モデルベースドテストの技術動向と研究事例

小川 秀人 （株式会社 日立製作所）
Andreas Hoffmann (Fraunhofer FOKUS)
Ina Schieferdecker (Fraunhofer FOKUS)

ソフトウェアテストシンポジウム 2007 東京
2007年1月30日

謝辞

- 本講演でご紹介するモデルベースドテスト研究事例は、Fraunhofer FOKUSと（株）日立製作所の共同研究によるものです。
- 本講演資料の一部は、Fraunhofer FOKUSによる下記講演資料の和訳です。
- 講演資料の転載をご快諾いただいたFraunhofer FOKUSの皆様に感謝いたします。
  - “Model Transformers for Test Generation from System Models”
    - Presenter: A. Hoffmann
    - Authors: M. Busch, R. Chaparadza, Z. R. Dai, A. Hoffmann, L. Lacmene, T. Ngwangwen, G. C. Ndem, H. Ogawa, D. Serbanescu, I. Schieferdecker, J. Zander-Nowicka
    - Presented at CONQUEST 2006.

目次

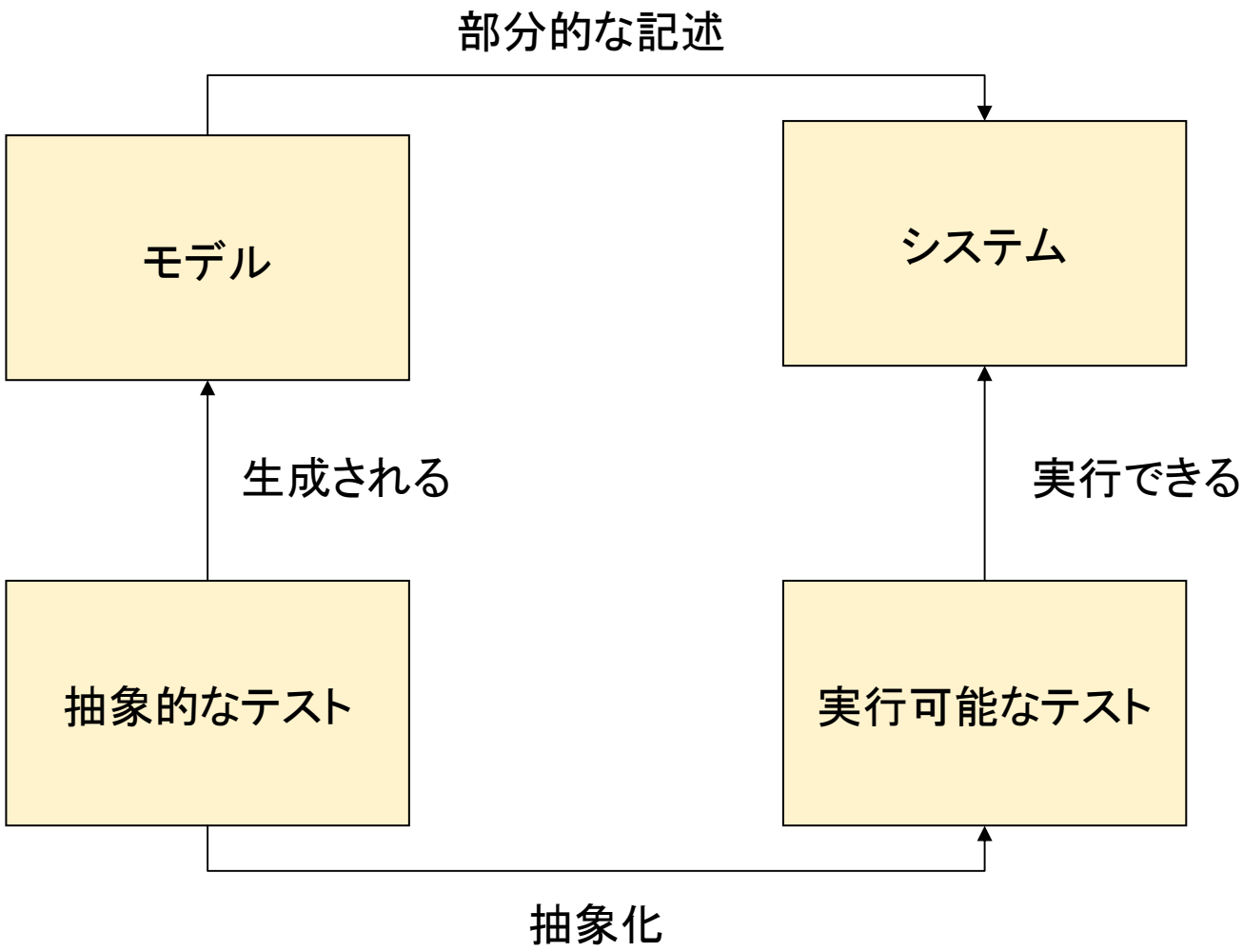
1. モデルベースドテストとは何か
2. テストのモデリング
3. システムモデルからのテストの生成
4. 研究事例紹介
5. まとめ

モデルベースドテストとは何か

- Model-based testing is
  - Software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of system under test (SUT)　[Wikipedia]

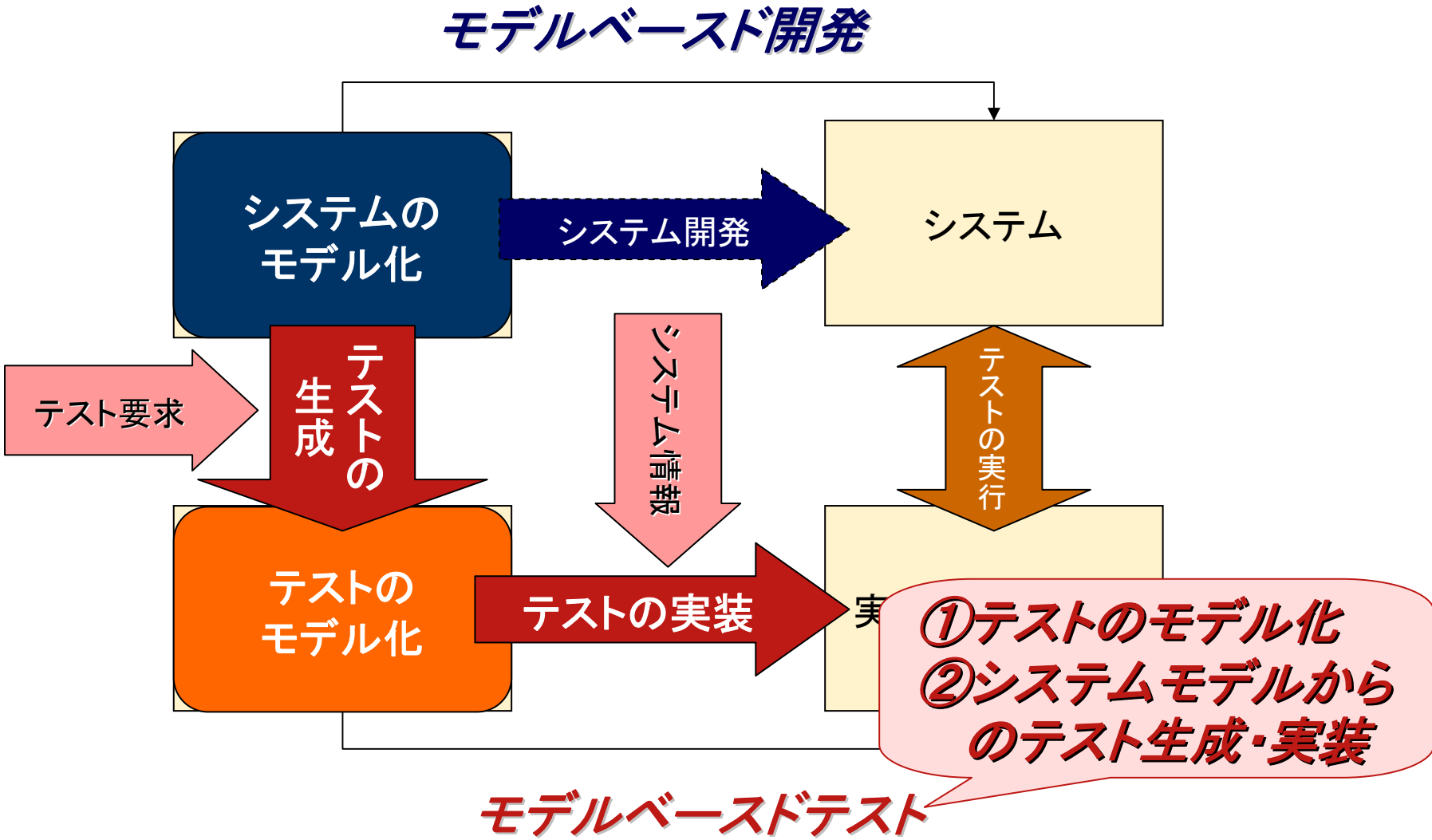
- モデルベースドテストとは、
  - テスト対象システム（SUT）の性質を記述したモデルから
  - 全部または一部のテストケースが生成される
  - ソフトウェアテスト

モデルベースドテストの構成要素



[Wikipedia]より和訳

モデルベースドテストの技術要素



[Wikipedia]に加筆

目次

1. モデルベースドテストとは何か
2. テストのモデリング
3. システムモデルからのテストの生成
4. 研究事例紹介
5. まとめ

テストのモデリング

- テストを（抽象的に）表現
  - テストデータ
    - テストデータを抽象的に表現しよう
  - テストの振る舞い
    - テスト動作を表現しよう
  - テストの結果の評価
    - 成功か失敗かの判断を表現しよう
  - テストの構造
    - テストを構造化しよう

テストモデリング言語の例

- TTCN-3
  - “Testing and Test Control Notation Version 3”
  - ETSI, ITU-T
  - http://www.ttcn-3.org/
  - 2001.6 ver.1.1.2 → 2005.6 ver.3.1.1
- U2TP
  - “UML 2.0 Testing Profile”
  - Object Management Group
  - http://www.omg.org/technology/documents/formal/test_profile.htm
  - 2005.7 formal
- xUnit ?
- テストツールの専用言語?
- 汎用言語によるテスト記述?

TTCN-3の概要

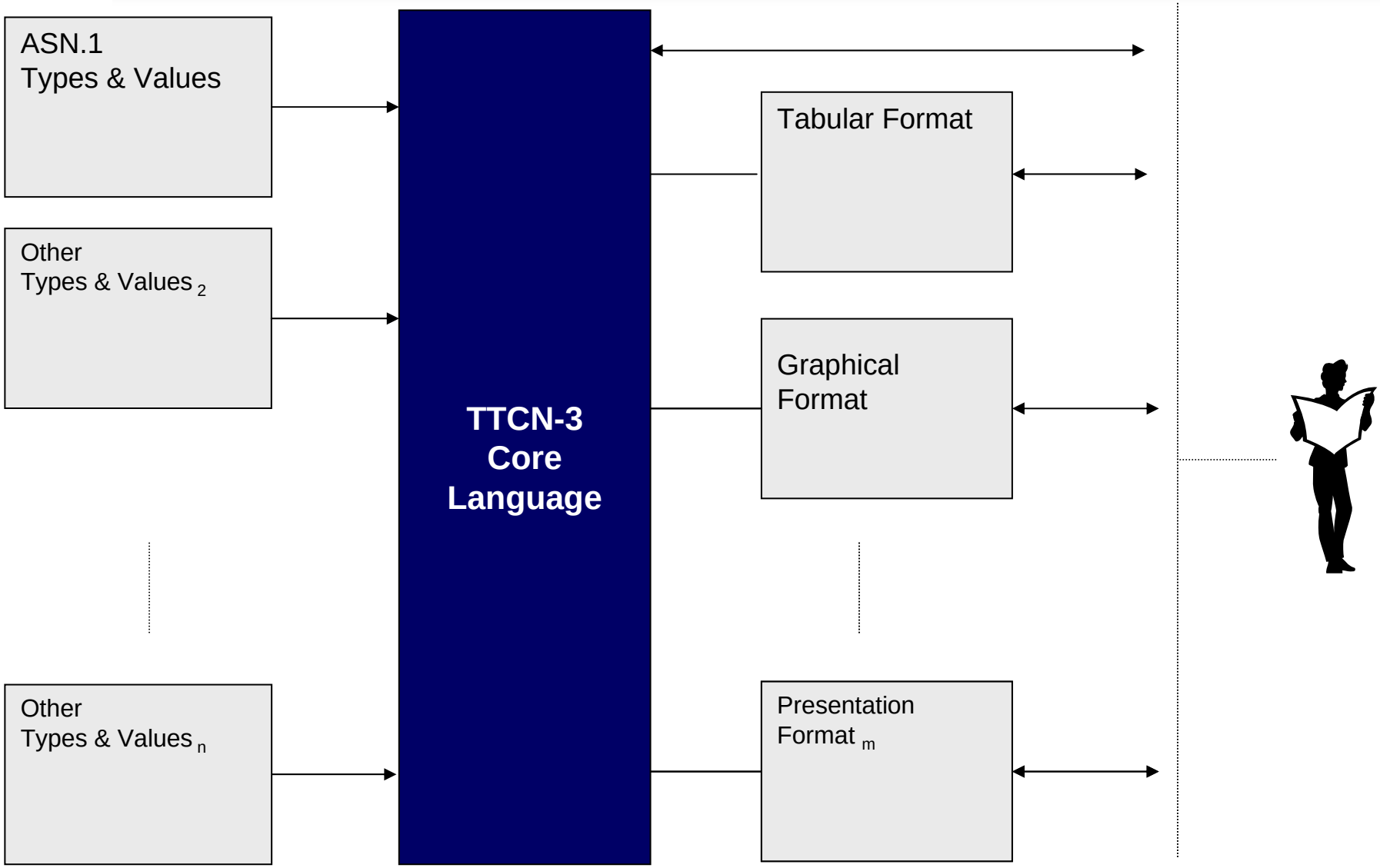
- “Testing and Test Control Notation Version 3”
- ETSI Technical Committee MTS (Methods for Testing and Specification) により開発、継続的にメンテナンス
- 15年以上*にわたり、標準化や産業界での利用実績
  - ETSIにおける 適合テストの実装
    - Session Initiation Protocol (SIP) for Voice over IP (VoIP),
      IPv6 (Core, Mobility, Security), Digital Mobile Radio (DMR) ,
      WiMax (HyperLan), 3GPP IP Multimedia Subsystem (IMS)　など
  - 産業界での利用
    - 通信、自動車、鉄道、金融など様々な分野で利用
    - 特に、欧州の通信機器メーカでの利用事例が多く発表されている
      - ✧ TTCN-3 Users Conferenceなど
  - テストの種類
    - ソフトウェアモジュールテスト
    - レイヤ/モジュールテスト
    - 統合テスト
    - 分散システムのテスト

* TTCN-1 からの実績を含む

TTCN-3の主な特徴

- コア言語と多様な表現形式
- テストケース
  - システムを評価する動作の記述
- テンプレートとマッチング
  - テストデータの記述と、値のパターンマッチング
- バーディクト
  - テスト結果のハンドリング
- モジュール化
- テスト・コンフィグレーション
  - SUTとのインタフェース定義
  - テストコンポーネントの動的生成
  - 並行、分散テストの記述

TTCN-3の言語



TTCN-3のテストデータ

- ユーザ定義型

```
type record MyRecordType {
  integer field1 optional,
  charstring field2,
  boolean field3
}
```

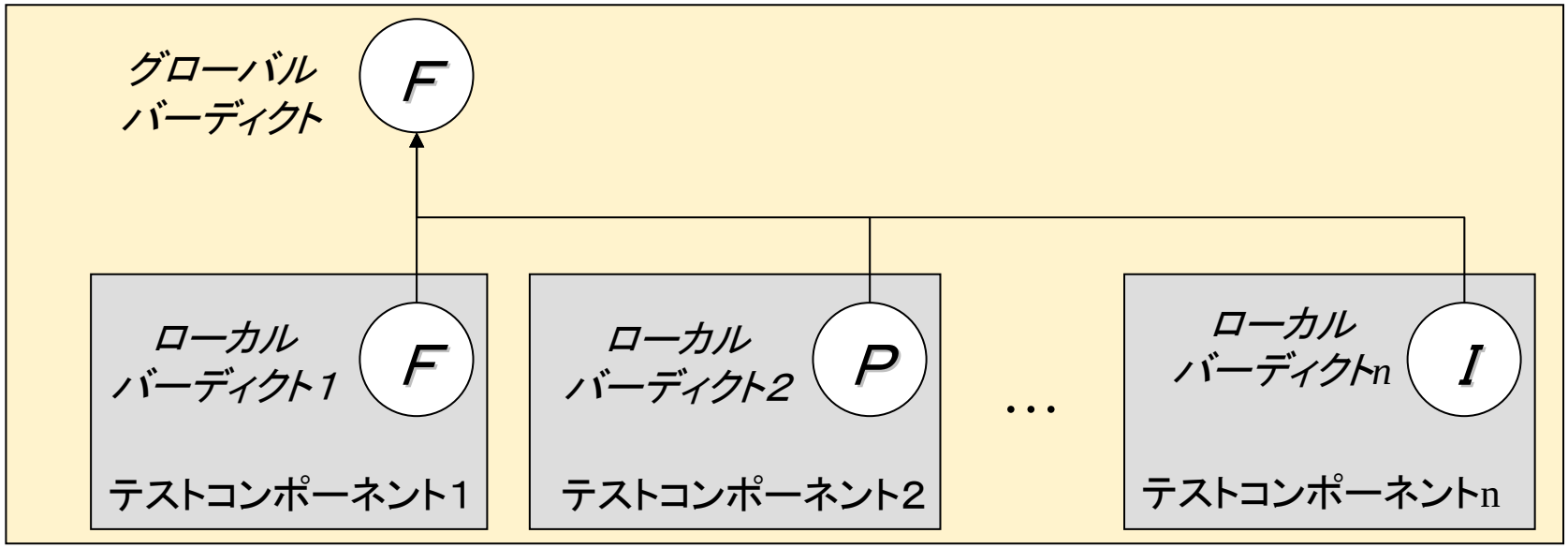
- テンプレート
  - 値に名前をつけて取り扱う
  - テストデータの抽象化・再利用

```
template MyRecordType test_data1 := {
  field1 := ?,
  field2 := pattern “JaSST in * 20*",
  field3 := true
}
```

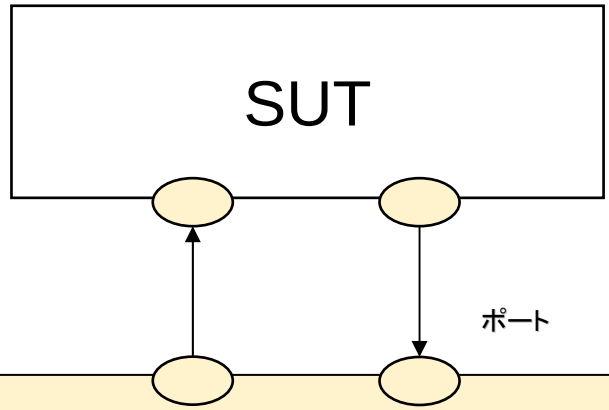
- マッチング
  - テスト結果の判定のためにテストデータを評価
  - ワイルドカード(*, ?)

TTCN-3のバーディクト

- バーディクト： テスト結果をハンドリングする仕組み
  - none < pass < inconc < fail < error
- 各テストコンポーネントがローカル・バーディクトを持つ
  - コンポーネントが読み書き
- 各テストケースは、グローバル・バーディクトを持つ



TTCN-3のテストケース



Port1.Send(Stimulus)

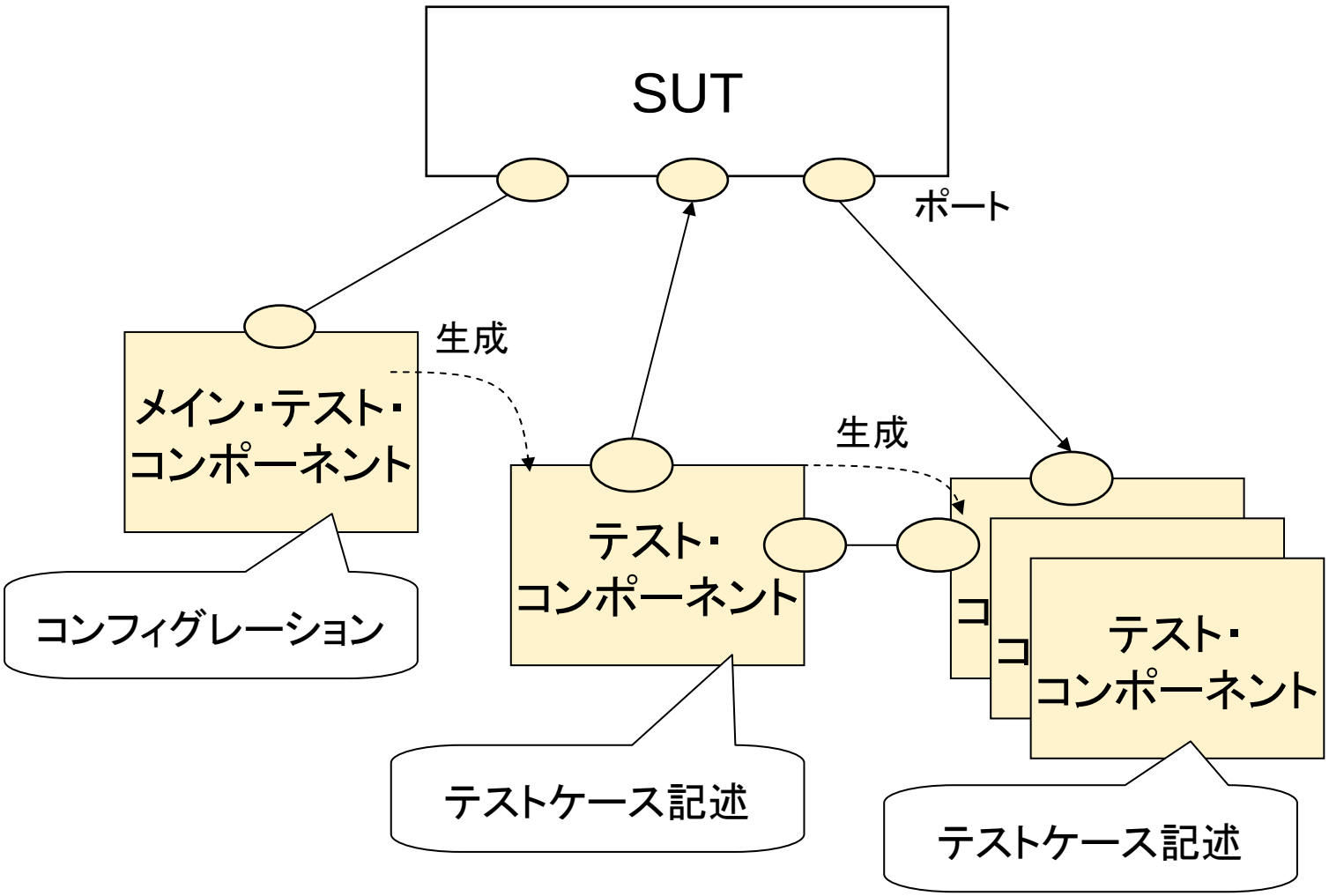
```
port1.send(Stimulus_Template);

alt {
   [ ] port2.receive(operationComplete_Type:
                        {response, true, ?})
      { setverdict(pass); }
   [ ] port2.receive(operationComplete_Type:?)
      -> value erroneous
      { setverdict(fail); }
   [ ] ...
}
```

Port2.Recieve(Response)

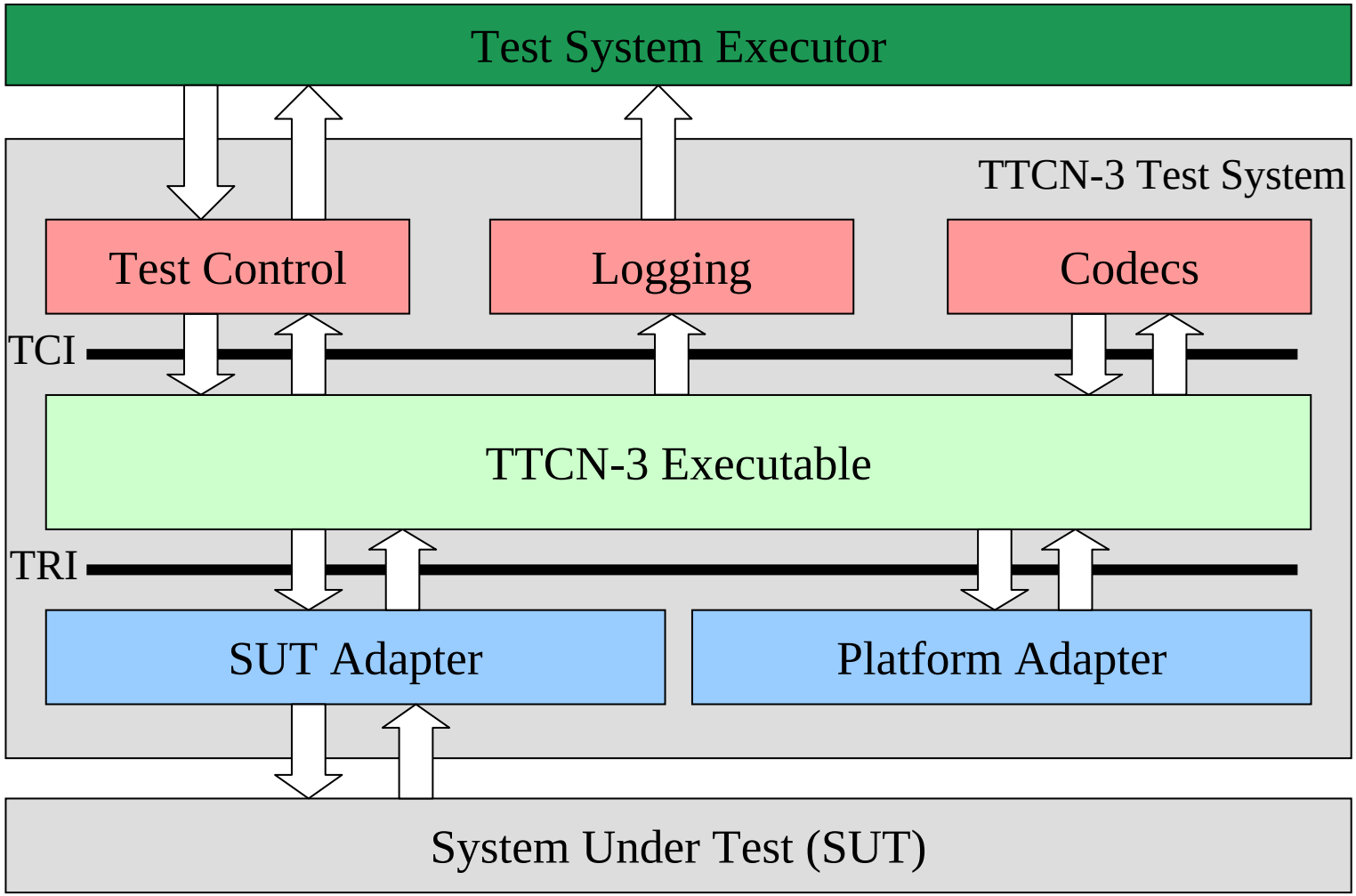
レスポンスの評価

バーディクトのセット

TTCN-3のモジュール化



TTCN-3のテストシステム



TCI = TTCN-3 Control Interface
TRI = TTCN-3 Runtime Interface

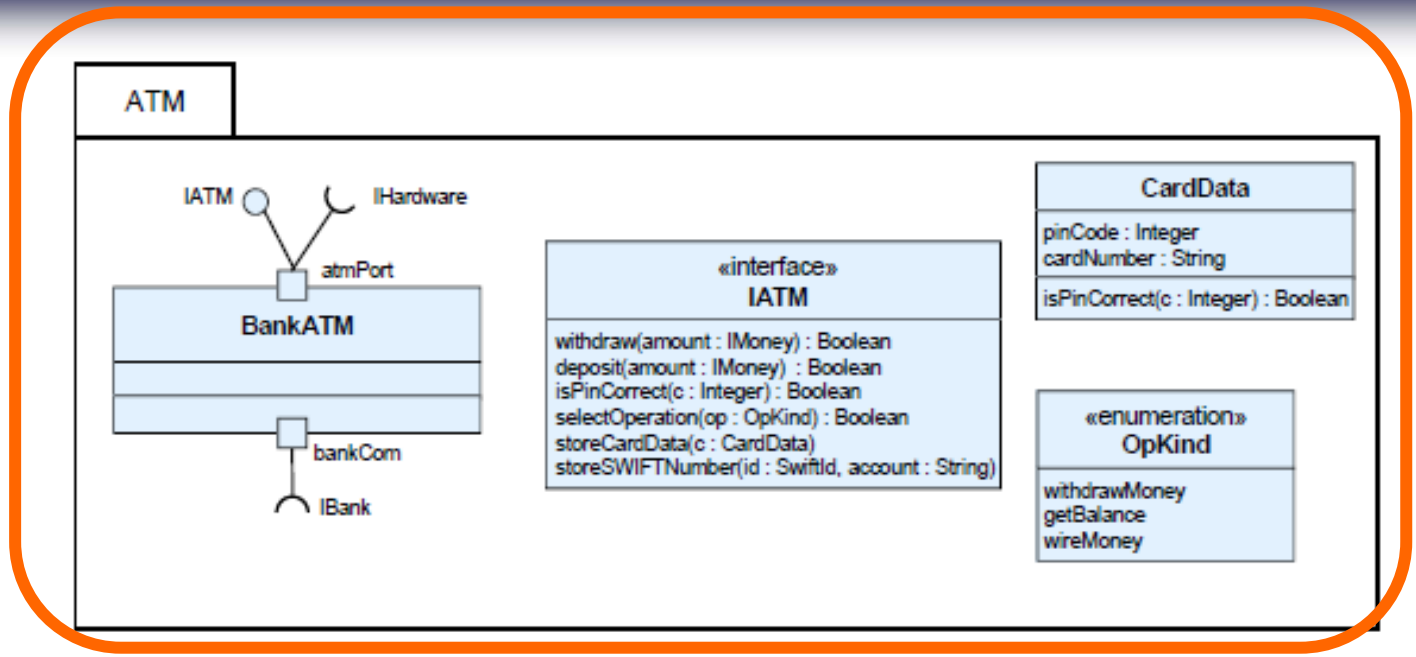
[http://www.ttcn-3.org/Testsystems.htm]

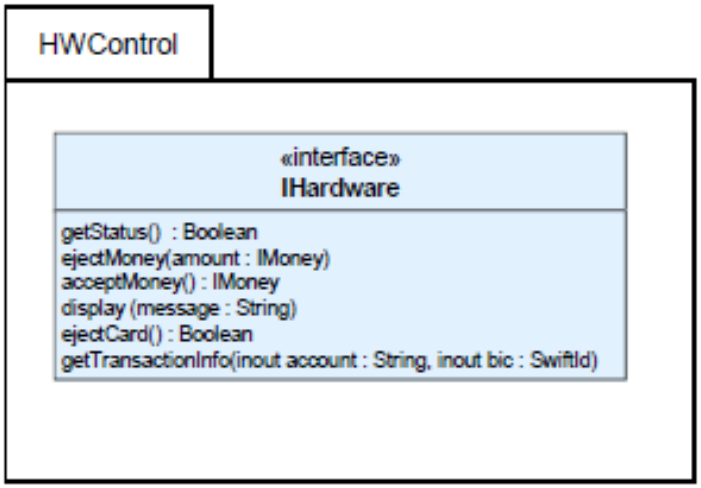
U2TPの概要

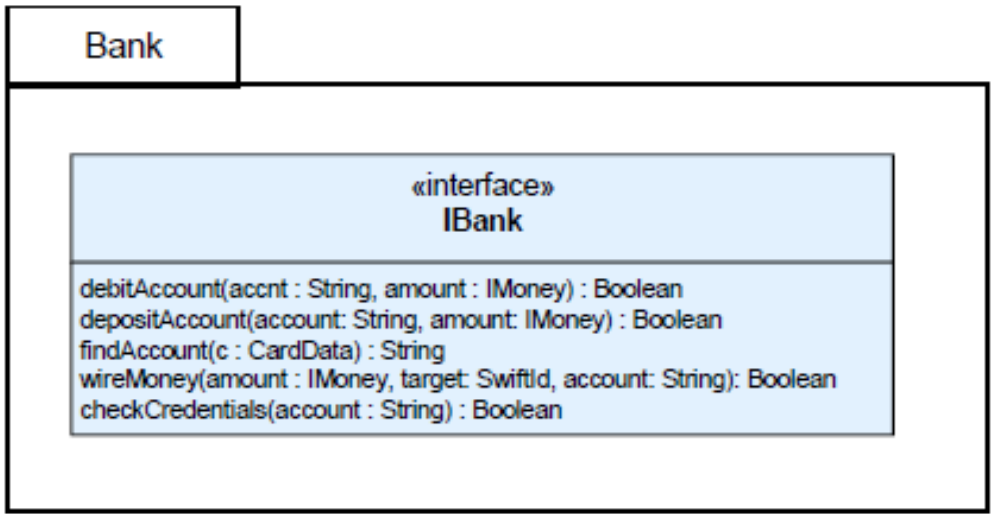
- “UML 2.0 Testing Profile”
- UML 2.0 にテストコンセプトを導入
- U2TPの構成要素
  - テスト・アーキテクチャ
    - テストコンポーネント、テストコンフィグレーション、テストコントロールなど
  - テスト・ビヘイビア
    - デフォルト、テストケース、バーディクト、テストログなど
  - テスト・データ
    - ワイルドカード、データプール、データパーティションなど
  - テスト・タイム
    - タイマ、タイムゾーンなど

システムの例:銀行ATM

[U2TP]

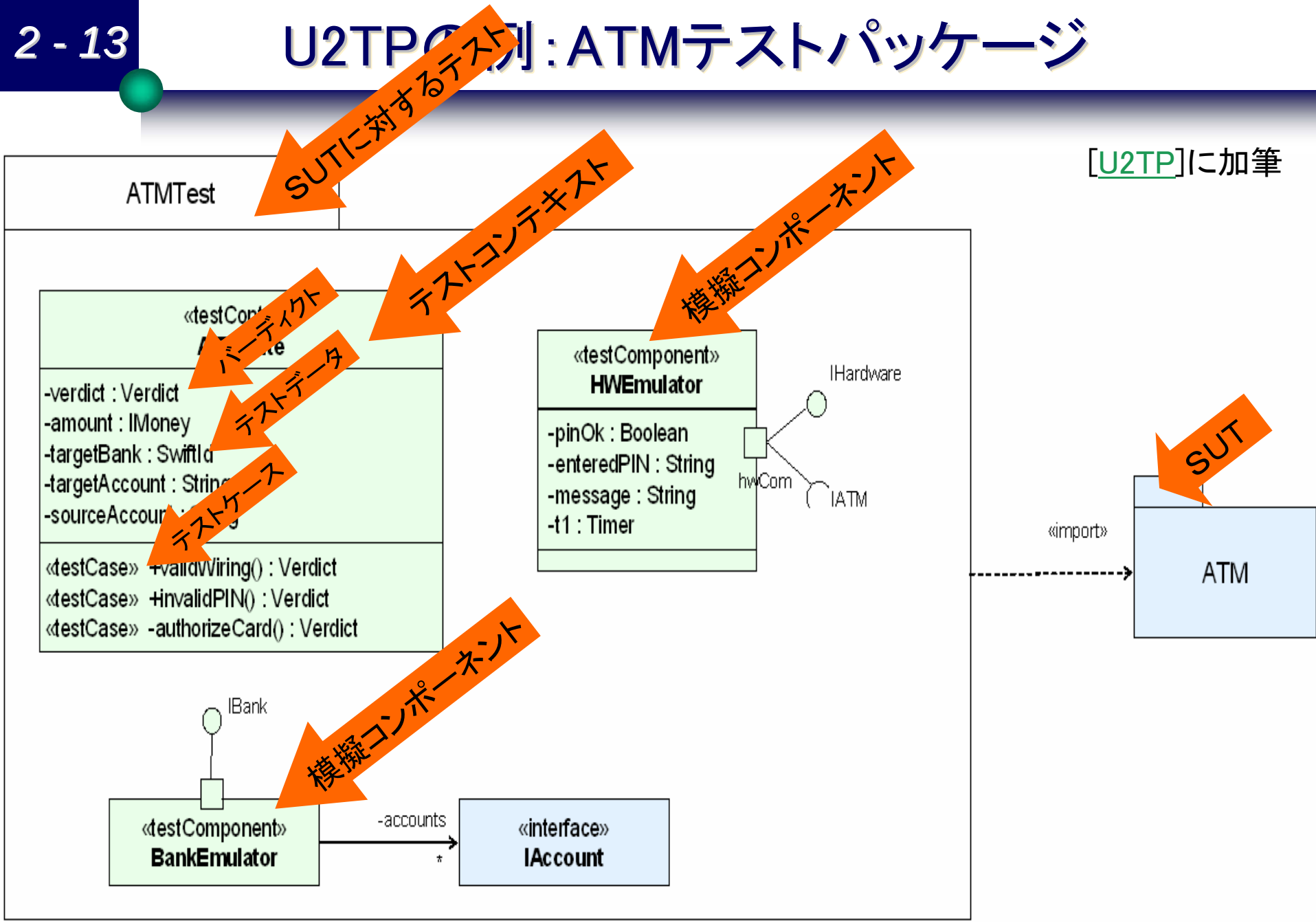




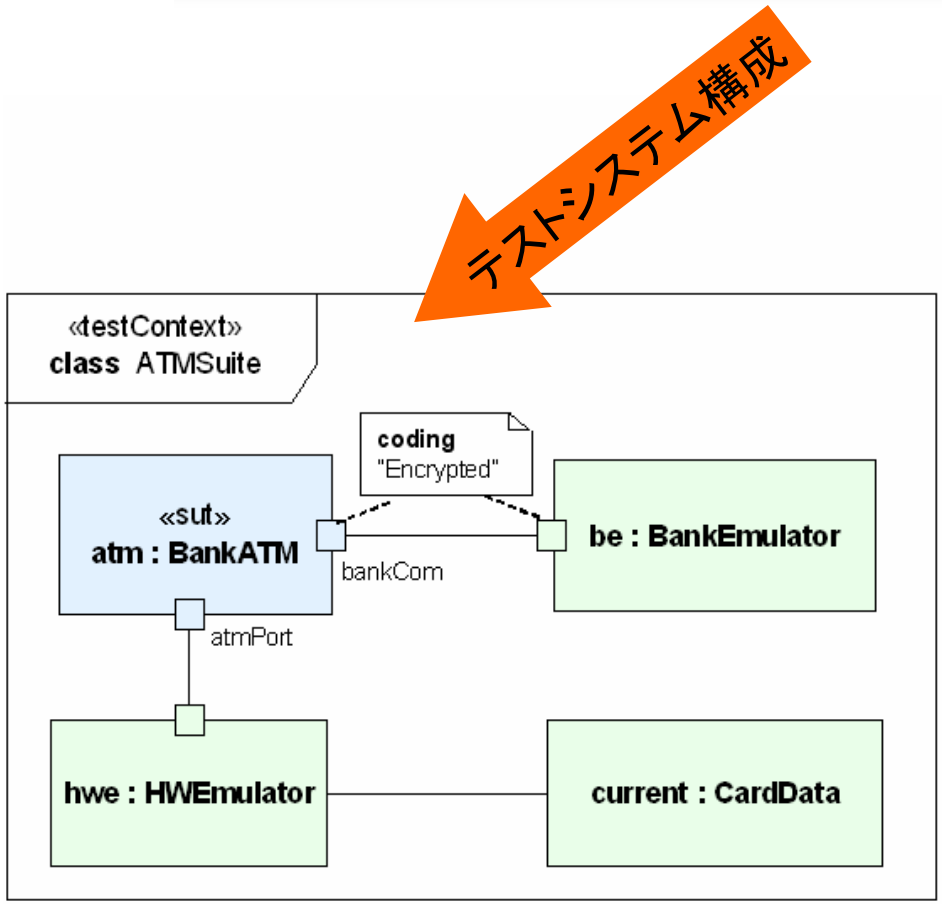


U2TP例：ATMテストパッケージ

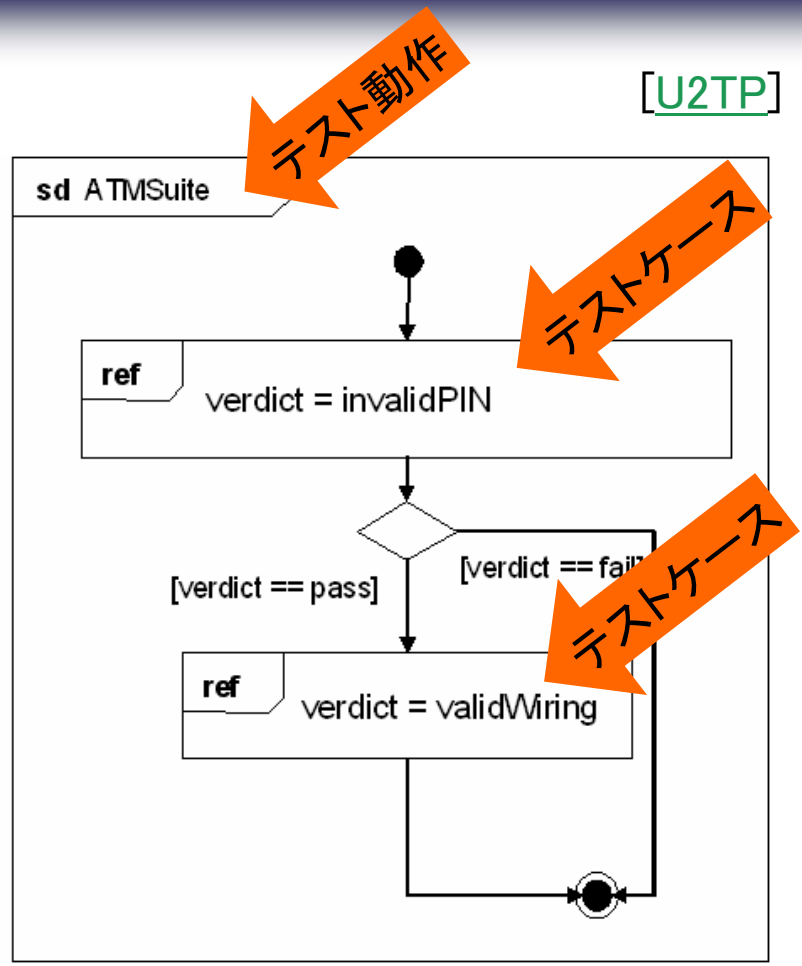
[U2TP]に加筆



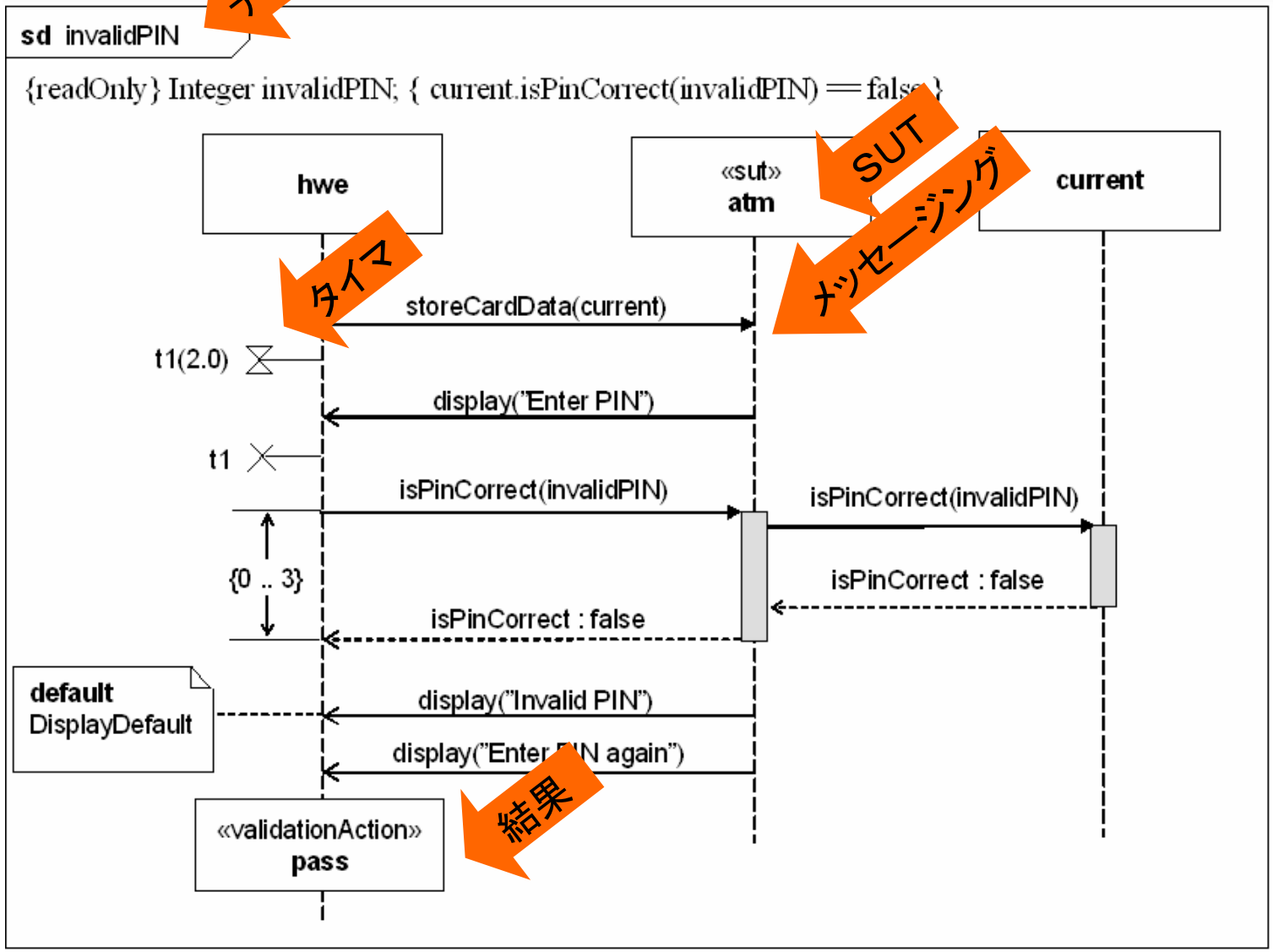
U2TPの例：テストコンテキスト



[U2TP]



U2TPの例：テストケース



[U2TP]

目次

1. モデルベースドテストとは何か
2. テストのモデリング
3. システムモデルからのテストの生成
4. 研究事例紹介
5. まとめ

システムモデルからのテスト生成

- 生成するもの
  - テストの内容
    - テスト動作
    - テスト入力と期待出力
    - 結果出力の評価
    - テスト結果の評価
  - テストシステムの構造
    - テストシステムとSUTとの間のインタフェース
    - テストシステムの実行方法
    - テストシステム内の構造

テストケースの生成

- システムのモデル　→　テストケース
  - *広い意味では、従来のテスト技法も該当する*
- 従来テスト技法の例：テストデータの生成
  - 同値クラス　→　限界値・境界値

クラス	値域
立見あり	101 ≦ N ≦ 130
着席	1 ≦ N ≦ 100
聴衆なし	N = 0

⇨ テストデータ
-1, 0, 1, 100, 101, 130, 131…

- 従来テスト技法の例：テストデータの組合せの生成
  - 因子、水準　→　直行表、All-Pair

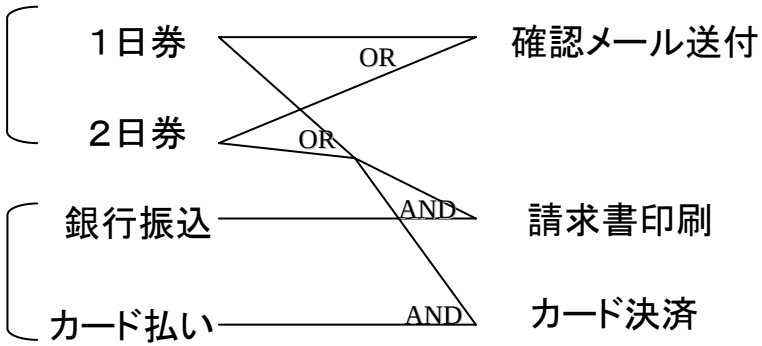
聴衆	気温	笑い
立見あり	高	あり
着席	適正	なし
聴衆なし	低	

⇨

#	聴衆	気温	笑い	結果
1	立見あり	高	あり	盛況
2	着席	高	なし	睡眠
3	聴衆なし	適正	なし	悲哀
…	…	…	…	

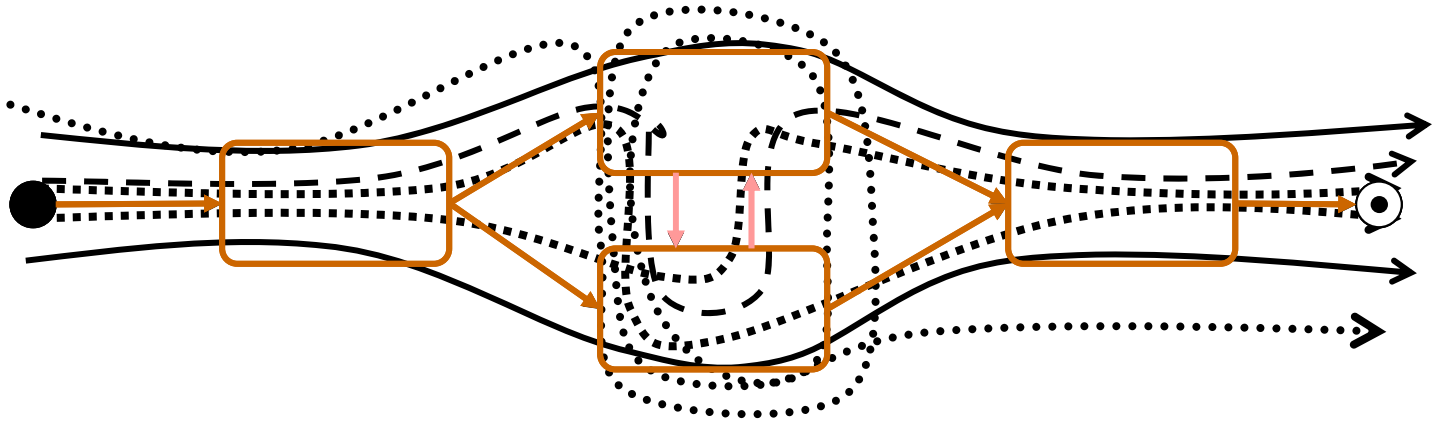
テストケースの生成（続き）

- 従来テスト技法の例：原因結果グラフ → 決定表



#	1	2	3	…
券種	１日券	１日券	２日券	…
支払い方法	振込	カード	振込	…
メール送付	○	○	○	…
請求書印刷	○	×	○	…
カード決済	×	○	×	…

- 従来テスト技法の例：状態遷移モデル → パステスト

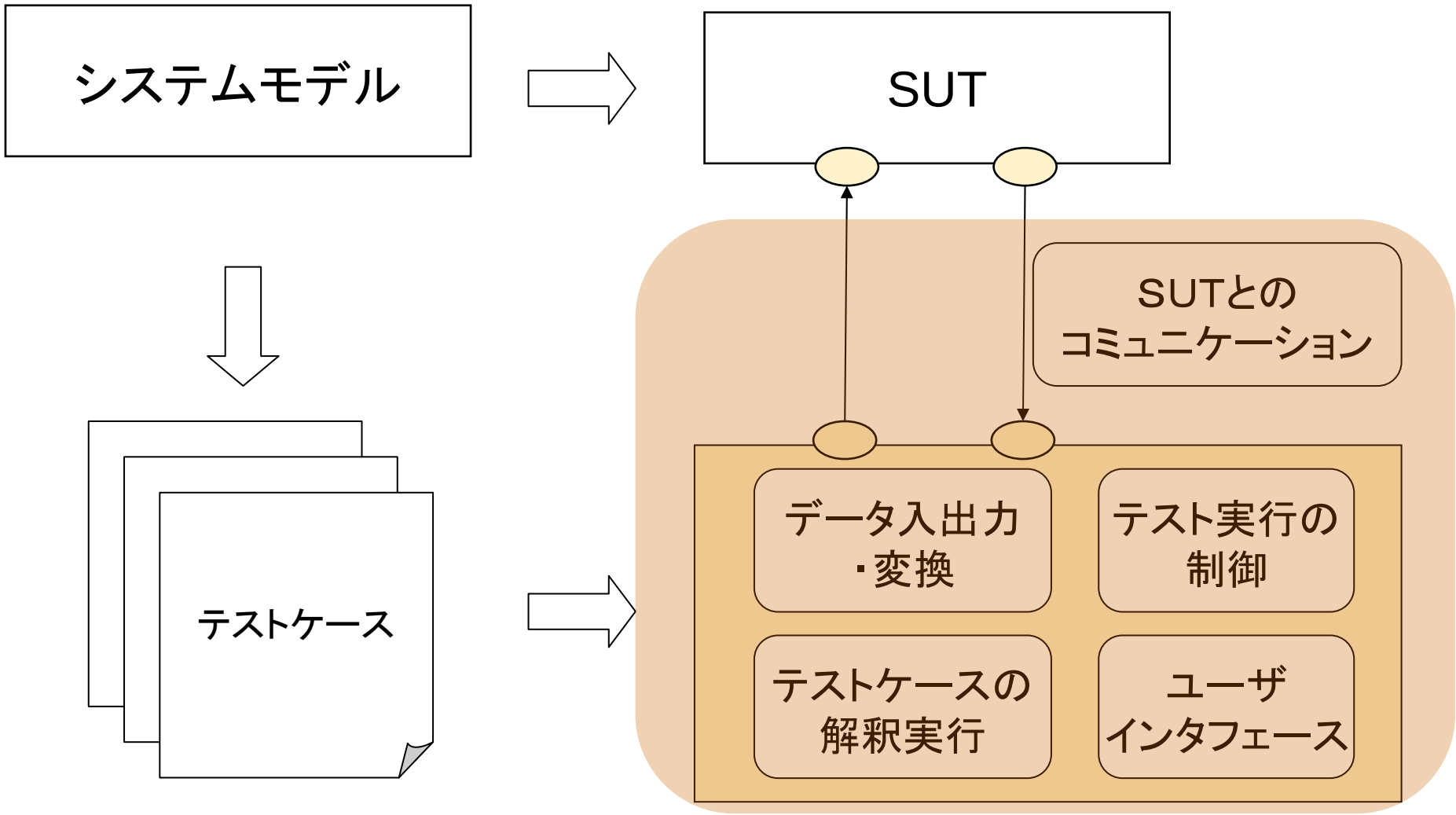


- モデルベースドテストは、これらの歴史を踏まえながら、より統合的、形式的、標準的に

テストケース生成の手法

- 有限状態機械（FSM）、状態遷移システム
- シナリオベーステストケース生成
- 定理証明を用いたテストケース生成
- 制約論理を用いたテストケース生成
- モデルチェッキングを用いたテストケース生成
- シンボル実行を用いたテストケース生成

テストシステムの構造の生成



モデルベースドテストの研究例

- IRISA, VERIMAG: TGV (Test Generation with Verification)
  - IOLTS (Input/Output Labeled Transitions Systems)
- IBM: AGEDIS (Automated Generation and Execution of Test Suites in Distributed Component-based Software)
  - AGEDIS Modeling Language
    (UMLベース;オブジェクトダイアグラム, ステートダイアグラム, IFアクション言語)
- Microsoft: AsmL Test Tool, Spec Explorer
  - AsmL: Abstract State Machine Language
- Telelogic: Tau
  - SDL: System Description Language
- Conformiq: Conformiq Qtronic
  - UML ステートダイアグラム
- ISPRAS: UniTesK
  - 専用仕様記述言語
- Reactive Systems: Reactis
  - Matlab/Simulink, Stateflow　　等々

さらに興味のある方は[Wikipedia] [MBTHP]等を参照

目次

1. モデルベースドテストとは何か
2. テストのモデリング
3. システムモデルからのテストの生成
4. 研究事例紹介
5. まとめ

（注）
本章のスライドは，[CONQUEST2006]の日本語訳に一部のスライドを追加したものです．

モデルベースドテスト研究事例の紹介

- OMG MDA(Model Driven Architecture)の概念をテストに拡張
- システムモデル
  - PIM (Platform Independent Model): UML2.0
  - PSM (Platform Specific Model) : EJB
  - 実行コード : Java/EJB, Java/WSDL
- テストモデル
  - PIT (Platform Independent Test): UML2.0 Testing Profile
  - PST (Platform Specific Test) : TTCN-3
  - テストコード : Java
- モデル変換
  - OMG MOF (Meta-Object Facility)

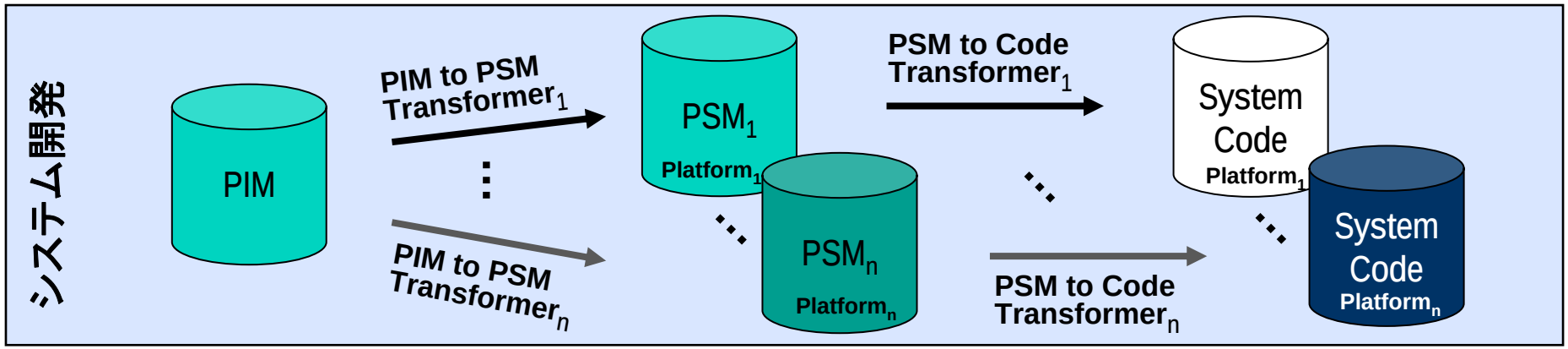
目的と背景

[CONQUEST2006]

- **目的:システムとテストの早期の統合**
  - 設計ミスや実装間違いをシステム開発の早期に摘出
  - 開発時間とコストを短縮
- **Model-Driven Architecture (MDA)**
  - プラットフォーム非依存モデルからプラットフォーム依存モデルへのリファインメント
  - メタモデルに基づいたモデル変換器によるモデル変換
  - しかし、テストに関しては言及なし => MDAを拡張
- **MDAへのテストの統合**
  - MDA, MOF, UML 2.0
  - UML 2.0 の記法を、システムとテストの両方のプラットフォーム非依存モデルに利用
  - EJBとWSDLシステム向けのテスト実行

Model-Driven Architecture [MDA]

[CONQUEST2006]

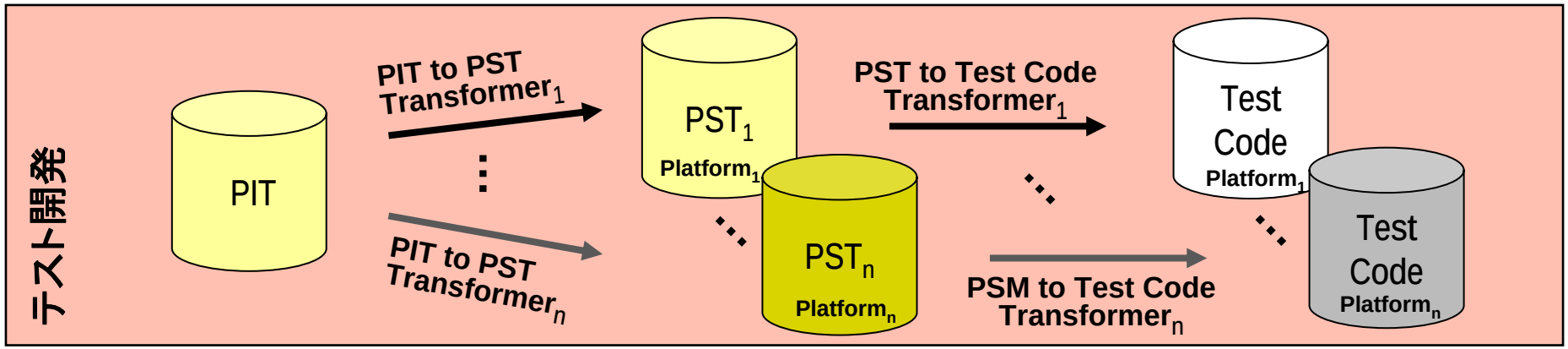


- プラットフォーム非依存モデル (PIM; Platform Independent Model)
  - ターゲットプラットフォームの詳細に依存せず、システム機能を記述
  - 1つのPIMから多くのターゲット向けの実装を生成
  - インタラクティブなリファインメントを行う
- プラットフォーム依存モデル(PSM; Platform Specific Model)への変換
  - ターゲットプラットフォームごとの変換器
  - 半自動/自動的な変換
- 実行コードの生成
  - 変換の完全性は、モデルと変換器の完全性に依存

MDAベースのテストモデリング

[CONQUEST2006]

- テストにシステムモデリングと同様の抽象化
  - プラットフォーム非依存テスト (PIT; Platform Independent Test)
  - プラットフォーム依存テスト (PST; Platform Specific Test)
- １つのPITから複数のPSTへ変換
- PSTからテストコードの生成



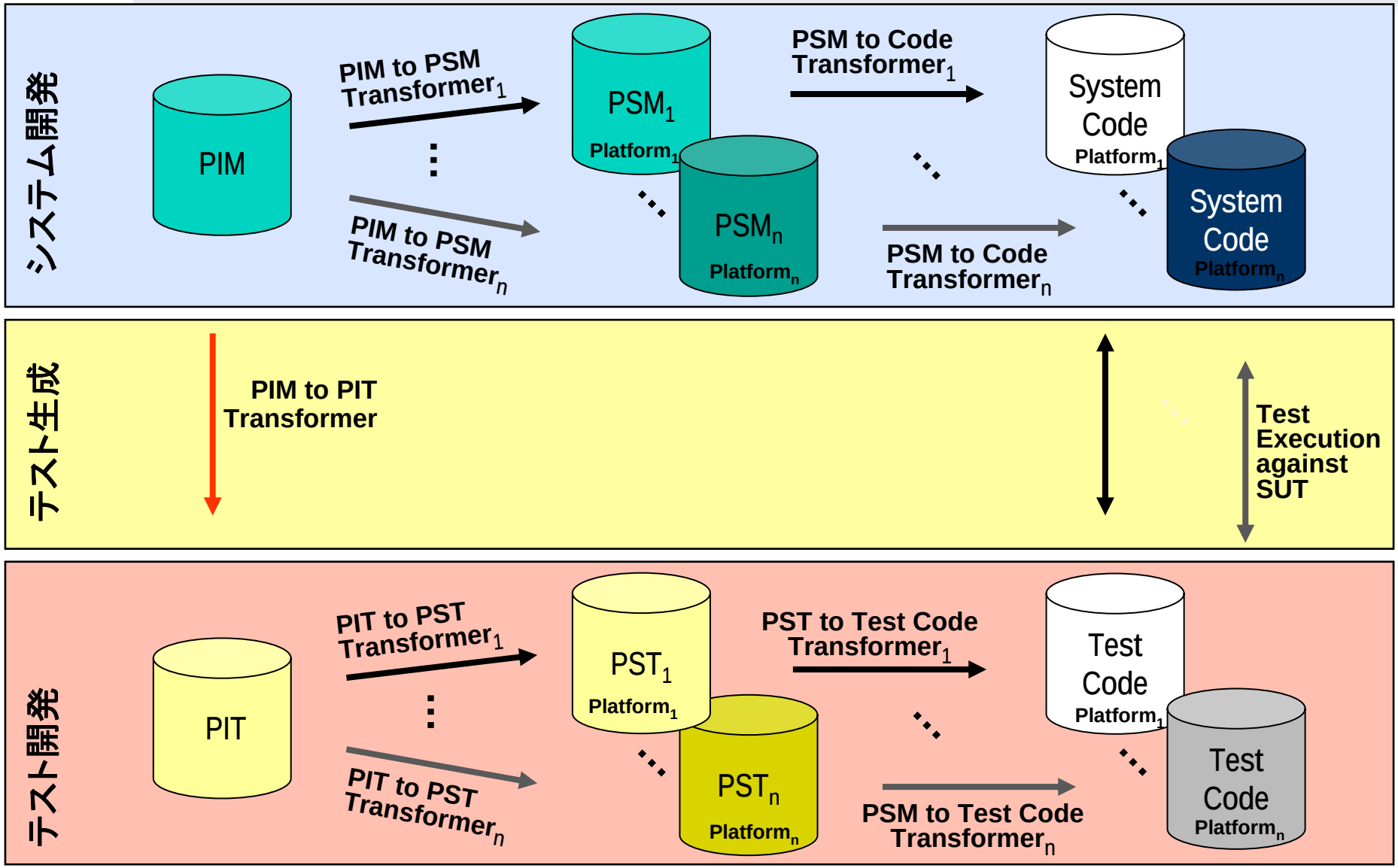
変換器によるテスト生成の意味

[CONQUEST2006]

- なぜシステムモデルからテストを生成するのか？
  - システムモデルの持つシステム情報をテストモデルに再利用
- ブラックボックステストにとってのシステム情報の有効性
  - システム構造、コンフィギュレーション
  - システムのコンポーネントポートで、システムの振る舞いを観察可能
  - 型システム
  - データの具体値

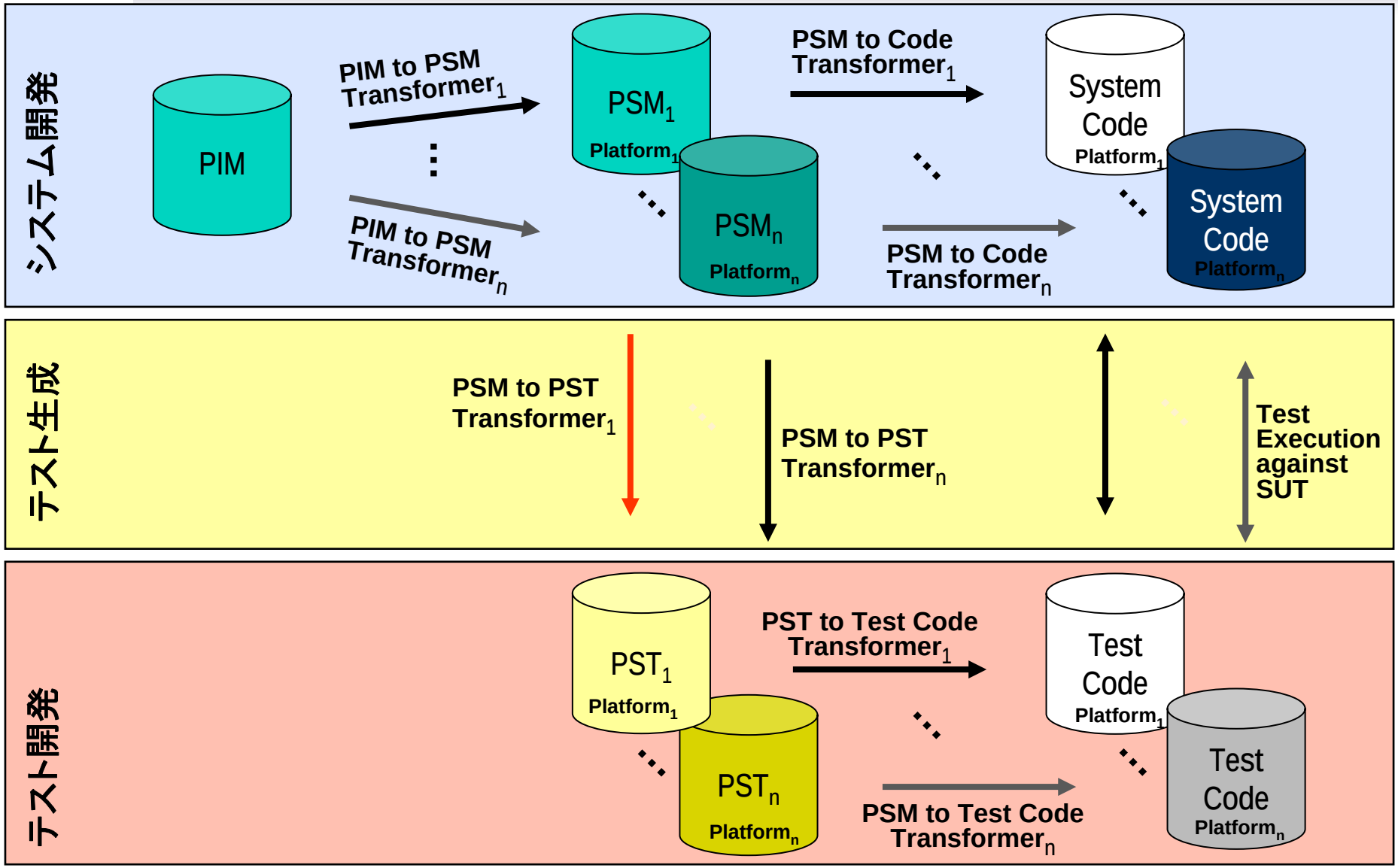
変換器によるテストの生成

[CONQUEST2006]



変換器によるテストの生成

[CONQUEST2006]



コード自動生成におけるテストの意味は？

[CONQUEST2006]

- テストは必要なのか
  - システムコードはシステムモデルから自動的に生成されるのに？
  - テストコードがシステムと同じモデルから生成されるのに？
  - システムコードとテストコードがPIMやSIMの持つ同じ情報をベースにしているのに？
- しかし、システムのテストは必要：
  - システムモデルは完全でない（詳細が省略されている）
  - 変換器が検証されていない、正しくない
  - 変換器が完全でない
  - コードレベルでの変更
  - ターゲット環境で実行することで機能が追加され、リソースを共有する

プラットフォーム非依存モデル

[CONQUEST2006]

- **UML2.0によるシステムとテストの非依存モデル:**
  - PIM ← ***eUML*** (essential UML)
  - PIT ← ***eTML*** (essential Test Modeling Language)
- ***eUML***
  - UML 2.0 の部分集合
    - UML 2.0 全体では範囲が広い
    - UML 2.0 の持つパッケージ・マージ機構の実装が難しい
  - UNL 2.0 の持つ13種類の図のうち、主要な5図を利用
- ***eTML***
  - eUMLを拡張し、テストモデルを導入
  - OMG標準のUML 2.0 Testing Profileの部分集合

プラットフォーム依存モデル

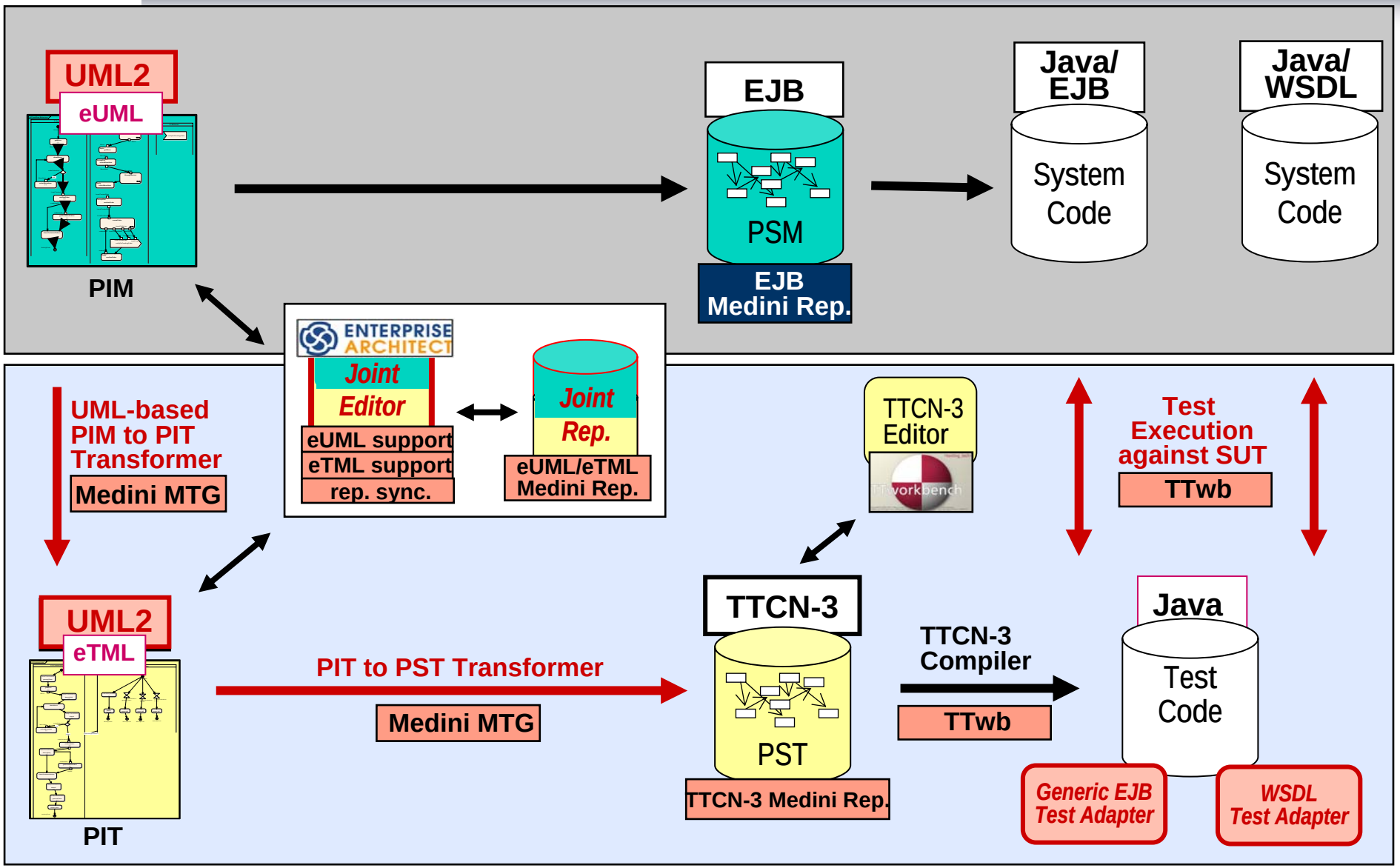
[CONQUEST2006]

- **PSTとしてTTCN-3を利用**
  - テスト仕様記述・実装のための言語
  - 広く利用されている
  - ETSIにより標準化・メンテナンス
  - 多くのベンダから強力なツールが提供されている
  - 様々なターゲットシステム、OS、実装言語向けの実行可能テストの自動生成がサポートされている

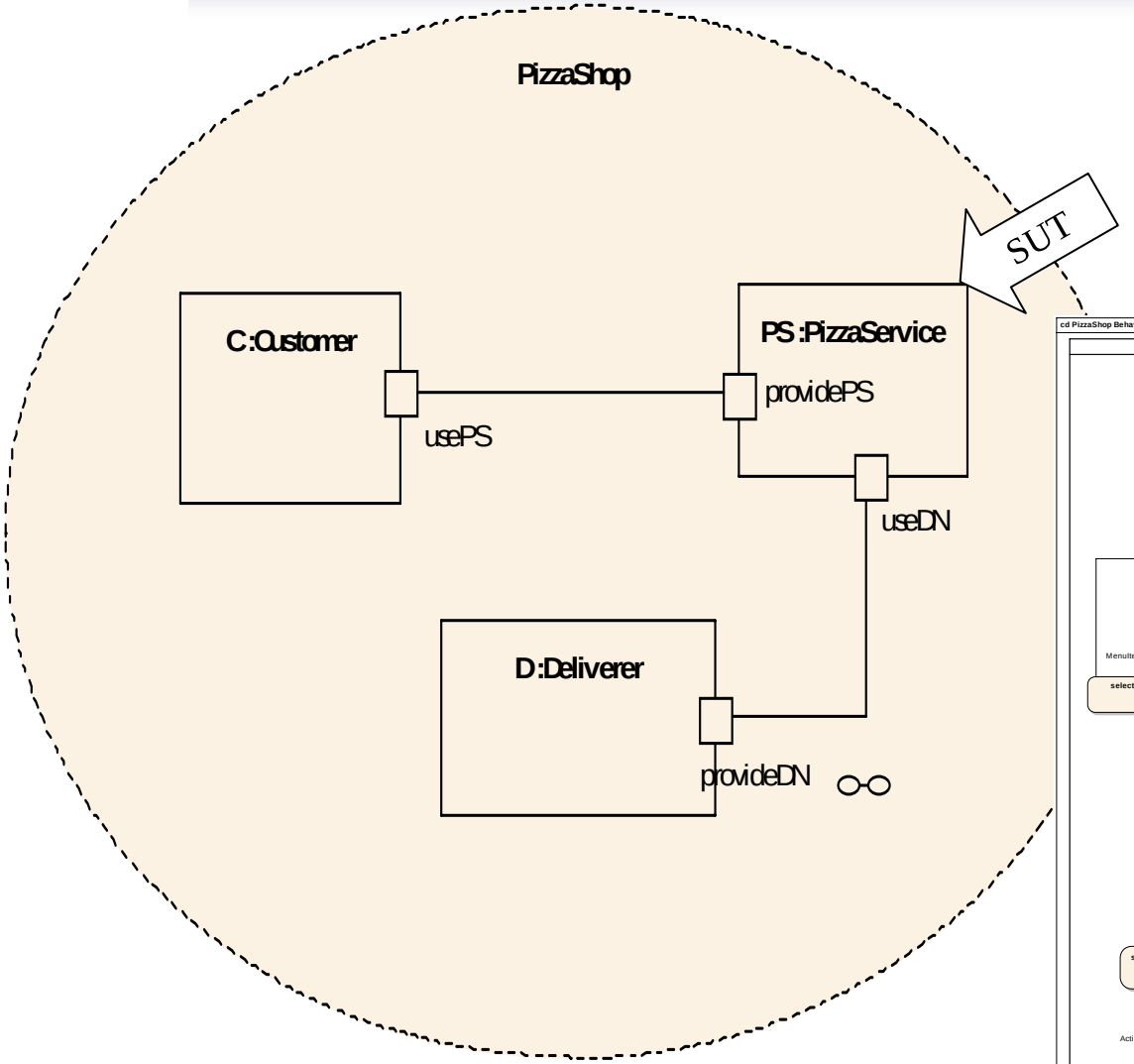
- **TTCN-3 は、UMLによるプラットフォーム非依存モデルからの生成開発アプローチに有用である**

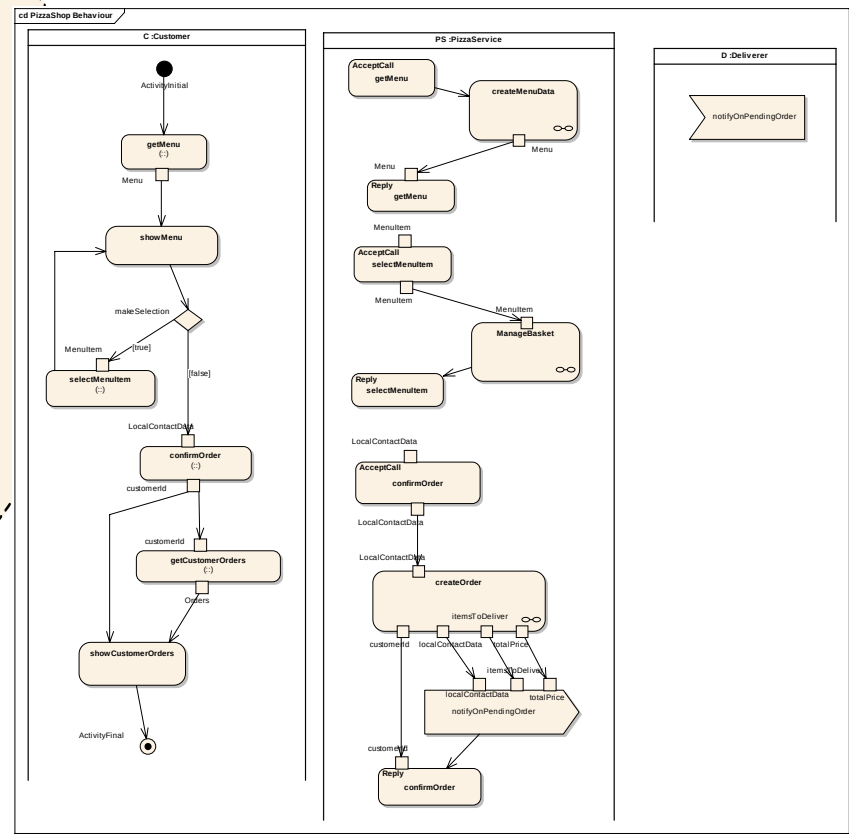
ツール・チェイン

[CONQUEST2006]



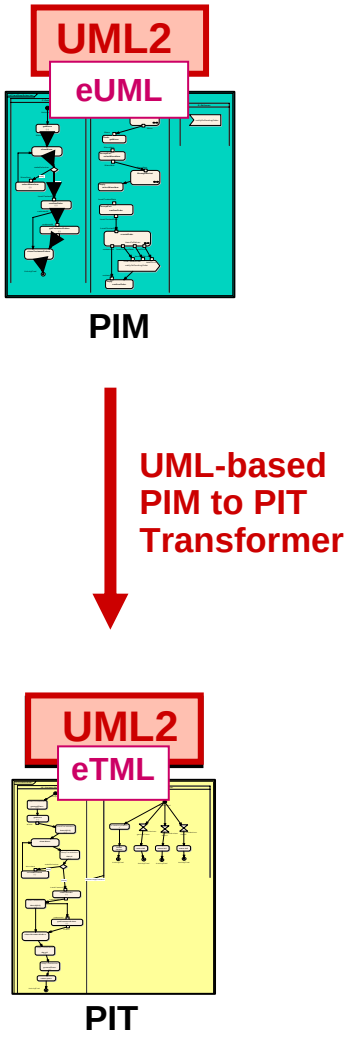
例:ピザショップシステム





変換ルール

[CONQUEST2006]



- *eUML* モデルから、eTMLモデルへの変換ルール

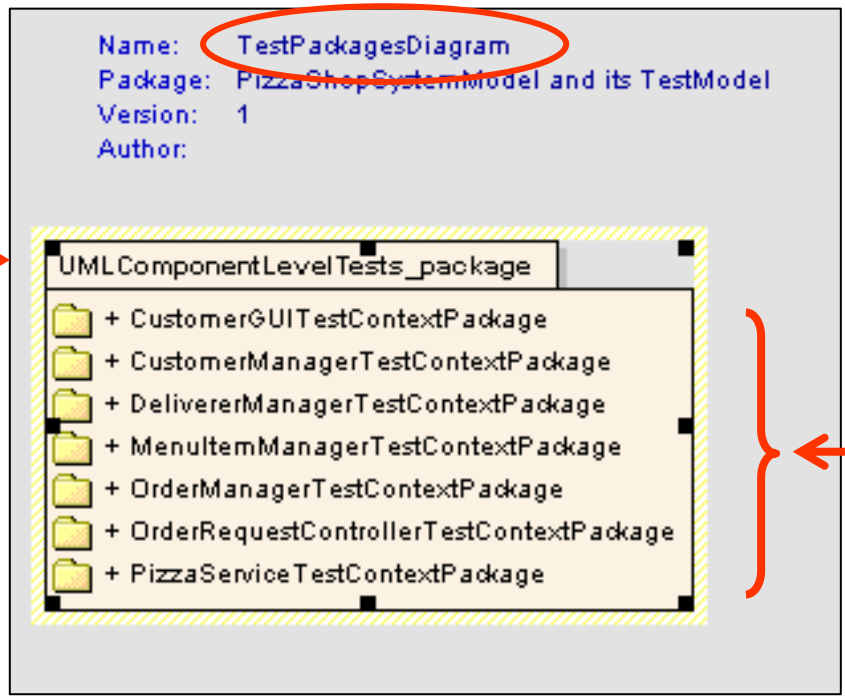
- **変換ルール1:**
  eUMLモデルの全要素を、eTMLモデルの初期状態に

- **変換ルール2:**
  システムモデルと並列してテストパッケージを生成
  →"TestPackagesDiagram"

変換ルール:テストモデル構造の生成

[CONQUEST2006]

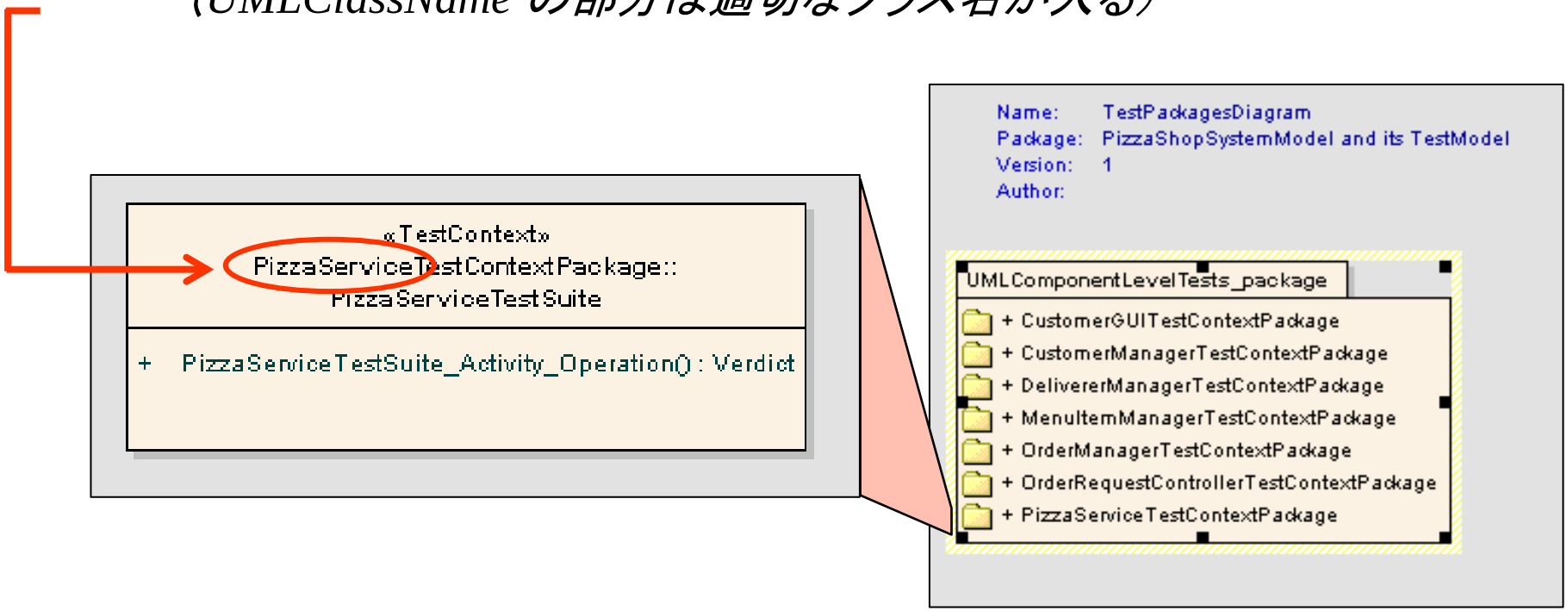
- **変換ルール3:**
  - *テストパッケージの内部にパッケージを作成　→*
    "UMLComponentLevelTests_package".
  - *各テスト可能なUMLクラスに対して、サブパッケージを作成*



変換ルール:テストコンテキストの生成 [CONQUEST2006]

変換ルール4:

- *各"UMLComponentLevelTests_package "クラスの内部に、*
  <<TestContext>>ステレオタイプのクラスを作成
  *→"UMLClassName**TestContextPackage**"*
  (UMLClassName の部分は適切なクラス名が入る)



変換ルール：テストコンフィグレーション・テストコンポーネントの生成

[CONQUEST2006]

他の変換の概要:

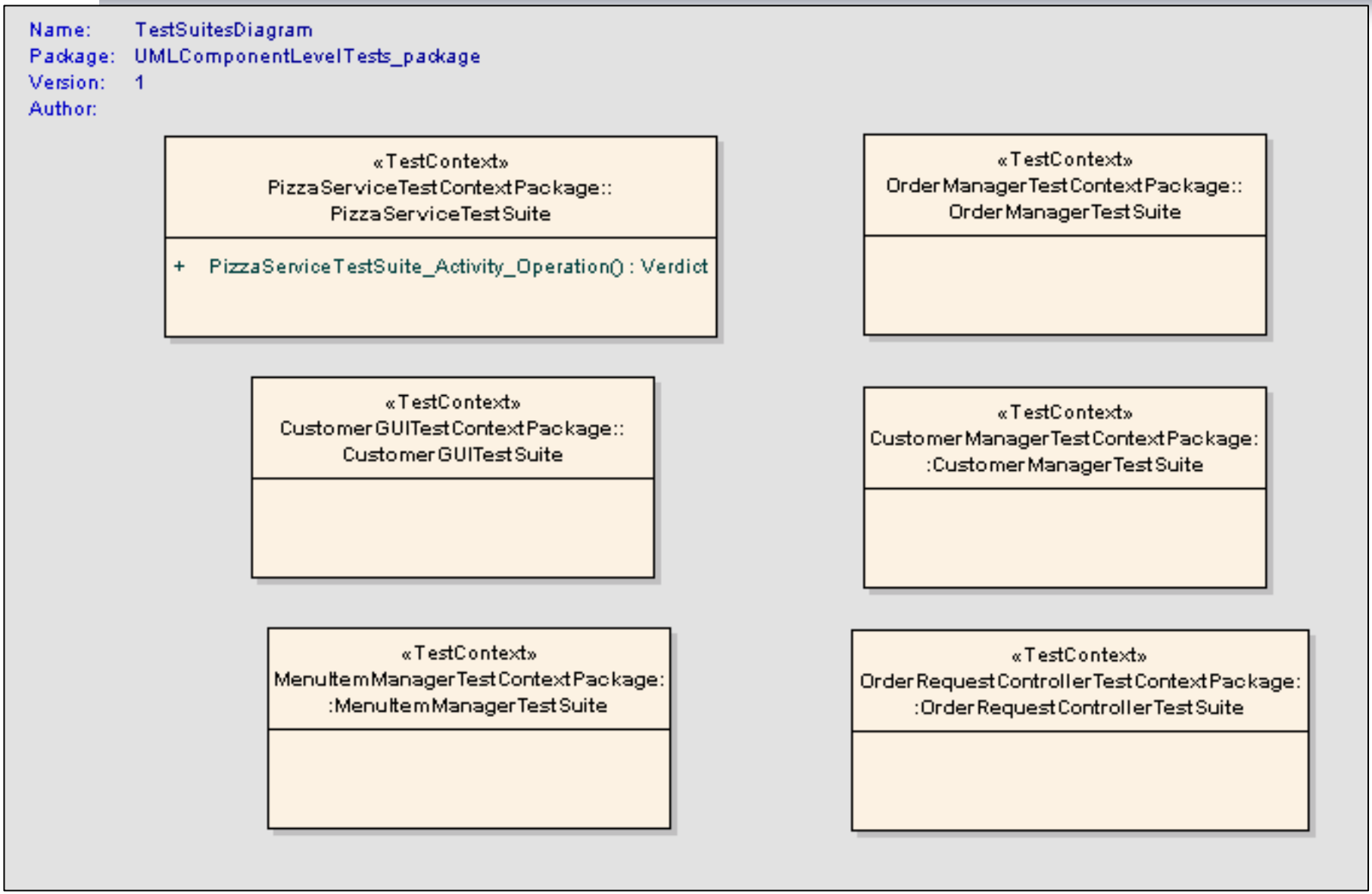
- 変換ルール５：テストコンテキストに対するテストコンフィグレーションを生成
- 変換ルール６：テストコンフィグレーションとそれに関係するクラスをテストコンテキストの対応するパッケージ内に配置
- 変換ルール７：SUTコンポーネントを生成
- 変換ルール８：テストコンポーネントを生成
- 変換ルール９：テストコンフィグレーションのクラスに*<<TestComponent>>*か*<<SUT>>*ステレオタイプを割り当てる
- 変換ルール１０：*<<TestContext>>*ステレオタイプクラスを生成
- 変換ルール１１：*“ClassNameEmulator_TestComp“*クラスを生成
- …

そのほか

- ***テストの振る舞い***の生成
- eTML から TTCN-3 への変換

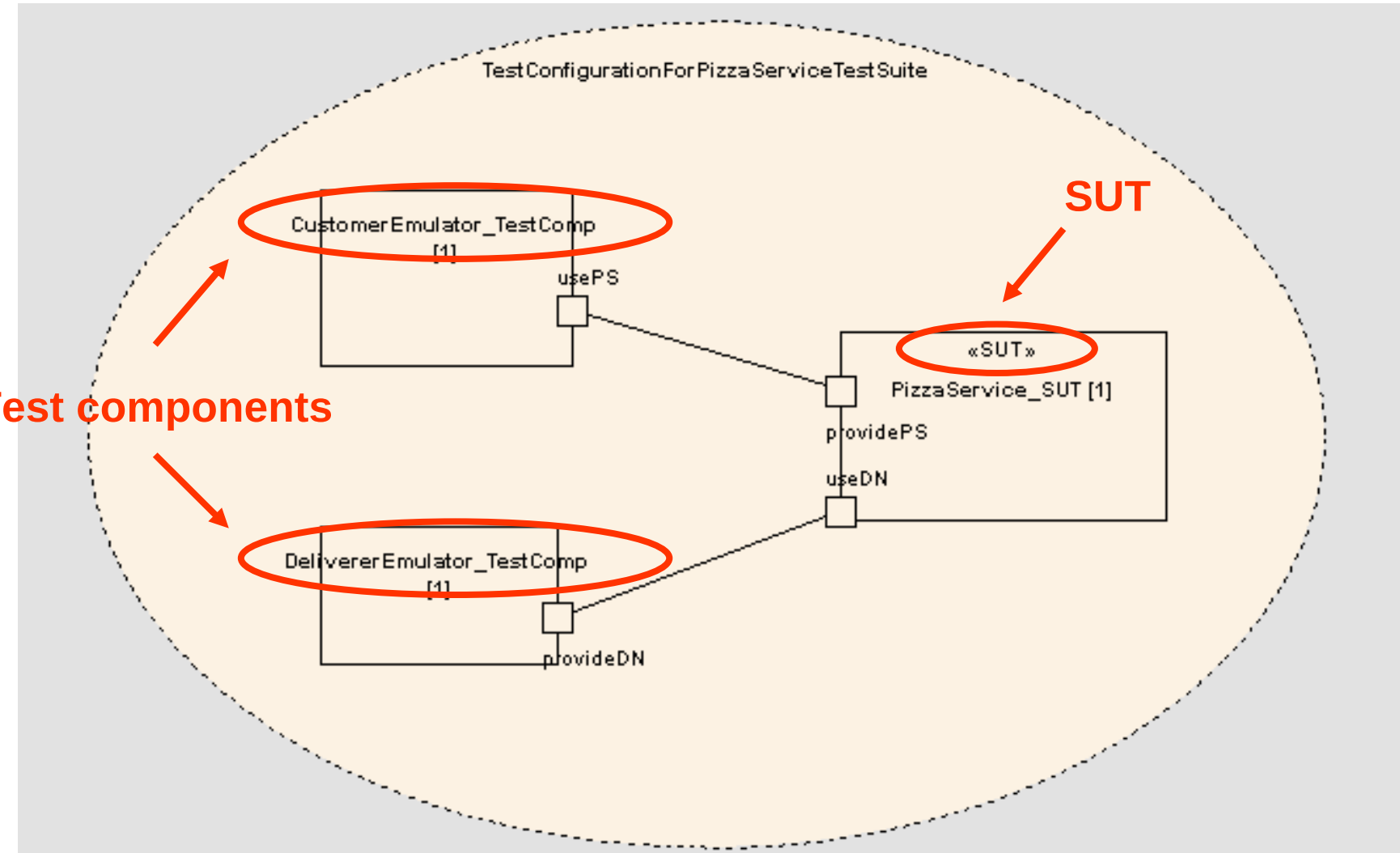
変換ルール:全体構成

[CONQUEST2006]



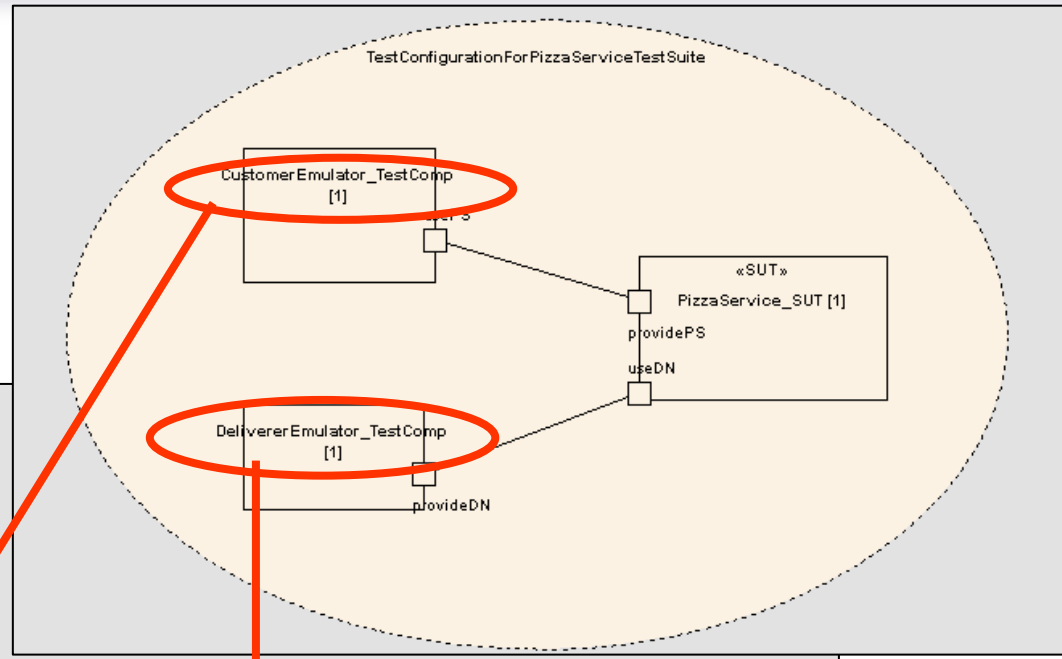
変換ルール：テストコンフィグレーション

[CONQUEST2006]

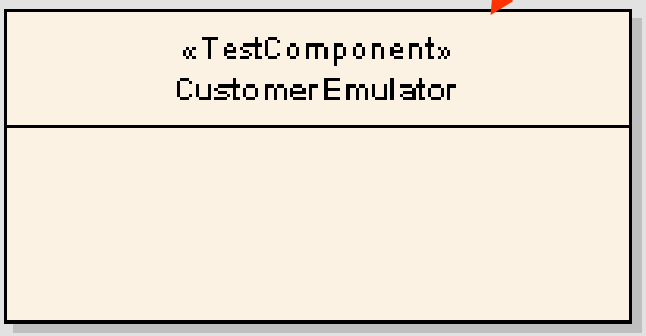


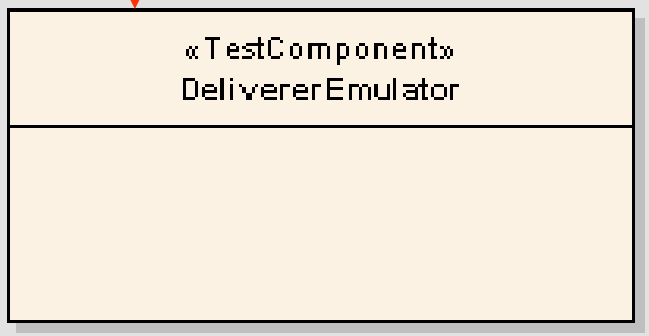
変換ルール：模擬コンポーネント

[CONQUEST2006]



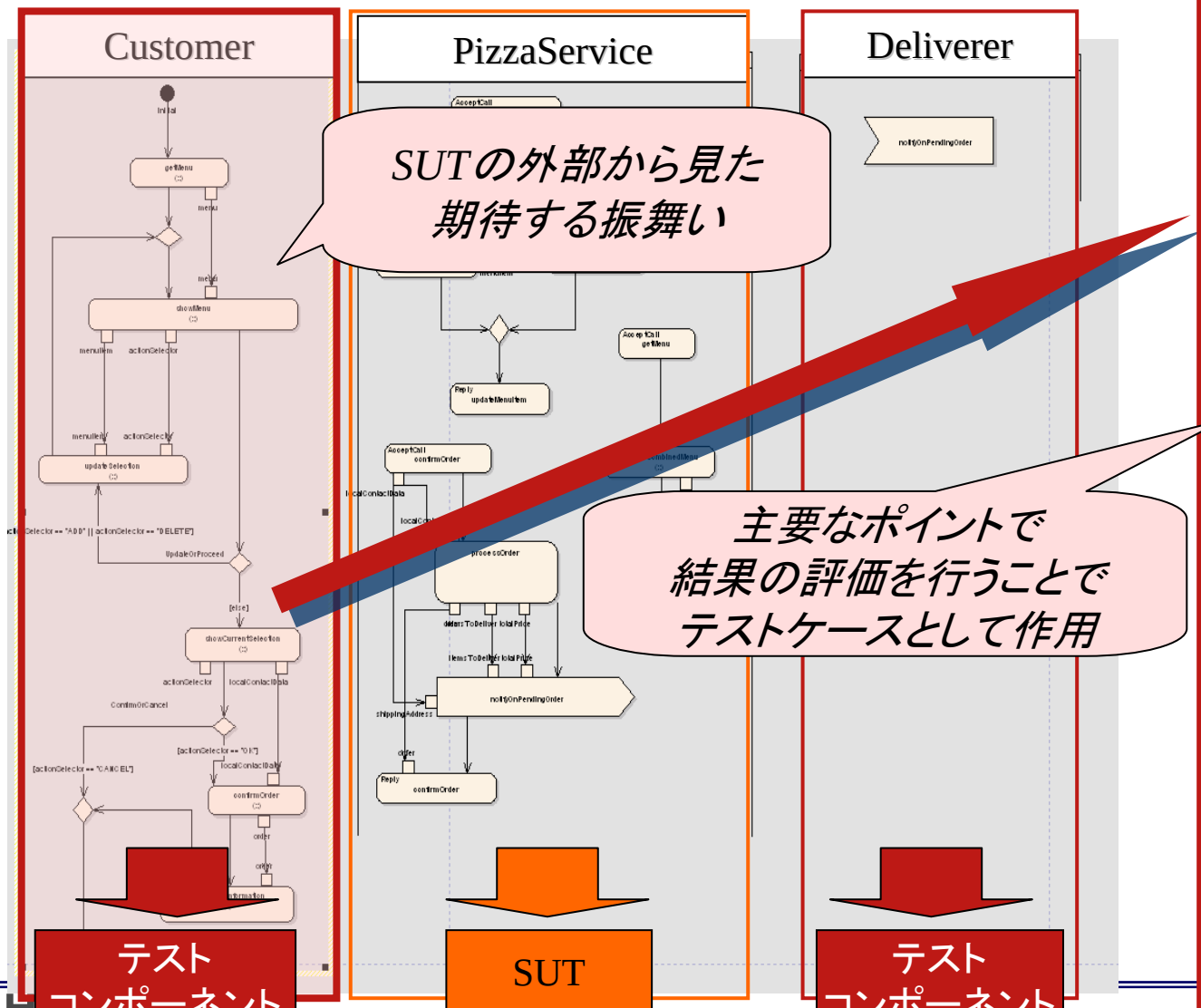
Name: CommonTypesDiagram
Package: PizzaShopSystemModel and its TestModel
Version: 1
Author:

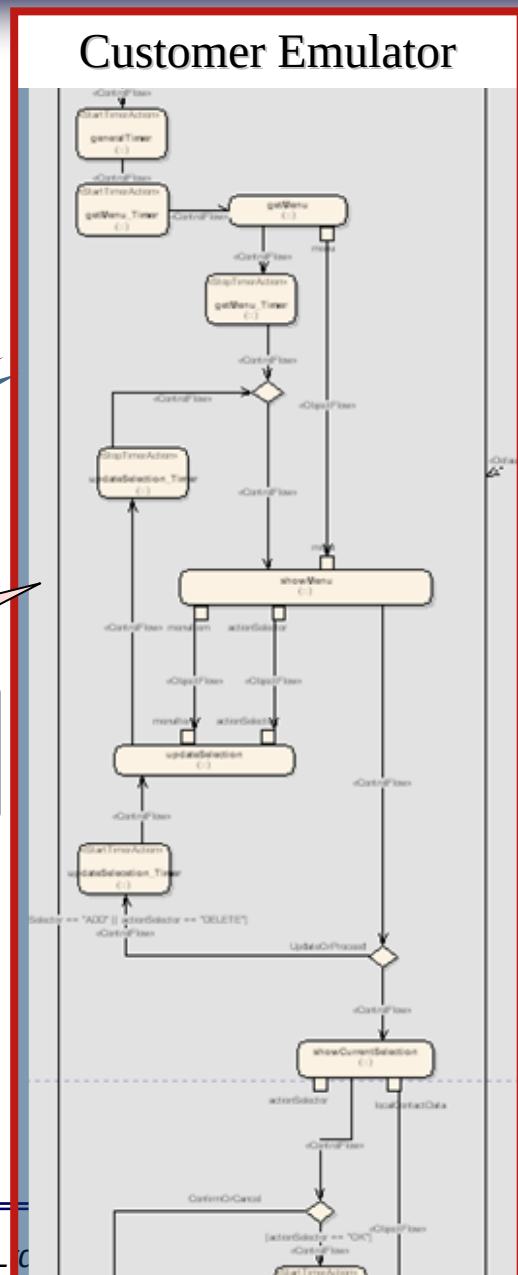




テスト動作の生成のヒント

システムモデルのアクティビティ図





研究事例のまとめ

[CONQUEST2006]

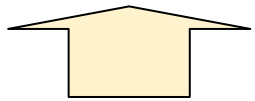
- **MDAの文脈でのモデル・ベースド・テスト**
  - OMGのMDAをテスト・モデルに関して拡張
  - システムとテストのプラットフォーム非依存モデルをUML2.0とすることで、テスト開発とシステム開発を早期に容易に統合できる
- **テストのモデリングへのモデリング言語の利用:**
  - PITとして、UML2.0とU2TPをベースとしたeTML(essential Test Modelling Language)を利用
  - PSTとして、言語が柔軟でテスト実行ツールが提供されているTTCN-3を利用
- **サンプル開発への適用**
  - ツール・チェインを実現、テストモデルと実行可能なテストコードを自動的に生成
- **より多くのUMLダイアグラムを扱い、テストの振る舞いを洗練するため、変換ルールの更なる改善が必要**

目次

1. モデルベースドテストとは何か
2. テストのモデリング
3. システムモデルからのテストの生成
4. 研究事例紹介
5. まとめ

モデルベースドテストのまとめ

- 期待してはいけないこと
  - 完璧なテスト
    - 「生成したテストを実施すれば完璧」
  - 完全な自動化
    - 「ボタン1つでテストが終了」
  - 全能的な汎用性
    - 「ツールを買えばどんなシステムもテストできる」



テストの特質:

- 完璧なテストというものはない
- どのようなテストをすべきか(テスト要求)は、人の持つ知恵
- システムごとに適切なシステムやテストのモデリング・実装は異なる

モデルベースドテストのまとめ（続き）

- それでも、モデルベースドテストに取り組む意義はある
- モデルベースドテスト
  - テストのモデリング
  - システムモデルからのテストモデルの生成
- テストをモデリングして嬉しいこと
  - テストの意味的側面を表現できる
  - テストを再利用できる
  - テストを実行できる
- テストをシステムモデルから生成して嬉しいこと
  - SUTに適合したテストシステムの構造ができる
  - SUTの詳細を開発する前にテストができる
  - SUTの基本的な機能を自動的に動作可能なテスト

おわりに

- 『モデルベースドテストっていいかも』と思われた方へ
  - 意図を汲み取っていただいて，ありがとうございます
  - でも，飛びついても役に立たないかもしれません
    - システムをモデル化するところから始めないといけませんよ
    - どんなテストをしたいか決めないと，意味のあるテストはできませんよ
    - 機械任せで済むと思ったら，痛い目に会いますよ
- 『モデルベースドテストってダメかも』と思われた方へ
  - 長時間お付き合いいただいたのに，説明がヘタでごめんなさい
  - でも，部分的にでも役に立つことがあるかもしれません
    - テストをモデリングするだけでも，テスト品質の向上に役に立ちますよ
    - テストシステムを構成するだけでも，コストは削減しますよ
    - 早期に基本的な動作テストができるだけでも，期間は短縮できますよ

おわりに

- 本質的には，
  - システムの仕様を適切に表現すること
  - 行いたいテスト内容（方針・戦略・要件）を明らかにすること
  - それらに従い，システム仕様に対して，体系的にテストを設計すること
  - 設計したテストを実行できるよう，適切に実装すること
  - テストを実行し，テスト結果を管理し，適切に処置すること
  - *設計資産と同様に，テストを貴重な資産として取り扱うこと*
- テストの質を向上するため，何をすべきか議論していきましょう

参考文献

- [CONQUEST2006] A. Hoffmann et.al., “Model Transformers for Test Generation from System Models”, CONQUEST 2006
- [TTCN-3] http://www.ttcn-3.org/
- [U2TP] “UML Testing Profile, v 1.0” , http://www.omg.org/technology/documents/formal/test_profile.htm
- [Wikipedia] Wikipedia, http://en.wikipedia.org/wiki/Model-based_testing
- [MBTHP] “Model-Based Testing Home Page”, http://www.model-based-testing.org

FOKUS

HITACHI
Inspire the Next

モデルベースドテストの技術動向と研究事例

ご清聴ありがとうございました

ソフトウェアテストシンポジウム 2007 東京

2007年1月30日