Java・Caché
真のオブジェクト指向開発を実現

～O/Rマッピングを排除し開発効率と安定性を向上させる～

第2回開発者向けセミナー Caché Symposia 2004

2004年5月19日
ネクストデザイン有限会社
村山 徹

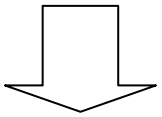
ご紹介の動機

- **成功事例として**

  オブジェクト指向技術が広く普及する中で、
  成功事例の発表件数が少ない。

- **オブジェクト指向開発の現状打開策として**

  多くの現場では、O/R**マッピングの負荷だけを背負い込み、**
  オブジェクト指向の恩恵は受取れていない。



Caché Java**バインディング／オブジェクトアクセスを提案**

ネクストデザイン(有)のご紹介

http://www.nextdesign.co.jp/

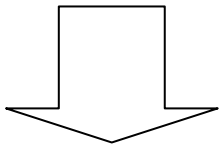
- 所在地: 福岡市 (ソフトウエア・リサーチ・パーク)
- 特徴: オブジェクト指向技術とＪａｖａに特化
- 業務内容:
  - 公開セミナー、オンサイトセミナー
  - コンサルティング
  - システム開発
  - ＵＭＬツールの開発・販売
  - Caché販売パートナー

事例の概要

- 自社製品の永続化エンジンにCachéを採用
  - ユースケース・モデリング・ツール Campass
- 開発技法・ツール
  - オブジェクト指向技術
  - 実装言語：Java (1.4.2_01)
  - GUIコンポーネント：SWT、JFace
  - Caché Javaバインディング (5.0.3)

開発時の基本方針

- オブジェクト指向技術を正しく適用し、その効果を最大限にするために



- リレーショナルデータベースを使用しない

  O/Rマッピングによる負荷をなくし、
  短期間で製品化する

O/Rマッピングとは

- 「オブジェクト・モデル」と「リレーショナル・モデル」のインピーダンス・ミスマッチを解決する手段
- オブジェクトを（複数の）テーブルに関連付ける
- オブジェクトをテーブル群に分解して保存する
- テーブル群を読んでオブジェクトを組立て、復元する
- 主キーを生成するオブジェクトやテーブルにアクセスするためのオブジェクトを追加する

これらは本来不要！

O/Rマッピングが引き起こすもの

- 本来不要なオブジェクトを追加しなければならない
- モデルが複雑になり、分かり難くなる
- 多くのSQLコードを書かなければならない
  (非オブジェクト指向コードを書く必要がある)
- その結果、コード量が増える (1.5〜2.0倍)
- そこにバグが多発する
- テーブルスキーマ変更時の変更箇所が多い

オブジェクト指向開発で失敗につながる最大の原因

- 工数オーバー
- 品質不安定

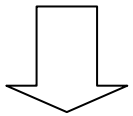
RDBを使わない永続化方式

- **選択肢**
  - （1）**オブジェクト指向データベースを使う**
  - （2）O/R**マッピングツール　＋**　RDB**を使う**
    - （例）Hibernate + PostgreSQL
  - （3）J2EE/CMP **を使う**
  - （4）Caché **オブジェクトアクセスを使う**

- Caché**を採用した理由**
  - **一般的な**Java**の開発スタイルが使えること**
  - **実績**
  - **価格**

その効果

- モデルを自然な状態に保てた
  - 分かりやすいモデル (安定したモデル)
  - 変更、拡張に強いモデル
  - コード量が減少 (1/3 から 1/2)
  - バグが減少

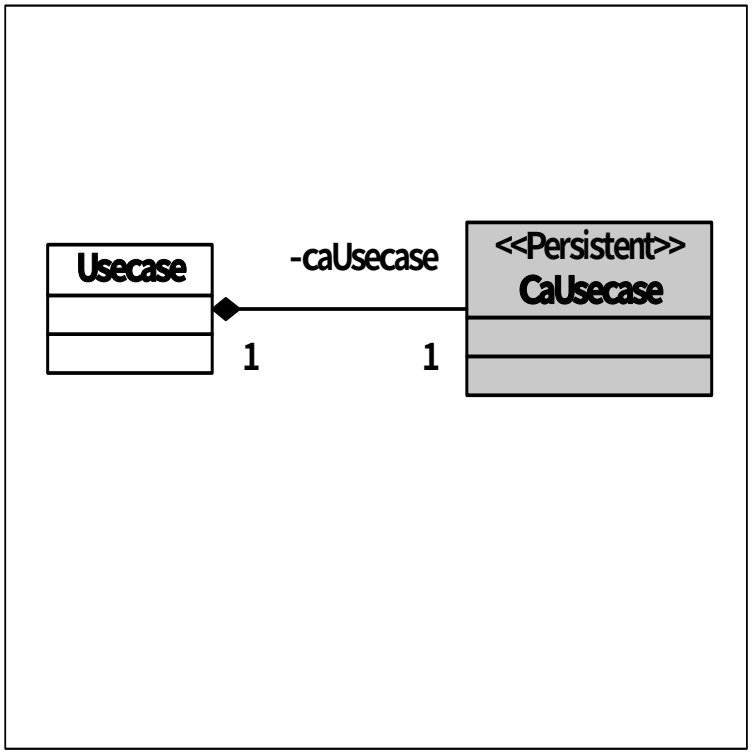


開発のスピードアップ

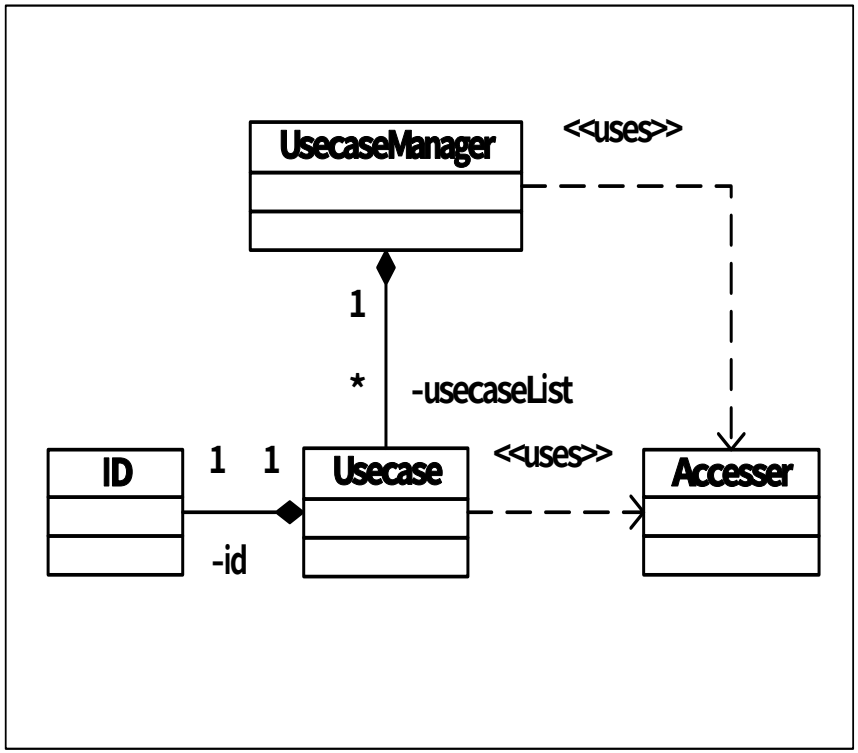
本来のドメインロジックに専念できた

設計モデルの比較

Caché**を使用する場合**

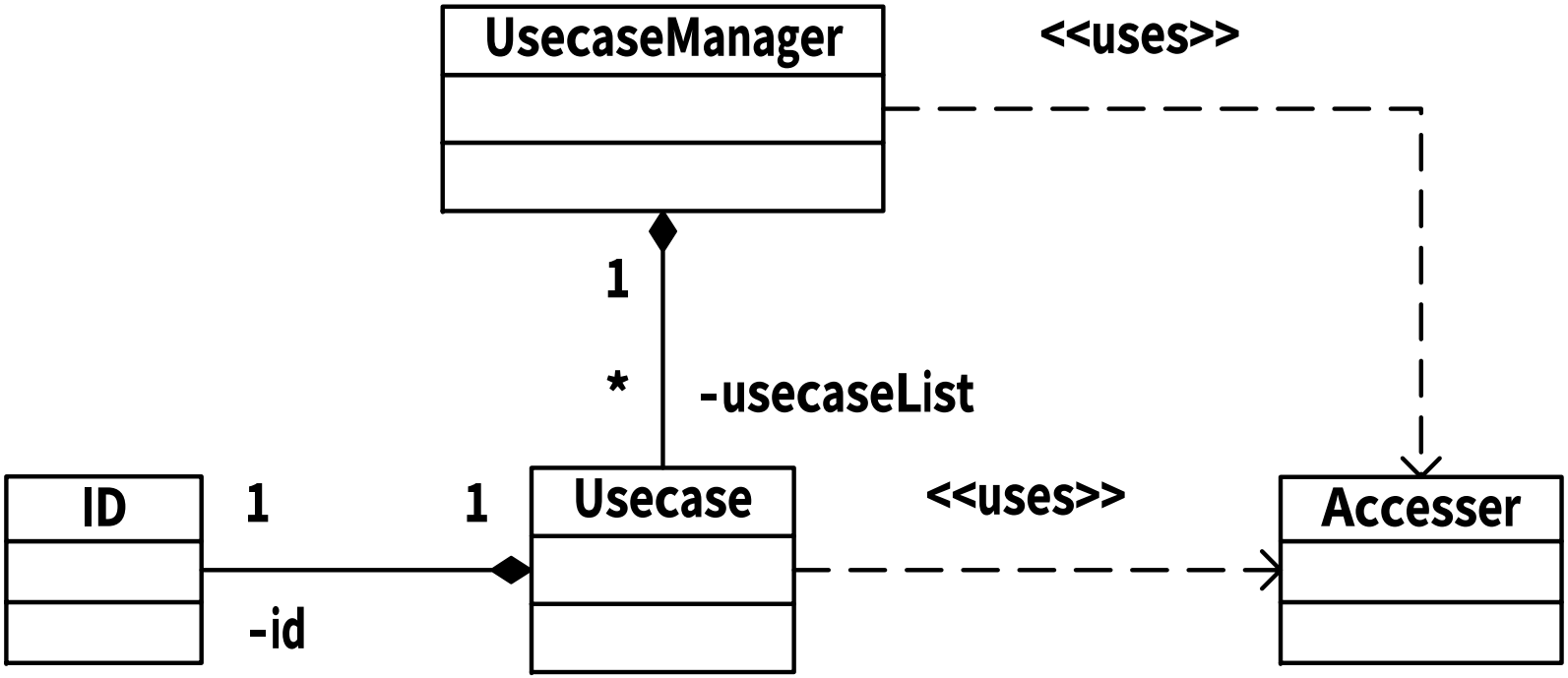


RDB**を使用する場合**



RDBを使用した設計モデル

主キーを生成する責務を持つオブジェクト
エクステントを管理するオブジェクト

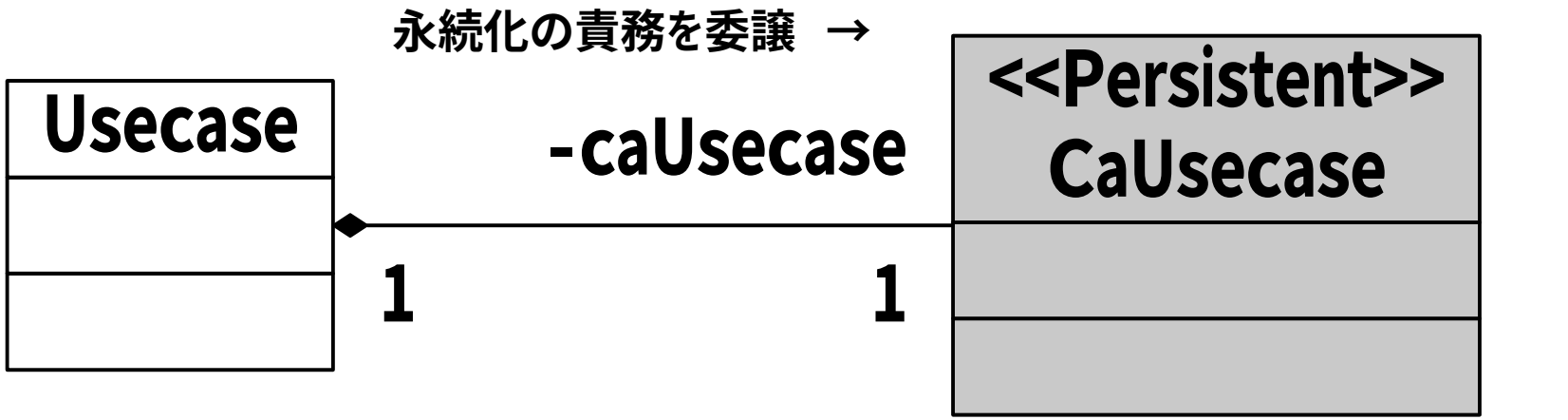


ドメインとしては
不要なオブジェクト

テーブルへのアクセスを
責務とするオブジェクト
ＳＱＬステートメントを含む

Cachéを使用した設計モデル

本事例では、継承ではなくコンポジションを使用した
強いカプセル化を行い、開発手順を含めた相互波及を最小限にした

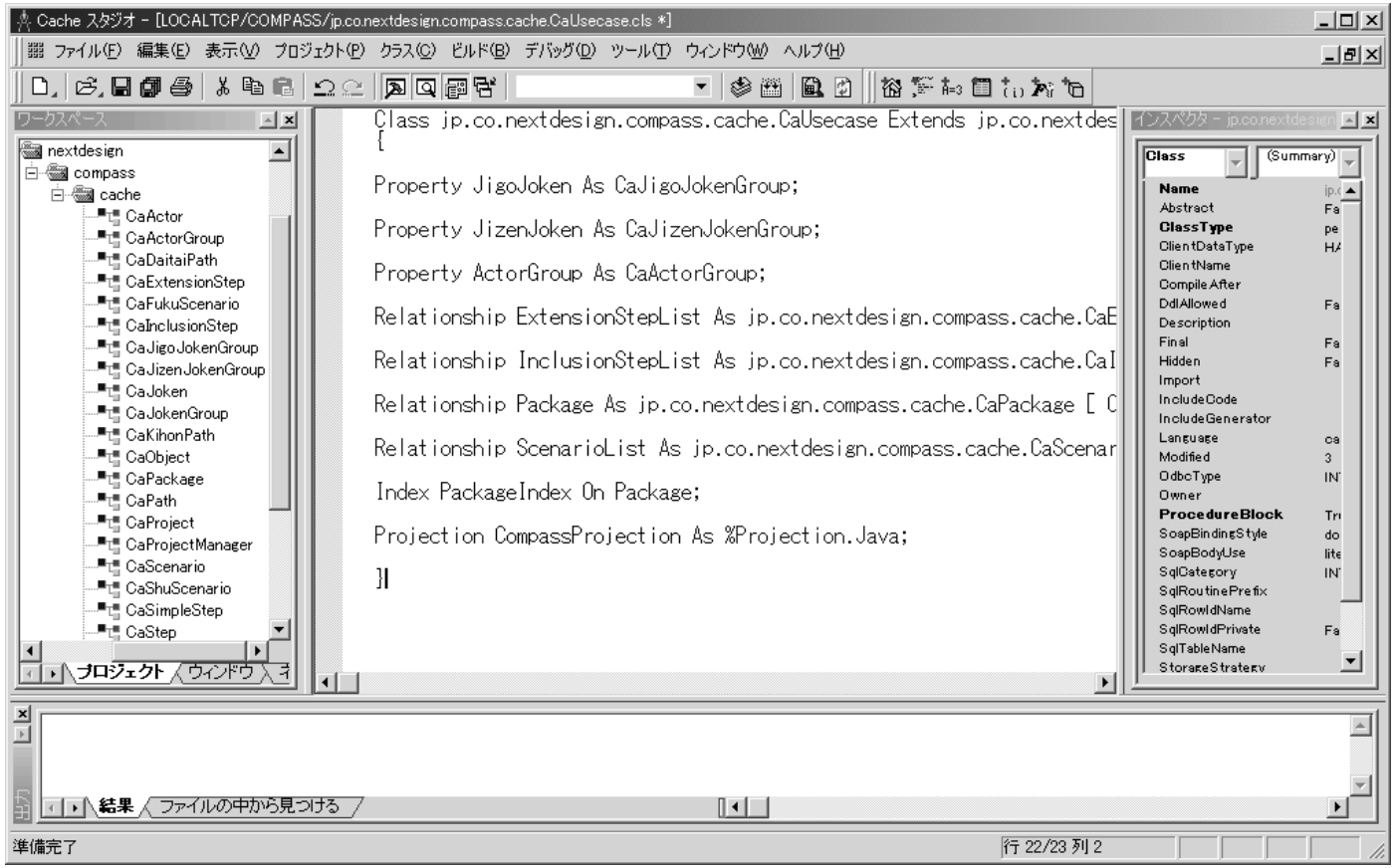


Cachéが自動生成するクラス
永続オブジェクト

開発手順 (繰り返し型)

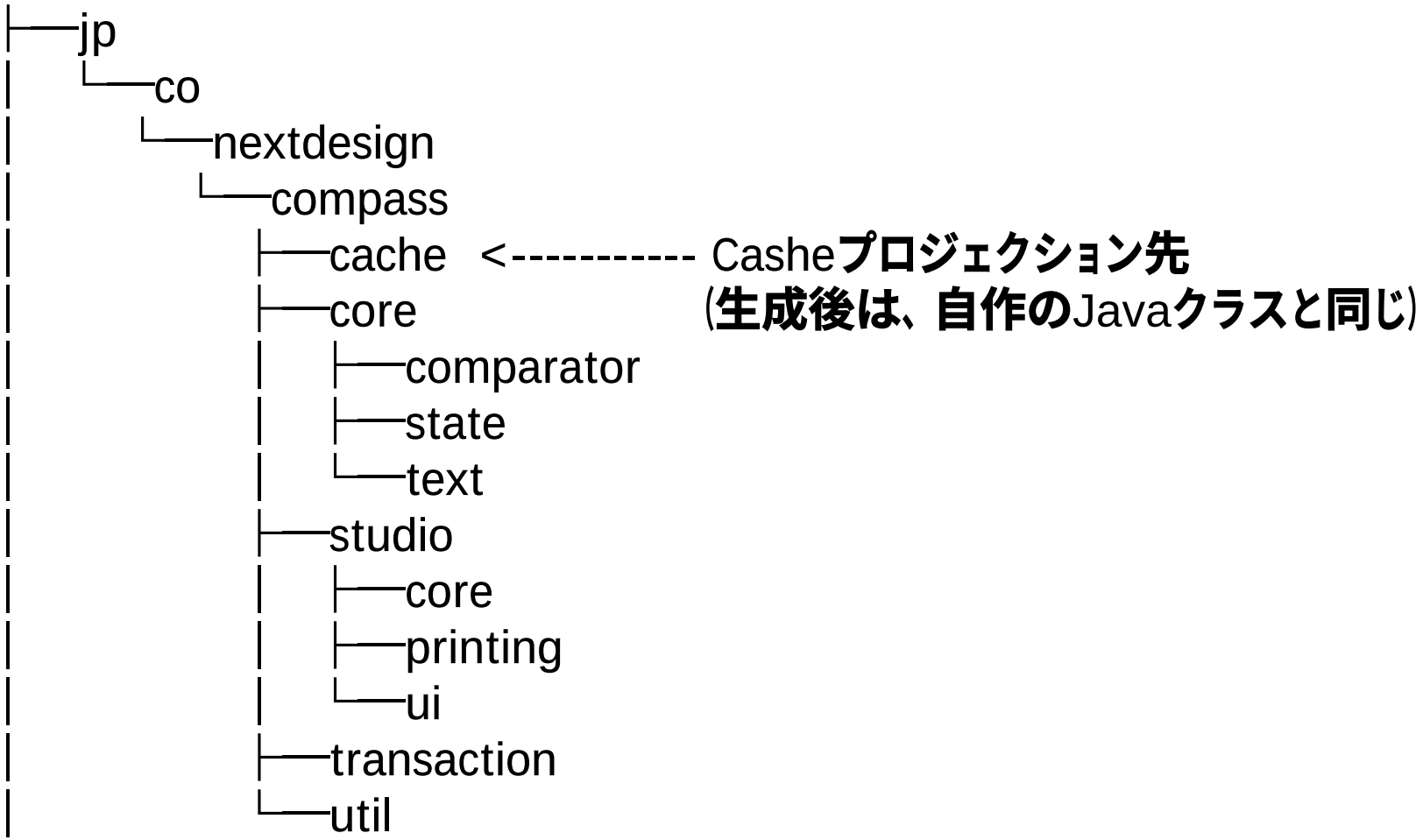
1. オブジェクトモデルを作成する (UML図)
2. Caché スタジオで永続クラスを定義する
   - 本事例では、オブジェクトの属性のみを定義した
3. Caché プロジェクションで Java クラスを生成する
   - 自分の開発フォルダーに出力する (Java ソースコード)
4. 他の Java クラスを作成する
   - テキストエディターや Eclipse を使用する (Java ソースコード)
5. 全てをコンパイルする
   - Ant や Eclipse を使用する
6. 実行テストする (1～6を繰り返す)

Caché**スタジオで永続クラスを定義**



Cachéスタジオで永続化クラスを定義し、Javaクラスを生成する

Caché**プロジェクションでクラスを生成**



Caché利用方法の選択

- **検討した利用方法**

(1) Caché**スタジオで全てのメソッドも定義する**

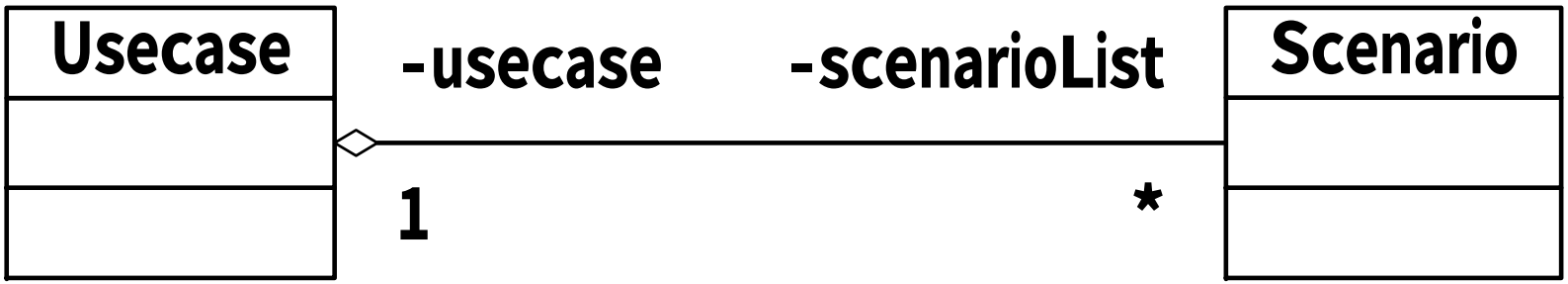
(2) Caché**永続クラスを継承する**

(3) Caché**永続クラスに(永続化のみを)委譲する**

- **(3)を選択した理由**
  - **開発手順の**Caché**スタジオ依存を小さくする**
  - Caché**自動生成クラスのカプセル化を強める**

関連(Relationship)のサポート

1対Nであればプログラマが特別なコードを書く必要がない



(Usecase側の定義)

```
Relationship ScenarioList As Scenario (JAVATYPE = "java.util.Map")
        [ Cardinality = many, Inverse = Usecase ];
```

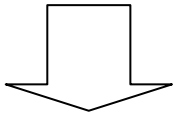
(Scenario側の定義)

```
Relationship Usecase As Usecase [ Cardinality = one, Inverse = ScenarioList ];
```

これらの定義だけで、関連を維持するコードがCaché生成クラスに実装される。

Compassの特徴

- UMLユースケース・モデリング・ツール
- スケジュール管理
- 工数見積り
- オブジェクト指向分析支援

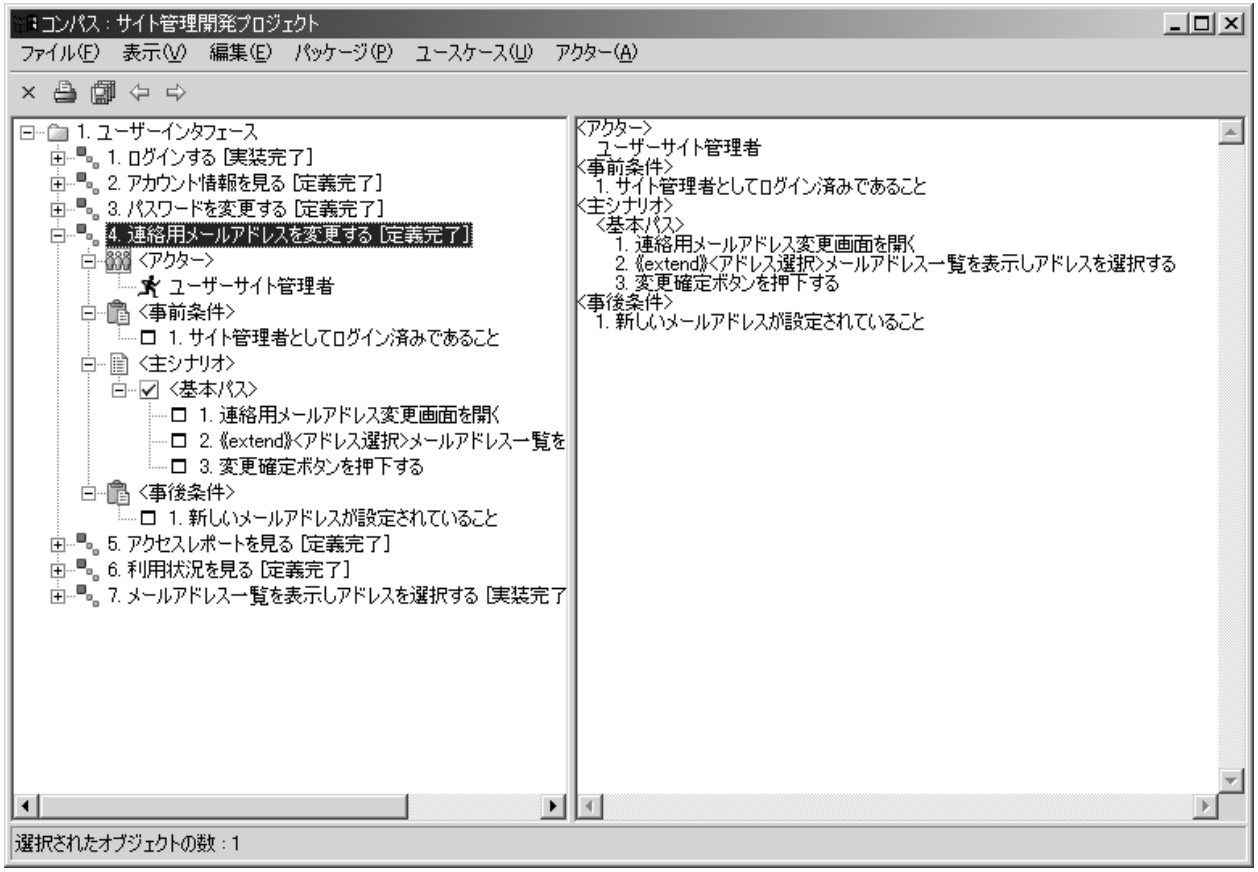


一貫してユースケースを活用できる

ユースケースは、オブジェクト指向開発の軸

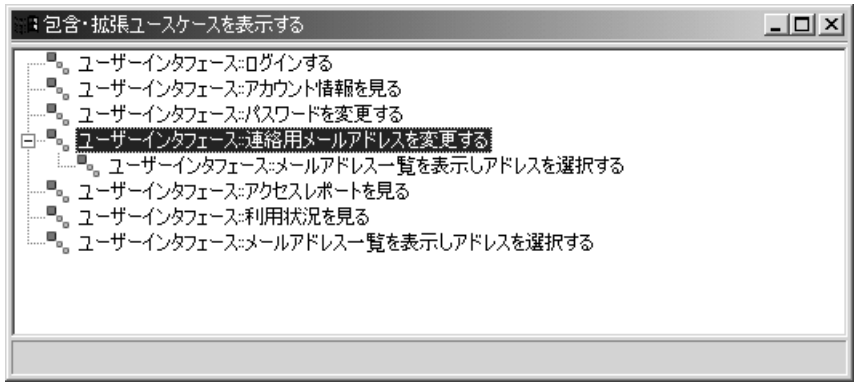
「書いて保存した後は利用しない」を解消

Compass Studio

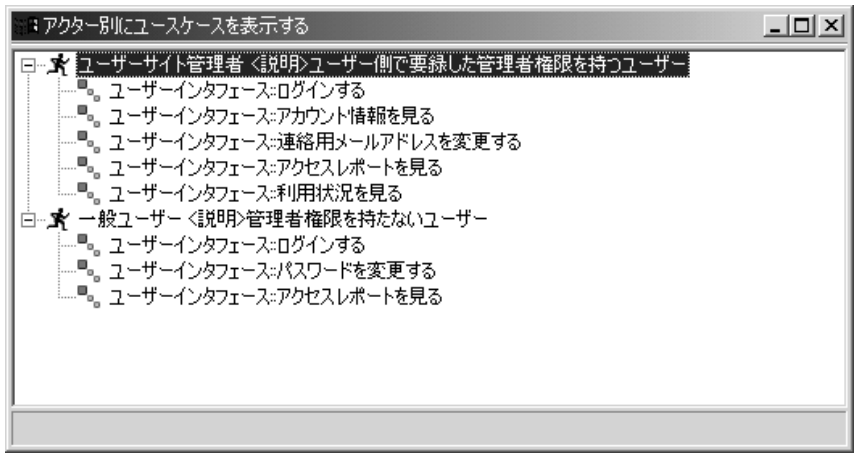


ユースケースの登録･表示･編集･削除
ステータス (進捗度) の登録･表示･編集など

拡張関連、包含関連を表示する



アクター別にユースケースを表示する



本事例の開発規模

- クラス数
  - 全２０５クラス
  - Caché が生成した２５クラス含む
- ステップ数
  - 全４１，１５３行（うちコメント１３，５９０行）
  - Caché が生成した２３，４６４行含む（うちコメント４，７４４行）
- 工数（実装）
  - 本事例では専任開発していないので、約４人月
    （初期のドメイン分析・設計は含まず）

本事例で再利用したもの

- MVCモデル
- Factoryパターン
- Singletonパターン
- Chain of Responsibilityパターン
- Stateパターン
- Commandパターン
- SWT/JFaceコンポーネント

ご質問など