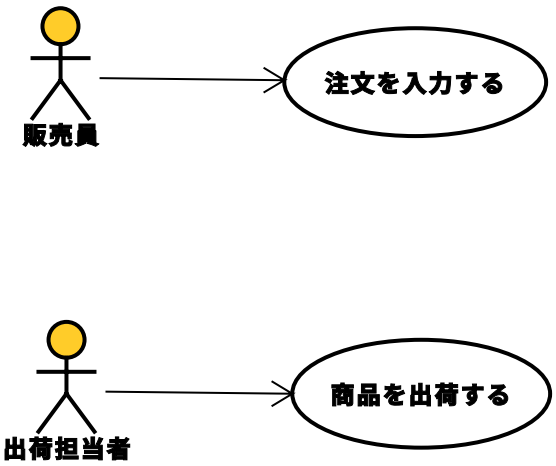
例題：よろず商会
要求モデル/販売管理システム・ビジョン

- For [**販売員**]
- Who [**販売作業を効率的に行いたい**]
- The [**よろず販売管理システム**] is a [**販売管理システム**]
- That [**在庫管理システムと連動したWebベースの販売管理システム**]
- Unlike [**紙台帳**]
- Our product [**在庫管理システムと連動して、出荷は自動的に行われる**]

例題：よろず商会
要求モデル/ユースケース摘要

アクター	ゴール	摘要
販売員	注文を入力する	顧客からの要求に従って注文を行う。
出荷担当者	出荷作業を行う	出荷の必要が発生したことの通知を受けて出荷作業を行う。

例題：よろず商会
要求モデル/ユースケース図



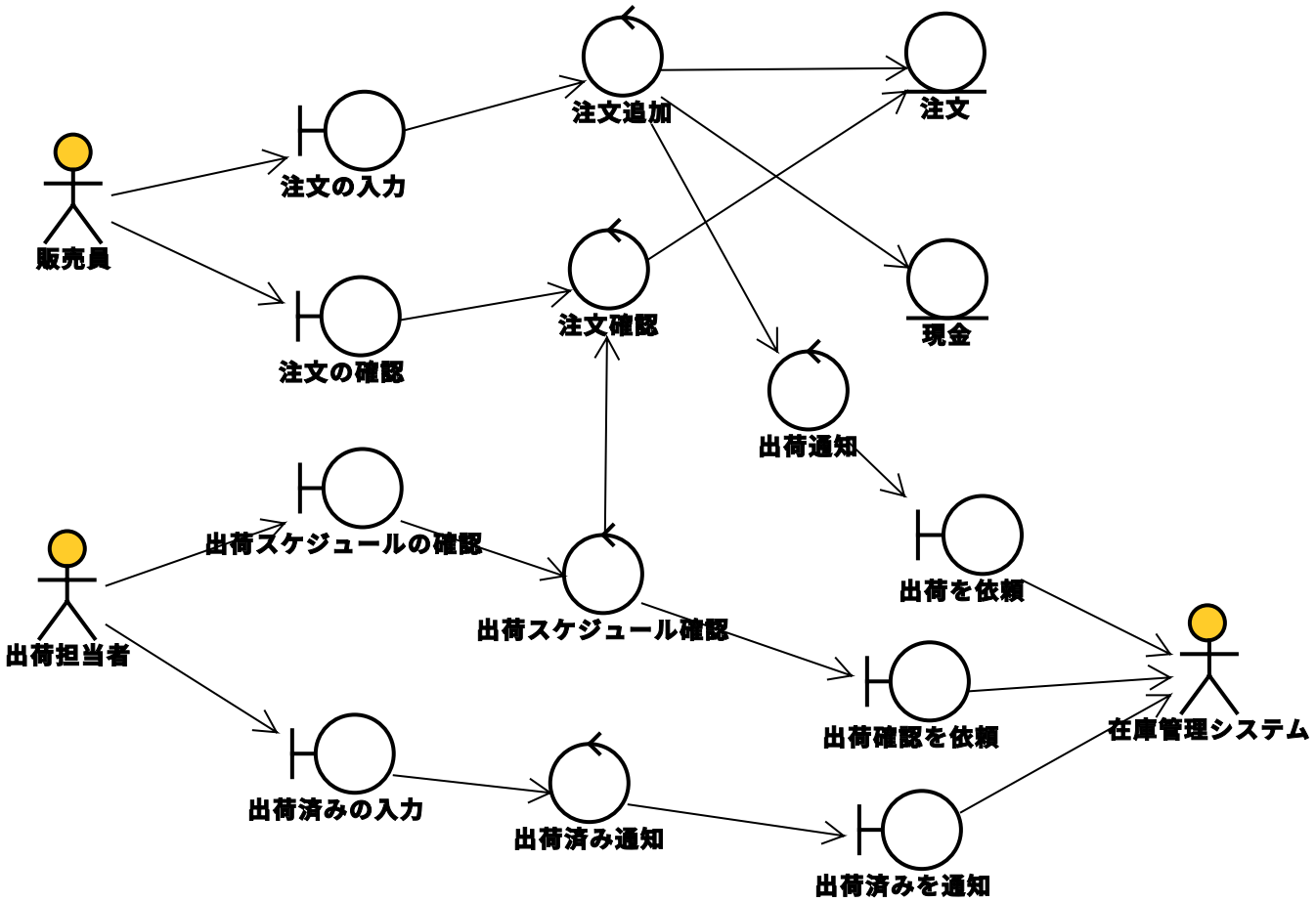
例題：よろず商会
要求モデル/ユースケース記述

ユースケース名	注文を入力する	**ユースケース番号**	UC0001
プライマリアクター	販売員		
セカンダリアクター	在庫管理ステム		
事前条件	顧客から注文内容を聞いている。 現金を預かっている。		
事後条件	新規注文が追加されている。 現金が増えている。		
基本フロー	1. 販売員は注文を入力する。 2. システムは在庫管理システムに販売を通知する。 3. 販売員は注文結果を確認する。		
代替フロー	N/A		
例外フロー	2. a. 在庫がない。		

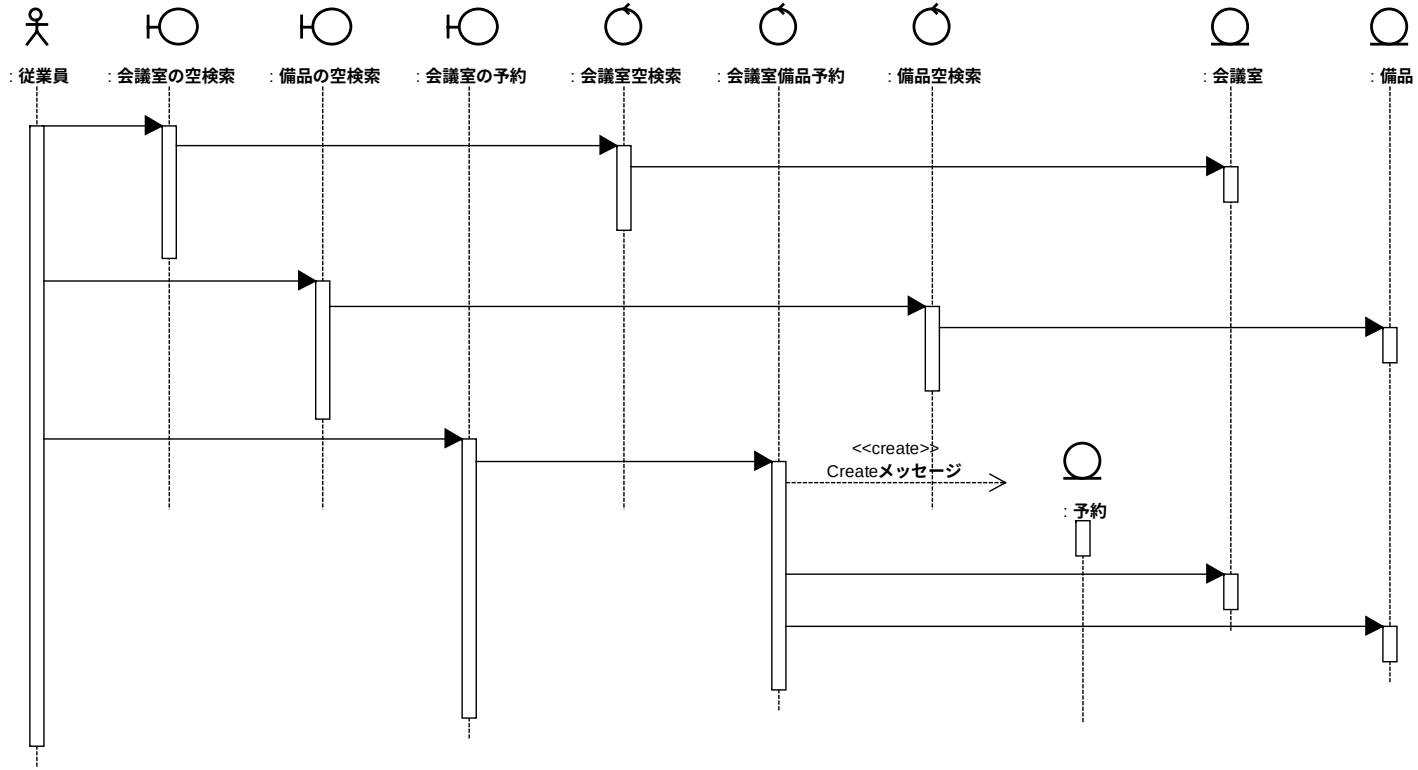
例題：よろず商会
要求モデル/計画表

アクター	ゴール	業務ニーズ	技術難易度	優先順位	ユースケース番号
販売員	注文を入力する	必須	普	1	UC01
出荷担当者	出荷作業を行う	重要	高	2	UC02

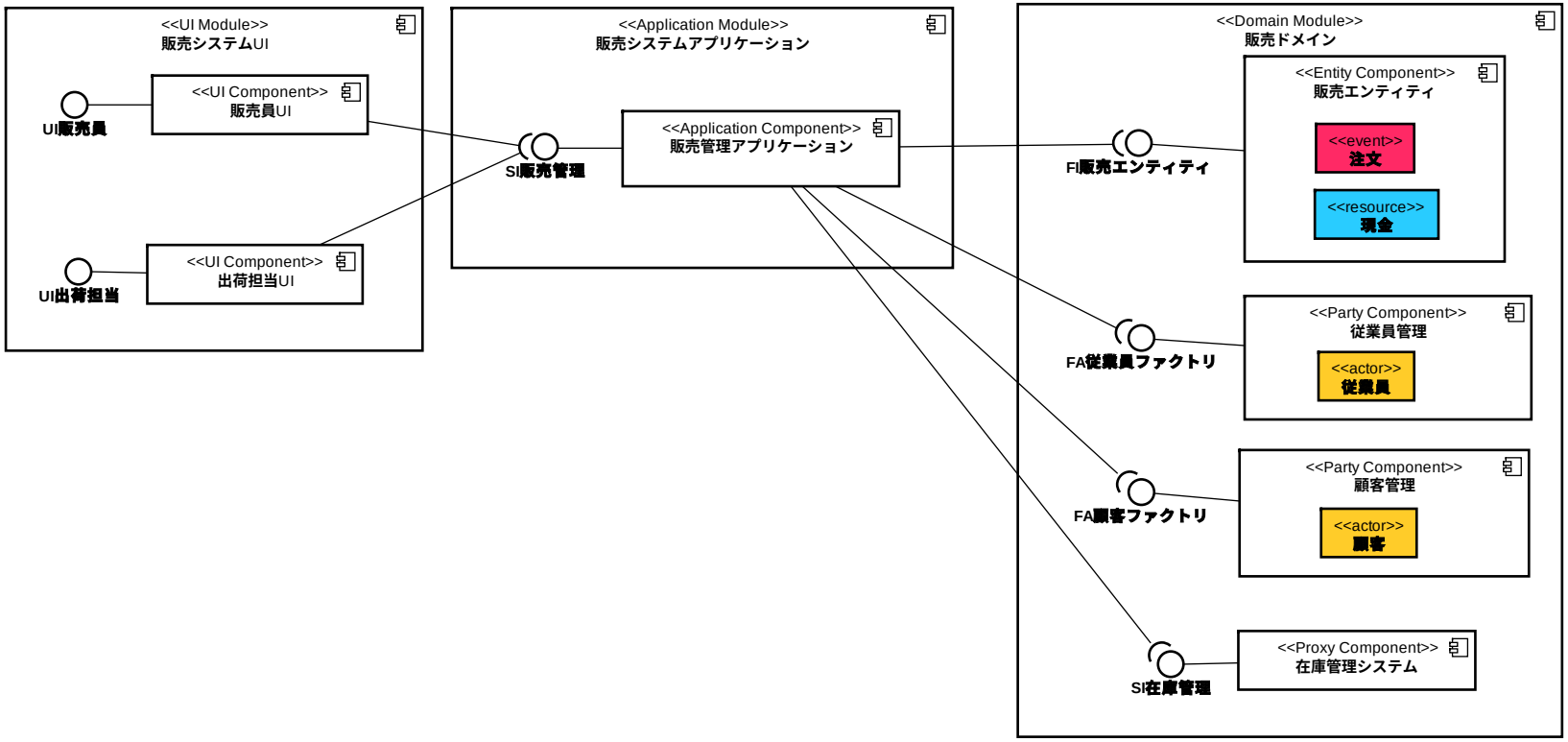
例題:よろず商会
システム・モデル/ロバストネス・モデル
クラス図



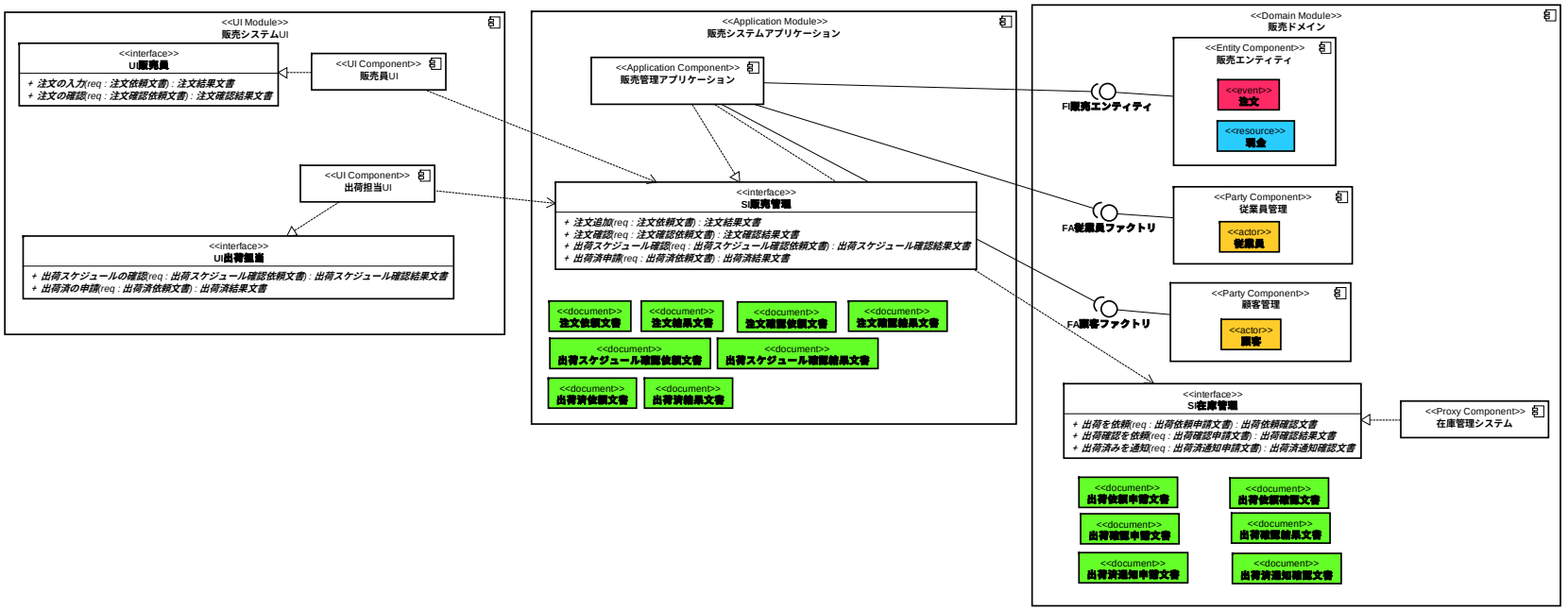
例題：よろず商会
システム・モデル/ロバストネス・モデル
シーケンス図



例題：よろず商会
システム・モデル/アーキテクチャ・モデル
コンポーネント図



例題：よろず商会
システム・モデル/アーキテクチャ・モデル
コンポーネント図



例題：よろず商会
システム・モデル/アーキテクチャ・モデル
シーケンス図