トップエスイー修了制作

形式手法を用いた 高可用性クラスタ･システムのモデル化と検証

日本ユニシスグループ　ユニアデックス株式会社　今泉 正雄　Masao.Imaizumi@uniadex.co.jp

システム設計における問題点

HAシステムの設計には，クラスタ･ソフトウェアの仕様や管理する各種リソースに対する，多くの知識･スキルを要する．またシステムが大規模化，複雑化すると，構築後の網羅的なテストも現実的でない上，それを補うべきモデル検証も人手だけでは困難である．

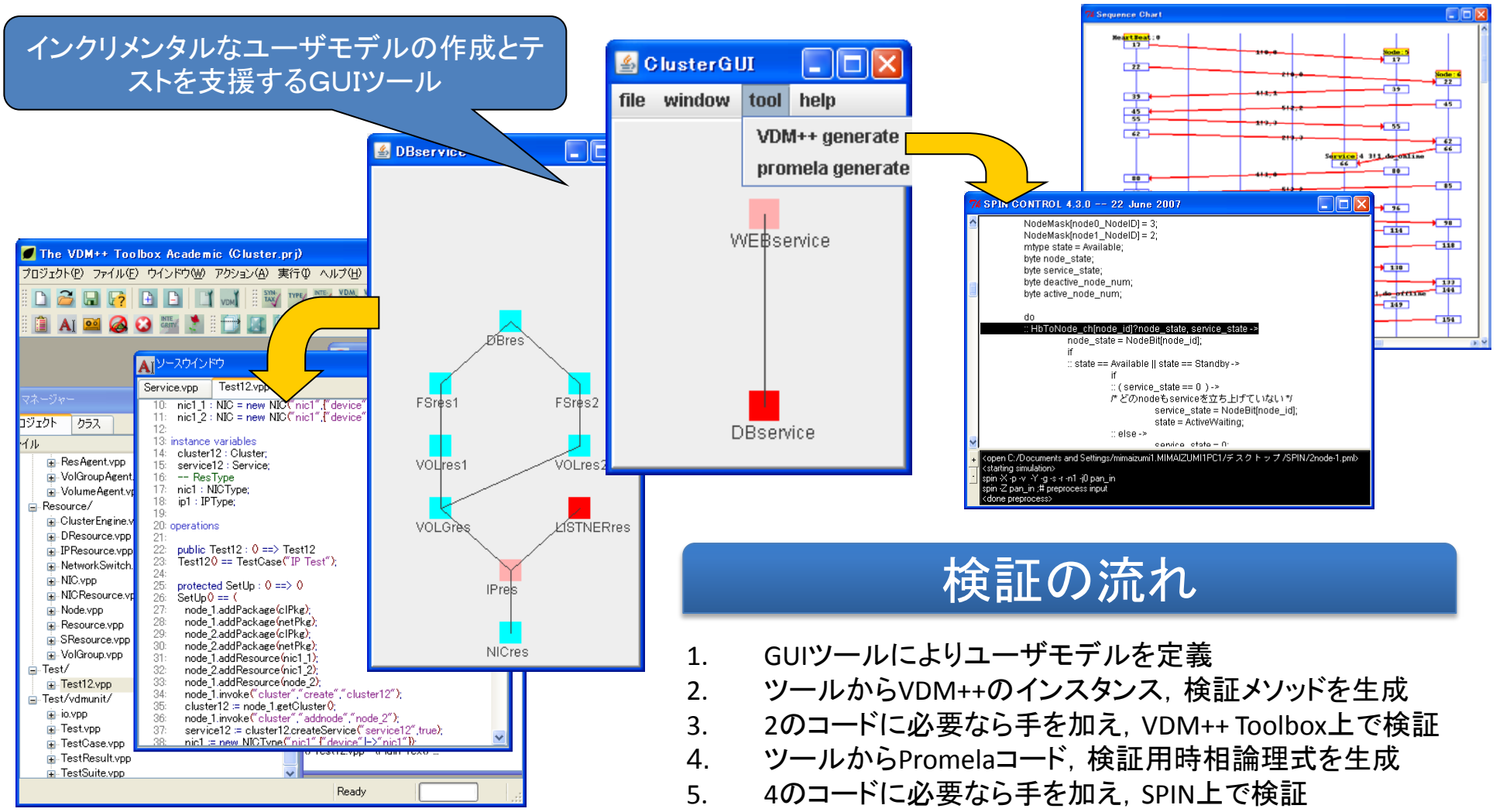
形式手法の適用による解決

クラスタ･ソフトウェアの仕様をVDM++により記述した．これにより，個々のユーザモデルが仕様において正しいことを，半自動で検査可能となった．また同じユーザモデルからPromelaモデルを生成することで，ハードウェア障害時の動作についてもシミュレート可能となった．

VDM++によるHAクラスタソフトウェアの仕様記述

- クラスタソフトウェアの仕様を，ユーザモデルの定義部，サービス起動等の動作部，ノードや各種リソース等の環境部に分割定義　⇒　ポータビリティの確保
- 定義部は，クラスタ，サービス，リソースタイプなどのクラス（ユーザモデルのメタクラス）として記述
- モデルの正しさは，不変条件やメソッドの事前条件，事後条件として記述
- ユーザモデルは，各クラスのインスタンスとして生成
- ユーザモデルを複雑化させながら，仕様の正しさを確認，修正　⇒　制約を利用したテスト駆動開発
- 出来上がった仕様に基づき，本来のユーザモデルを検証　⇒　動的な検証はSPINを援用

インクリメンタルなユーザモデルの作成とテストを支援するGUIツール



検証の流れ

1. GUIツールによりユーザモデルを定義
2. ツールからVDM++のインスタンス，検証メソッドを生成
3. 2のコードに必要なら手を加え，VDM++ Toolbox上で検証
4. ツールからPromelaコード，検証用時相論理式を生成
5. 4のコードに必要なら手を加え，SPIN上で検証

トップエスイー修了制作

ゴール指向モデルを利用したソフトウェアテストの精度向上

日本ユニシス　沖汐大志　motoji.okishio@unisys.co.jp

テストにおける問題点

ソフトウェアの品質を確保するためには，ソフトウェアが要求仕様を満たしていることを網羅的に，かつ高い精度で確認する必要がある．
網羅性はテスト技法で改善できるが，精度については，要件定義書の完成度やテスト担当者のスキルへの依存が大きい．テストの精度向上に有効な技法があるわけではない．

手法・ツールの適用による解決

要件定義書や設計書から得られる情報が不足している場合でも，ゴール指向モデルを作成することで，上位目標から要求までの関係を明らかにできる．
KAOS法のゴール指向モデルを利用して，テスト担当者がテストの精度を向上する方法を提案する．

テストの精度向上手順

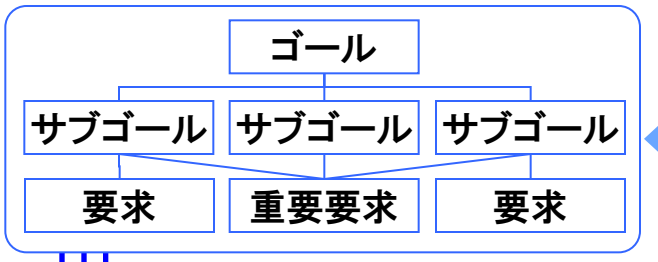
重要度の高い要求の選択，情報収集

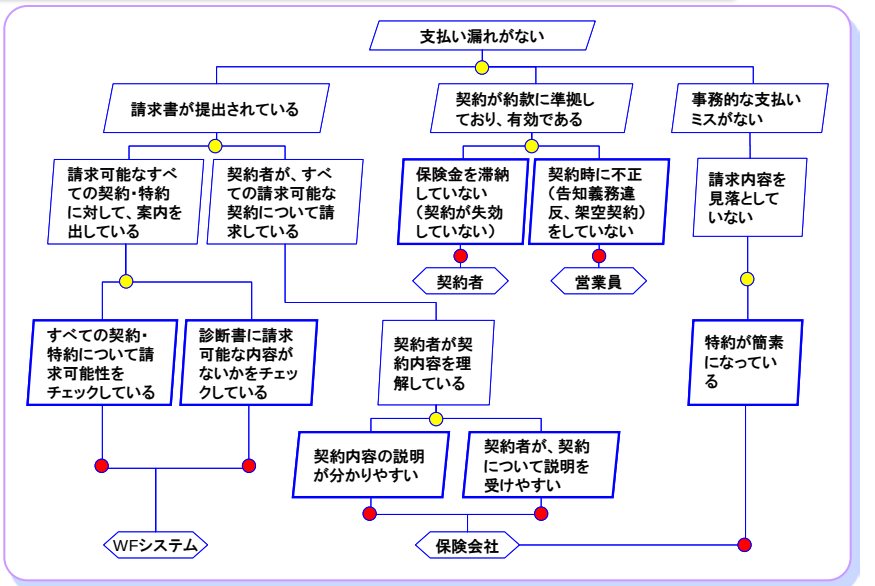
要件定義書 ＋ ヒアリング / WEB・資料

重要な要求 ⊂ 関連情報

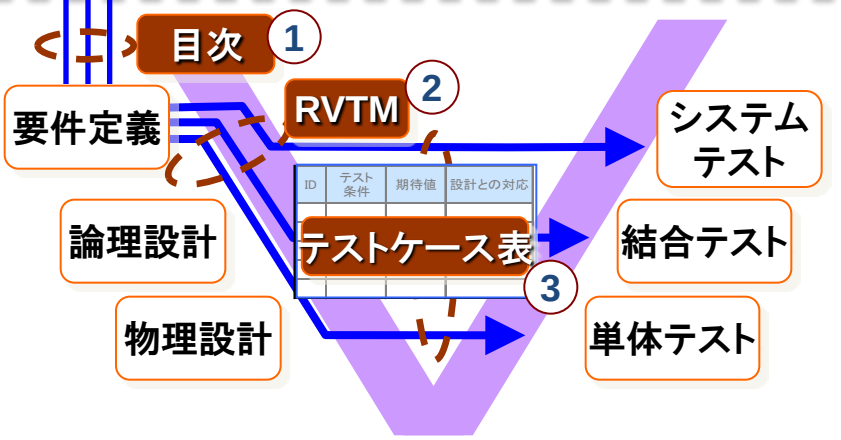
要求

ゴール指向モデルの作成





要求とテストケースの対応付け



1. 要求に対応する要件定義書の項目を，目次などを利用して関連付ける．
2. RVTM(トレーサビリティマトリクス)を利用して，要件定義書に対応する設計書の項目を関連付ける
3. テストケース表の設計書対応欄を利用して設計書に対応するテストケースを選択する

テストの精度向上

元のテストケース

・システム境界における精度の向上
・上位目標に対する精度の向上
・記述上の確認による精度向上

精度を向上したテストケース

＜条件＞
・「追加案内自動判定」の結果，保険金追加案内の「担当者判定待ち」と判定されるものが存在するとき，「担当者判定処理」の判定画面で「案内要」と判定した場合
＜期待値＞
「案内文書作成」処理に進むこと

＜条件＞
・入院給付金を支払い済みの契約者が，通院給付金の請求も可能なとき，「追加案内自動判定」で保険金追加案内の「担当者判定待ち」となり，「担当者判定処理」の判定画面で「案内要」と判定した場合
＜期待値＞
「案内文書作成」処理で，通院給付金の請求案内を作成できること

トップエスイー修了制作

UMLと実装モデル検査を組み合わせた導入コストの低い開発プロセスの提案

川上　真澄
Masumi KAWAKAMI

開発における問題点

（1）モデル検査適用コストが大
- 設計とは別に、検証用モデルを作成するコスト
- モデル検査言語の学習コスト

（2）特性の保障に失敗するケース有
- 設計時：不適切な手動抽象化による検査失敗
- 実装時：不適切な手動実装による誤り混入

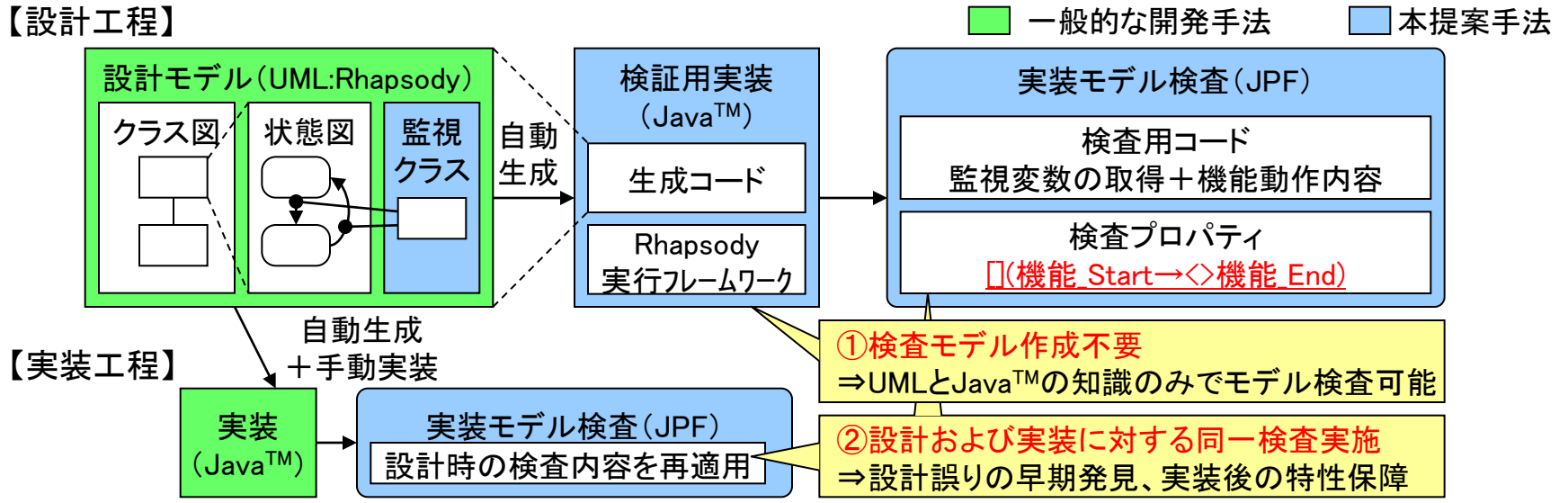
手法・ツールの適用による解決

UMLツール（Rhapsody®）と実装モデル検査（JPF）を組み合わせた開発プロセスを提案

（1）少ない適用コストで、設計および実装をモデル検査可能な開発プロセスを提案

（2）モデル検査の知識がない設計者にも利用できるよう、検査方法をパタン化

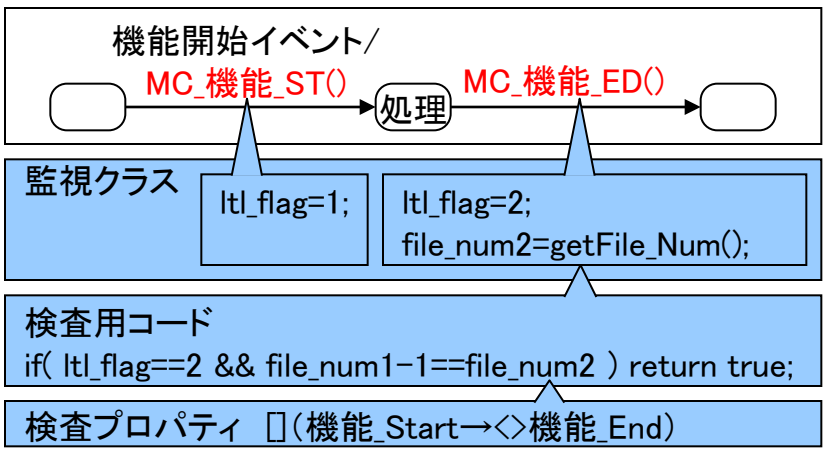
設計および実装をモデル検査可能な開発プロセス



検査方法のパタン化

【通常の機能的なテスト】
- ある条件で機能を動かして、機能が実行できることを確認
（例：削除機能を実行するとファイル数が1減る）

【本方式】
- 機能実行時の進行性＋機能確認内容をJPFで網羅検査
- 機能の開始/終了点を状態図で指定



評価

【従来手法との比較】

#	比較項目	SPIN	JPF	UML+JPF（本方式）
1	検査モデルの作成コスト	× Promela記述	○ 56step追加	○ 56step追加
2	設計時の特性保障	△ 手動抽象化	× コード必要	○ UMLを検査
3	実装時の特性保障	× 実装と乖離	○ 実装を検査	○ 実装を検査

【今後の課題】
- 検査時の状態数削減（現状2スレッドの単純な問題で確認）
- 設計モデルまたは実装から、検査したい機能に関連するモデルまたは実装をスライシング

NII 国立情報学研究所
National Institute of Informatics
トップエスイー
～サイエンスによる知的のものづくり教育プログラム～

Javaに関連するすべての商標およびロゴマークは、米国 Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です
Rhapsodyは、世界の多くの国で登録されたInternational Business Machines Corp.の商標です。

トップエスイー修了制作

UPPAALを用いたリアルタイムシステムにおける設計・運用品質の向上手法

駒形　龍太

大規模システムの設計における課題

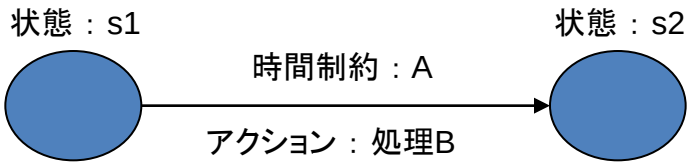
システムを構成するプロダクトは多様化し、かつ、管理するチームも分散しており、システム全体としての品質確保が困難になっている。問題点の1つとして、プロダクトのパラメータ設計の不備により、障害発生時の原因究明までの時間が延伸するという事象が発生している。

モデル検査の適用による解決

プロダクト間の連携処理に影響する時間に関するパラメータ設計（タイムアウト値等）の整合性の確保のために、リアルタイムシステムにおける時間に関する性質を検証可能なツールであるUPPAALを用いて、設計時・運用時双方に適用可能な、品質向上のための手法を提案する。

UPPAALとは

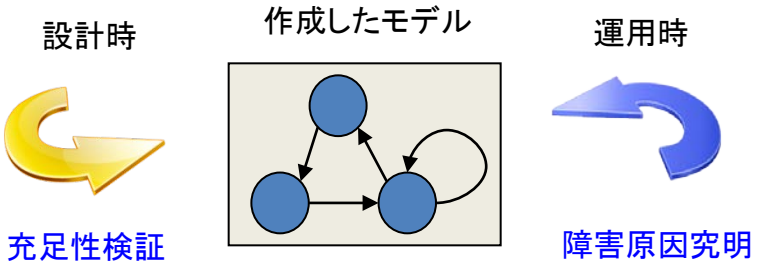
クロック値を判定ロジックに用いて動作するようなシステムを表現可能なモデル検査ツールであり、作成したモデルに対して、”動作開始から60秒以内に処理Aを終えることができるか？”といったシステムの時間的性質の検証を行うことができる。



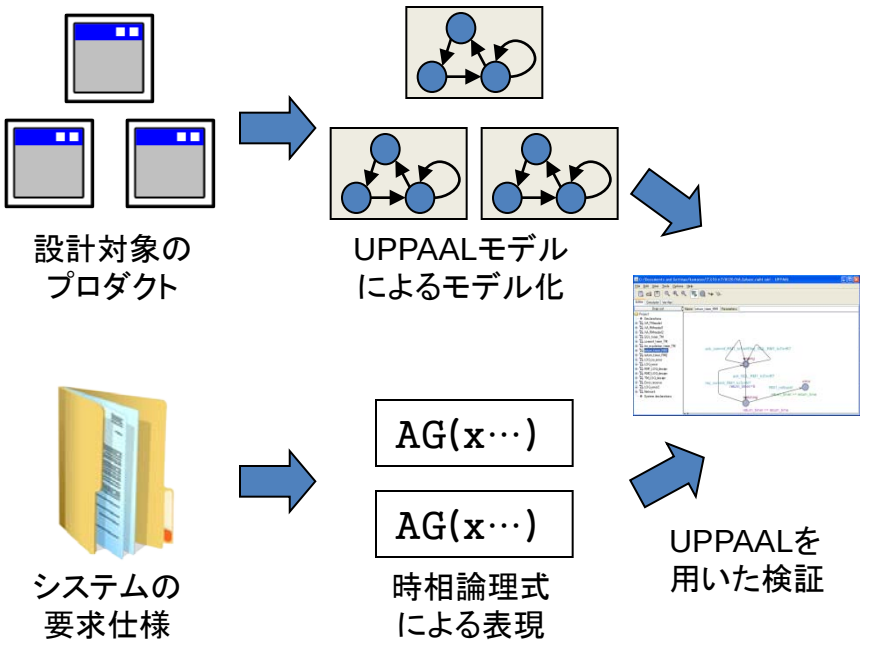
例：制約Aを満たす時、処理Bを実行し、s1からs2に遷移する。

モデル検査対象

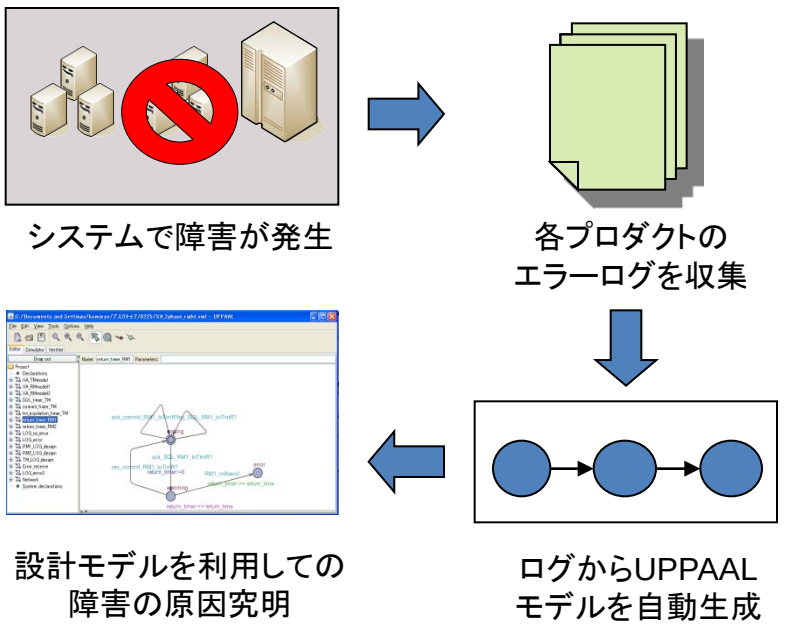
モデル検査は、ソフトウェア設計時に要求仕様の充足性の検証を行うために用いられることが一般的であるが、本手法では、更に、運用時の障害原因究明の手段として、モデル検査の適用を行う。



設計時のフロー



障害発生時のフロー



トップエスイー修了制作

複数言語対応のソースコード処理系フレームワーク ～テストカバレッジ測定への適用例～

早稲田大学 鷲崎研究室　　坂本 一憲　　kazuu@ruri.waseda.jp

開発における問題点

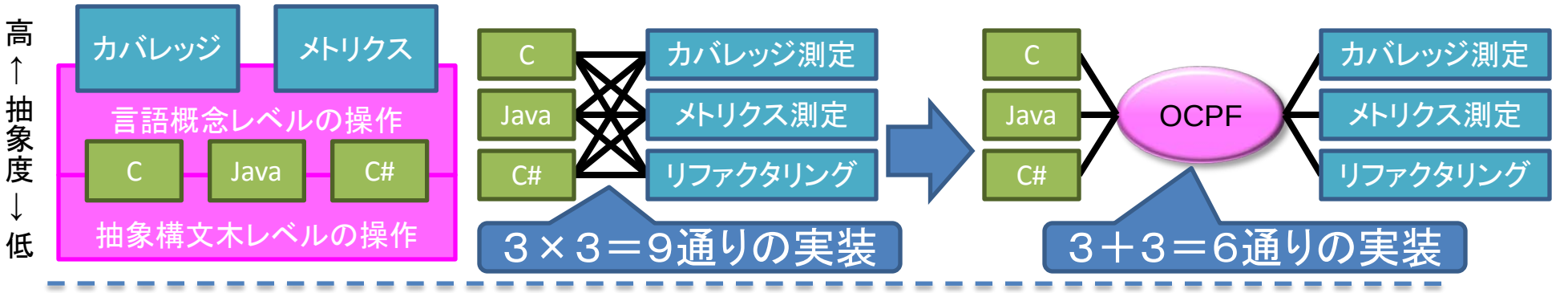
ソースコードを対象とした処理ツールが言語依存で，言語毎にツールが存在する．そのため，ツールの P1)**ノウハウ共有が困難**な点，複数言語を用いるプロジェクトでP2)**統一的に利用できない**点，新言語への対応等P3)**開発コストが高い**点，言語仕様の変更への追従等P4)**保守コストが高い**点で問題がある．

手法・ツールの提案による解決

複数プログラミング言語対応のソースコード処理フレームワーク：OpenCodeProcessorFramework (OCPF)を提案する．共通処理/言語固有/処理系固有の3部分に整理して，処理系固有と言語固有を分けることで，S1)**処理の実装を言語非依存**，S2)**言語の対応を処理非依存**にする．さらに，共通処理を提供することで，S3,4)**開発と保守コストを低減**する．

Open Code Processor Framework の概要

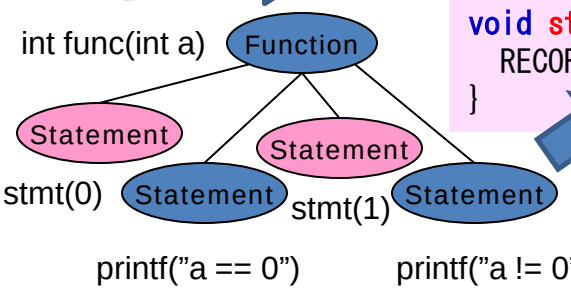
共通処理部（コールドスポット）：言語と処理系の両方に非依存．OCPFが想定する言語概念レベルの操作をインタフェースで規定（ステートメントの追加や削除等）．抽象構文木（AST）レベルの操作を提供（ノード追加や削除，置換）
言語固有部（ホットスポット）：各プログラミング言語固有の処理．各言語の言語概念レベルでの操作を実装．
処理系固有部（ホットスポット）：各ソースコード処理系固有の処理．提供される言語概念レベルの操作を利用．



カバレッジ測定への適用例

```
int func(int a) {
  if (a == 0) {
    printf("a==0");
  }
  else if (false) {
    printf("a!=0");
  }
}
```

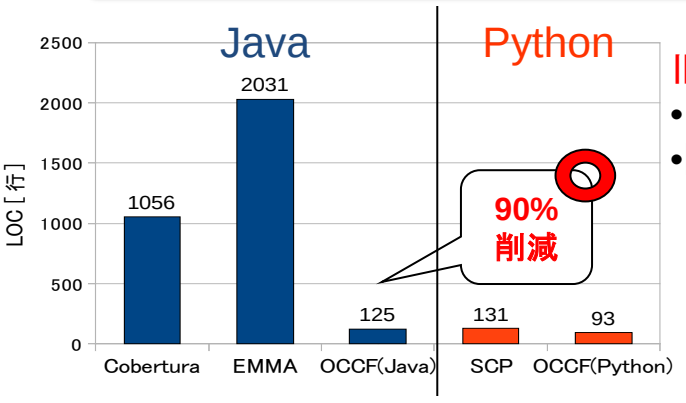
AST上で挿入




```
int func(int a) {
  if (branch(0, a == 0)) {
    stmt(0);
    printf("a == 0");
  }
  else if (branch(1,false)) {
    stmt(1);
    printf("a != 0");
  }
}
void stmt(int id) {
  RECORD_STATEMENT(id);
}
```

測定用コード
- 自動的に挿入
- カバレッジ情報出力
- 副作用なし

適用例への評価



旧バージョンで評価
- カバレッジ測定
- 開発コスト低減
  - 埋込処理のLOC
  - 90%程度削減

実装内容	達成数	平均時間
新カバレッジ追加(OCCF)	4人	13.5分
Python3への対応(OCCF)	4人	47.5分
新カバレッジ追加(SCP)	0人	4時間以上
Python3への対応(SCP)	0人	—

- 機能拡張実験
  - Python2への新カバレッジ
  - Python2からPython3へ変更

トップエスイー修了制作

モデル検査技法を用いた マリシャスコードパターンの分析

JPCERT コーディネーションセンター　椎木 孝斉　takayoshi.shiigi@jpcert.or.jp

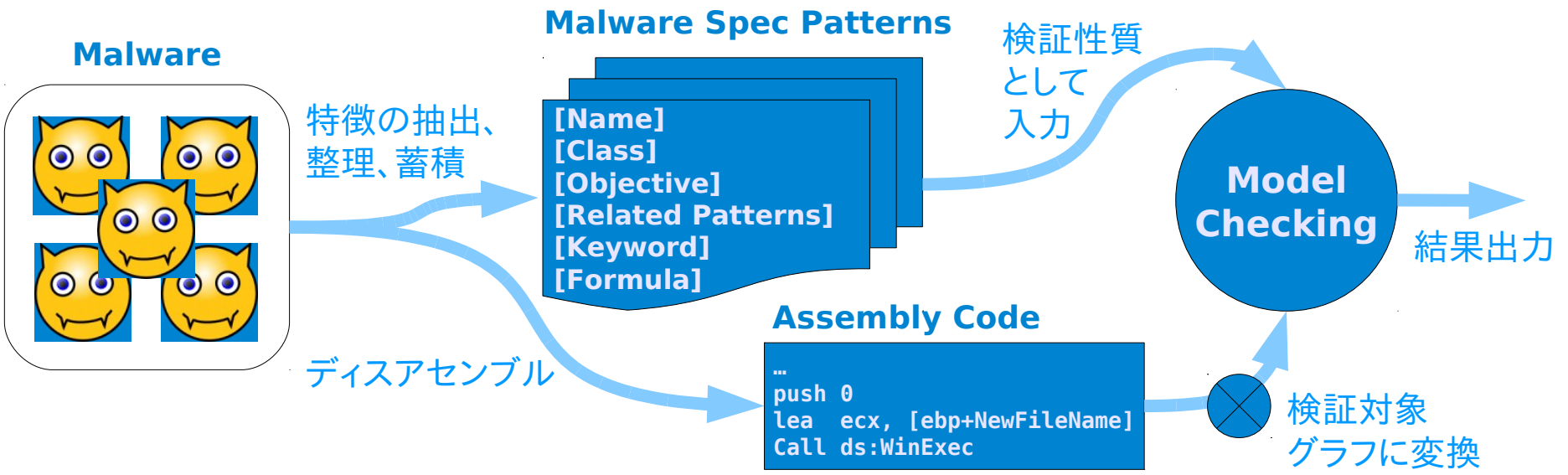
セキュリティ上の問題点

・近年、インターネットセキュリティインシデントの多くがマルウエアに関連するものとなっており、その対応が大きな課題となっている。
・マルウエア対策を行うためには、その脅威を明らかにするために分析を行う必要があるが、マルウエアの分析（特にコード分析）は難しく、必要な知識の蓄積、共有も十分ではない。

手法・ツールの適用による解決

・マルウエアのコード分析（静的分析）に必要な情報（マルウエアの挙動の特徴、分析ノウハウ等）をパターン（Malware spec pattern）として整理された形式で、作成、蓄積する。
・作成されたMalware spec patternについて、分析対象のマルウエアがその性質を満たすかどうかをモデル検査技法を用いて検証する。

提案手法の概要



Malware Spec Patternの例

[Name]
Downloader
[Class]
Common malware behavior
[Description]
Malware download program from remote site and execute it. .....
[Objective]
- To achieve initial attack smart....
[Related patterns]
Write exec
[Keyword]
URLDownloadFileA
WinExec | CreateProcess | ShellExecute
[Formula]
and @apicall(URLDownloadFileA(0, [_], [x]))
(EX EU @noassign(x) @@Exec(x))

期待される効果

・マルウエアの挙動の特徴や、分析ノウハウ等を様々なパターンとして、再利用可能な形で作成しておくことで、マルウエア分析に関する知識の蓄積、共有が可能となる。

・マルウエアのコード分析（静的分析）とモデル検査技法を組み合わせた検査を行うことで、より効率的、効果的なマルウエア分析を実施することが期待でき、対策に役立てることができる。

・今回提案する手法を分析者や技術者が使い易いツールとして実装することで、形式手法やモデル検査技法といった手法の当該分野への適用を進めることができる。

トップエスイー修了制作

セキュリティ要件定義のためのゴール指向モデリング手法

東芝ソリューション株式会社　斯波　万恵　Shiba.Masue@toshiba-sol.co.jp

開発における問題点

セキュリティの要件定義には以下のような課題があった.

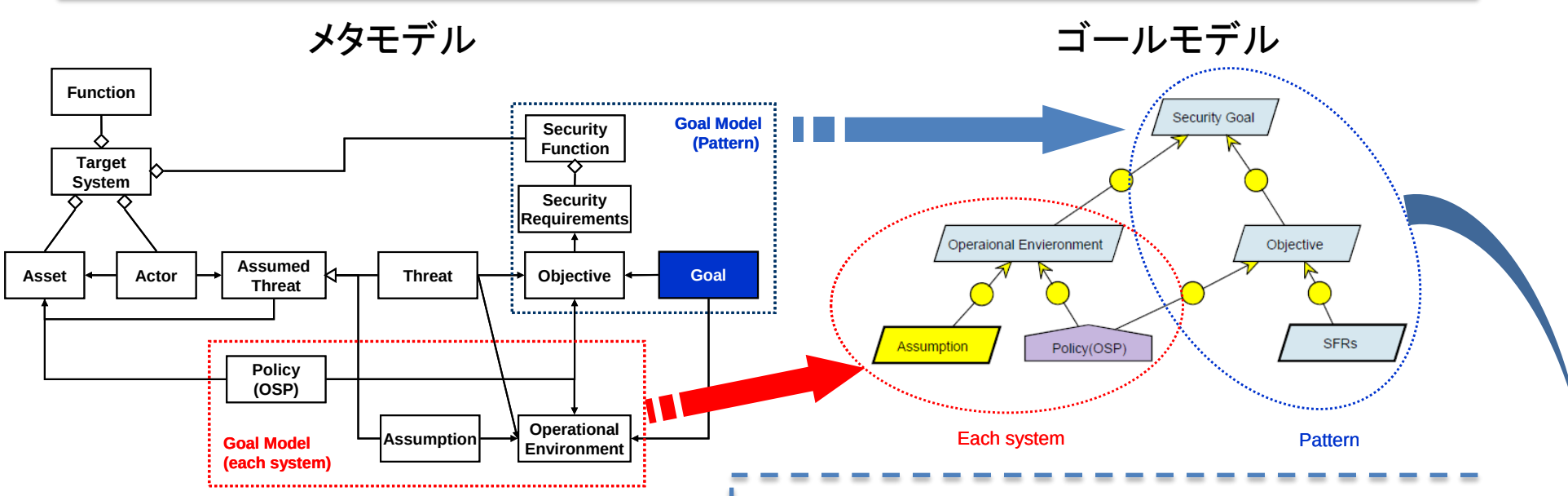
・脅威分析工程が重い
・セキュリティ機能要件の抽出が難しい
・CC Part2 コンポーネントの個々の要件化（操作）が難しい
・システムへの要求や機能からセキュリティ要件を決定する手法がない

手法・ツールの適用による解決

・脅威分析を行わない／システムへの要求による**セキュリティ要件定義モデル**が必要
・セキュリティ対策と**機能要件のパターン**が必要
・セキュリティ要件定義のための**操作モデル**が必要

セキュリティ要件定義の**メタモデルにゴールを拡張**し，KAOSのゴールモデルと操作モデルのオブジェクトにステレオタイプで新しいモデルを与えた.

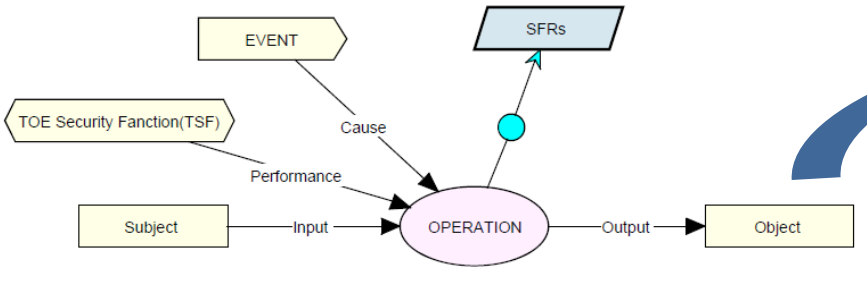
提案するゴール指向モデル



セキュリティゴール

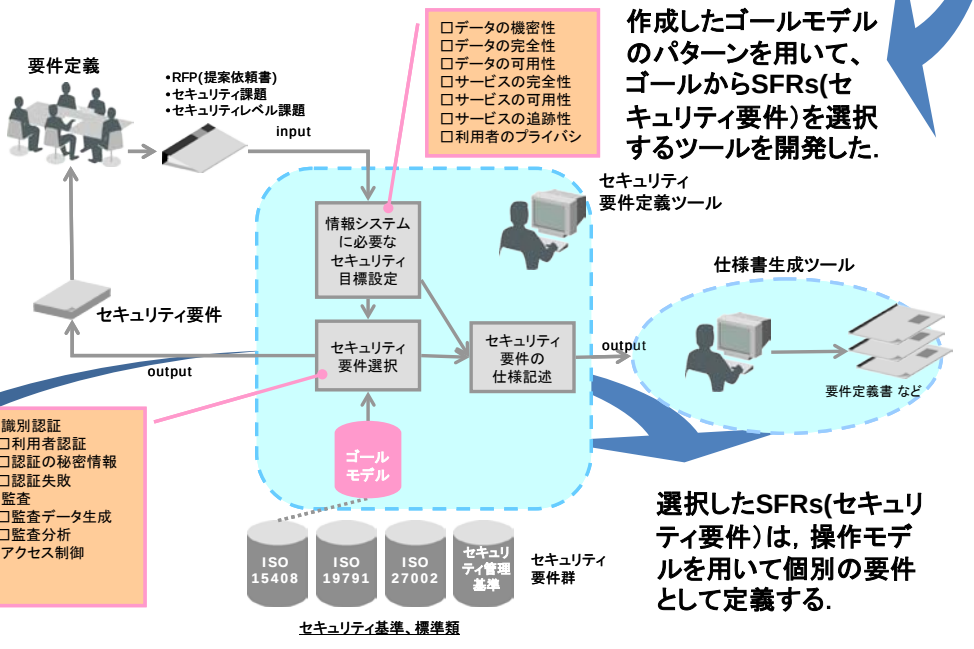
- □データの機密性
- □サービスの機密性
- □データの完全性
- □サービスの可用性
- □データの可用性
- □サービスの追跡性
- □利用者のプライバシ

操作モデル



提案モデルの適用例

セキュリティ要件定義ツール



作成したゴールモデルのパターンを用いて、ゴールからSFRs(セキュリティ要件)を選択するツールを開発した.

選択したSFRs(セキュリティ要件)は，操作モデルを用いて個別の要件として定義する.

トップエスイー修了制作

ユーザ企業におけるゴール指向要求分析法の導入とテスト効率化手法の提案

鹿島建設株式会社　清水靖則　yshimizu@kajima.com

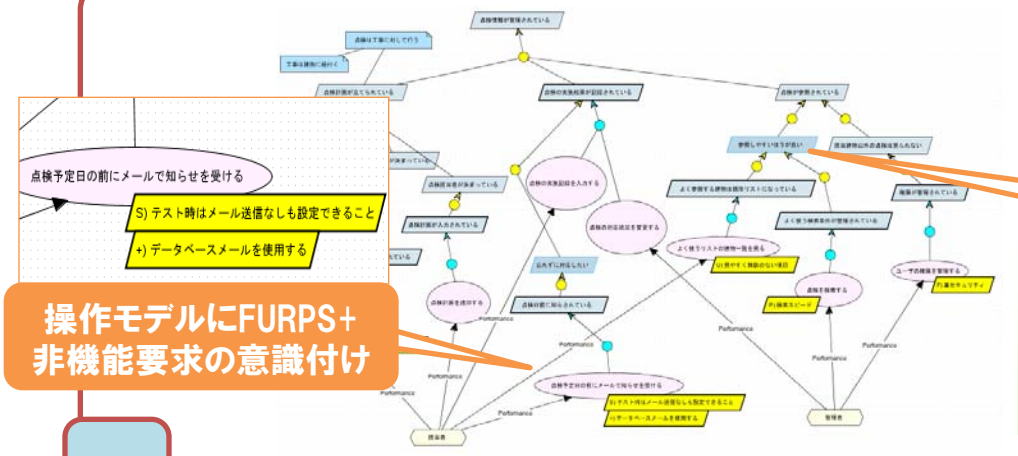
開発における問題点

曖昧で整理されていない要求分析は、開発中の手戻りによる納期遅延、リリース直後の追加開発、継続的な修正対応を招く。また、不十分な総合テストは、リリース後の不具合対応を招く。これらの工程を主体的にかつ確実に行うことはユーザ企業側の役目であり、さらに機能や運用コスト低減策の提案が求められている。

手法・ツールの提案による解決

要求分析では、ゴール指向要求分析手法KAOSをベースに、分析の手助けを行うガイドセンテンスの作成を提案。また非機能要求の早期意識付けのためにFURPS+の取り込みを行った。
総合テストでは、要求仕様書からテスト仕様書を自動作成する、テスト項目自動抽出ツールを使用した。

KAOS＆FURPS+による要求分析手順とテストの効率化



操作モデルにFURPS+
非機能要求の意識付け

分類	ガイドセンテンス	補足
ロール目線	他の抽出アクターには関係ないか？	他のアクターにもその要求はあるかもしれない。
	管理部門長は関与しないか？	回覧、周知をする機能が必要かもしれない。
	経営者はどう考えているか？	経営者の目的に合致しないゴールは変わりやすいゴールや不要なゴールかもしれない。
業務目線		あるか
機能の洗練	すいか？	また討が
	違う視点での一覧は必要ないか？	ユーザビリティの向上。
	電子ファイル保存は必要ないか？	長期保管が義務付けられている資料の電子ファイル保存化の必要性を確認する。
	知らせる方法は？	ワークフローを含む業務システムの場合、周知手段を考える。小範囲なら不要という選択ある。

ソフトゴールにガイドセンテンス
を当てて提案機能を導出

設計
→ 製造
→テスト

要求分析

KAOS
ゴールモデル

KAOS
責任モデル
操作モデル

FURPS+

ユースケース
図/記述

要求仕様書IEEE
Std.830

総合テスト

テスト計画書

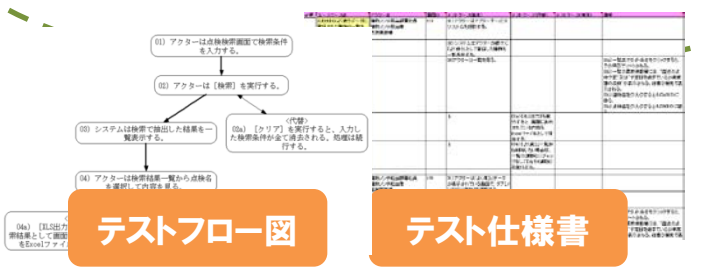
テスト仕様書
兼結果報告書

テスト実施

テスト項目
自動抽出ツール

テスト仕様書が
約5分で完成

要求仕様書の
修正漏れに
気付く（効果）



テストフロー図

テスト仕様書

【今後の展開】
・KAOS用ガイドセンテンスの充実。要求分析の経験を抽象的なガイドセンテンスとして蓄積していく。
・ゴール指向要求分析と非機能要求分析を合わせたモデリング手法の更なる探求。

トップエスイー修了制作

状態遷移表のモデル検証自動化ツールの開発

（株）東芝　ソフトウェア技術センター　　高田　沙都子　　satoko.takada@toshiba.co.jp

開発における問題点

近年、高まりをみせているソフトウェア品質の向上への施策としてモデル検証が着目されている。しかし、独自の記述言語や時相論理といった専門的知識が必要な点や人手によりモデルを作成することで検証結果を得るまでに時間を要する、といった点がモデル検査を開発現場へ導入、展開する際の足かせとなっている。

モデル検証自動化ツールの開発

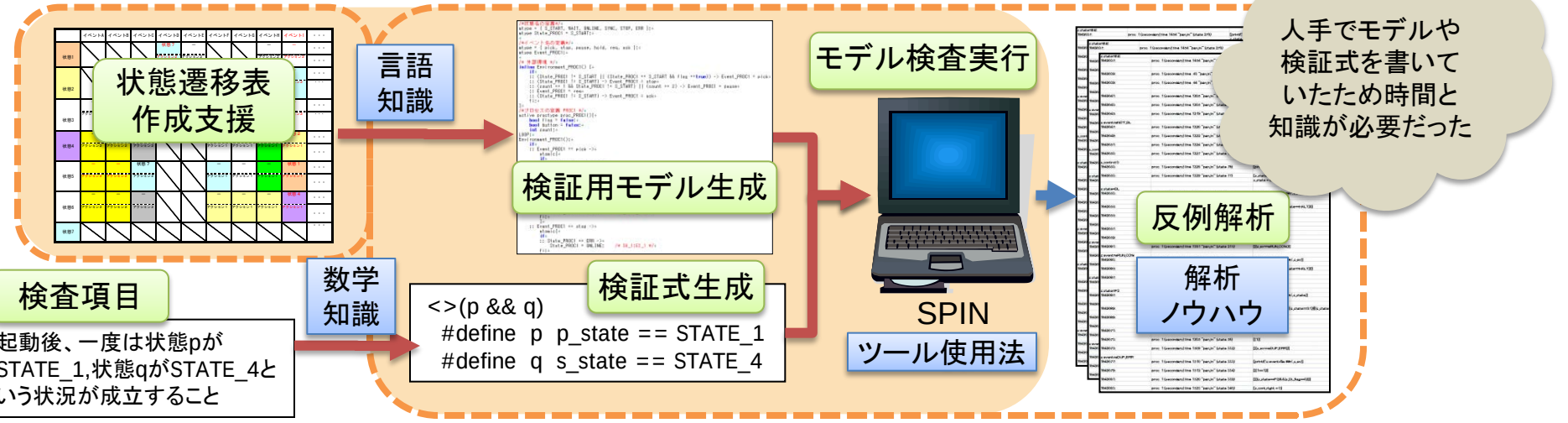
状態遷移表から検証用モデルを生成し、モデル検査を実行する作業を自動化し、更に時相論理の知識が必要とされる検査式作成を支援する機能を加えたツールを開発した。これにより、モデル検査経験のない開発者でもモデル検査適用への敷居を感じることなく取り組むことができるようになった。

モデル検証の流れと開発ツールの概要

ツールによって作業の一部を支援・自動化する

開発目標範囲　今回開発した個所

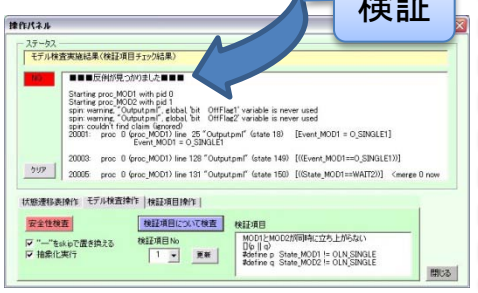
ツールによって自動化された個所



開発ツールの特徴

パネル操作で検証式や検証用モデルを生成後、検証を実行

検証



これまで…
モデルや検証式の記述方法、ツールの使い方を習得した上でモデル検査を実行

状態遷移表と変数定義を入力すればパネル上でモデル検査が可能

ツール適用の効果

実際の事例を適用

- 二重化構成による冗長性を持たせたシステム
  - 規模状態遷移表：約300セル（状態×イベント）
  - Promela行数：約500行

検査結果獲得までの時間が短縮された

開発ツール未適用	開発ツール適用
約30日	**約1日**

  - ツールの使用方法や言語習得が不要
  - モデルが自動生成されるのでケアレスミスの混入がない

専門的知識を意識せずに検証が可能

  - 簡単なGUI操作で結果を取得できる

モデル検証経験のない開発者でも取り組みが容易になった

トップエスイー修了制作

スケジュール管理Webアプリケーションへの アスペクト指向プログラミングの適用

NTTコムウェア株式会社　竹澤　亮　takezawa.ryo@nttcom.co.jp

開発における問題点

Webアプリケーション開発における以下の課題について取り組む。

・予期していない変更・機能追加への柔軟な対応
・既存部分に横断的にまたがる機能追加への対応
・提供先毎に異なる要求への対応

AOPの適用による解決

アクセス制御機能の追加開発を題材として、アスペクト指向プログラミング(AOP)を適用。
・横断的な関心毎をアスペクトとして設計・実装。既存に影響を与えずに、予期せぬ追加機能(アクセス制御)を達成。
・設定ファイルにてアスペクトの追加・取り外しが可能になり、提供先毎に最低限の機能を提供できる。

適用の概要

スケジュール管理Webアプリケーションに、アクセス制御機能をアスペクトとして追加。

Aさんの情報を更新 OK
Bさんの情報を更新 NG

本質
予定管理
ユーザ管理
グループ管理

横断的関心毎(アスペクト)
アドバイス
アクセス制御

ポイントカット(設定ファイル)
XX管理の作成メソッド呼び出し時に挿入

Weave

ポイントカットによって絞り込まれた、挿入能な位置(ジョインポイント)にアドバイスを織り込む

予定管理
ユーザ管理
グループ管理
アクセス制御

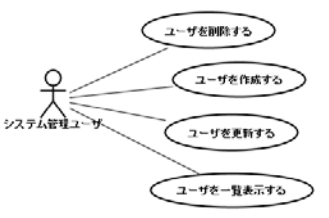
適用の評価と課題

〇予期せぬ横断的な機能追加に、母体への修正なしで対応できた。
〇取り外し・交換可能な状態に分離できた。
〇本質・アスペクトの個々について見通しが良くなった

修正数	クラス	メソッド	ライン(LOC)	設定ファイル
適用時	0	0	0	大 (約400行)
未適用	5	30	50〜100	小 (数十行)

×全体を結合した状態で、矛盾無く動作するか、把握しづらい。
×設計段階でのレビューや、試験項目作成が困難に。
△設定ファイルの肥大化。

試作した適用支援ツール

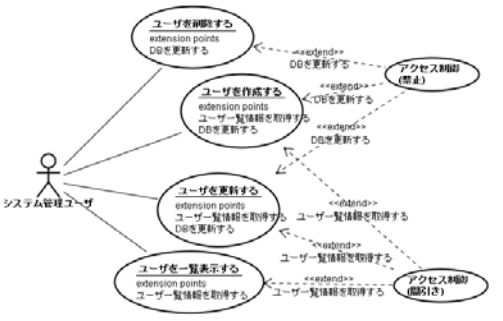


結合ルール
(設計ドキュメントをCSV出力したもの)
ユーザを作成する, DBを更新する,
アクセス制御(禁止)

アクセス制御(間引き)
アクセス制御(禁止)

Weave

ユースケースをコードと同様に織り込む。



トップエスイー修了制作

インクリメンタル型 ソフトウェア開発の品質予測

富士通株式会社　竹田和正　takeda.kazumasa@jp.fujitsu.com

インクリメンタル開発の品質予測

・ウォーターフォール型プロセスでは
品質予測モデル*が確立している
*ソフトウェア信頼度成長モデルやレイリーモデルなど

・インクリメンタル型開発プロセスにおいて
予測モデルをそのまま適用することができない

欠陥除去率の予測モデルを構築

・欠陥除去率の予測モデルを構築することで
開発途中の残存欠陥数を可視化する

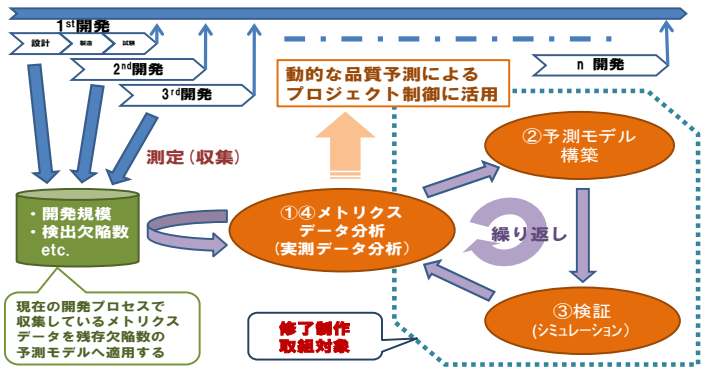
・インクリメンタル開発プロセスに適用可能な
品質予測モデルを構築することで適切な
プロジェクト制御を可能とする

モデル化のアプローチ

右図のとおりの手順を繰り返す
① メトリクスデータ分析（実測データ分析）
② 予測モデル構築
③ 検証（シミュレーション）

●リリース毎の**欠陥除去率（DRE* : Defect Removal Efficiency ）**に着目
* DDR = リリース前欠陥数÷（リリース前欠陥数＋リリース後の欠陥数）
● **欠陥作り込み密度（DID * : Defect Injection Density）**
* DID = 欠陥数÷開発規模
●各リリースにおける追加の**開発規模**と**摘出した欠陥数**（残存欠陥数）との関係に着目

最終リリース（出荷）における残存欠陥数（＝顧客評価で検出される欠陥）を予測する

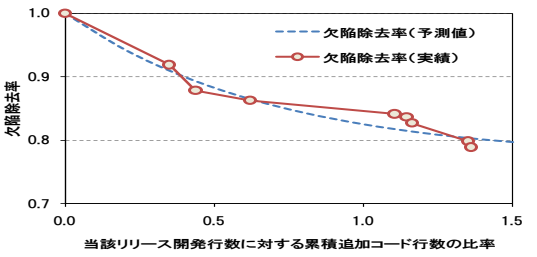


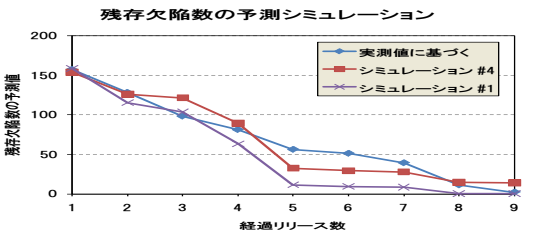
モデル構築とシミュレーション結果

①シミュレーション作成ツール*により残存欠陥数を予測するモデルを構築
リリースid，開発規模，リリース前後の摘出欠陥数，欠陥除去率(DRE)，
作り込み欠陥密度(DID),から、残存欠陥数の予測値分布を生成した.
*GeNie（ネットワークモデル化ツール）使用

②実測値データから欠陥除去率を予測するモデルを構築
実測値（欠陥除去率）を外接する指数型モデルとして定式化した.
⇒ 一般の欠陥発見モデルにしたがった

③累積追加開発規模比率を使用して，欠陥除去率を予測するモデルを構築
②をリリースidではなく，累積追加した開発規模比率を説明変数としてモデル化した.





・リリース毎の開発規模と欠陥除去数から**残存欠陥数を予測することが可能**である.
・プロセスパラメータ(DRE,DID)を目標値とした**品質制御が可能**である.

評価結果

●インクリメンタル開発プロセスにおける**品質予測のモデル化が可能**であることを検証できた.
●リリース途中の品質傾向が可視化可能となり，**プロジェクト制御に耐えうる（メトリクスとして有効）**レベルであることが検証できた.
●出荷後の品質評価が定まった（欠陥数の検出が収束した）時に，シミュレーション結果の最終的な評価を行うことでより精度の高いモデル（静的モデル）の構築が可能となった.

今後の展開

●リリースid，開発規模，欠陥除去率から残存欠陥数を予測した
⇒ さまざまな**プロセス要因*を加味**することで，プロジェクト の実状にあった予測モデルへ発展させることが可能.
*プロセス要因:プログラム階層,経験値,工数,仕様変更量etc.
● **ベイジアンネットワークモデル**への本格適用
⇒ メトリクスデータから確率分布を推定（学習）する試みを行うことでモデル化できないか.

トップエスイー修了制作

Promela における割り込み制御の半自動モデル化

フェリカネットワークス株式会社　只野 賢二　Kenji.Tadano@FeliCaNetworks.co.jp

開発における課題

ファームウェアの品質向上手法として，モデル検査を導入するにあたり，C 言語のソースコードをモデル記述言語である Promela に変換（モデル化）する必要がある．
モデル化の過程で，割り込み制御情報をモデル化する必要があり，割り込みの数が増加するにつれて，モデル実装者の負担や実装中のミス混入が懸念される．

本研究の提案

モデル実装者の負担軽減，実装中のミス混入軽減をするために，以下を機械的にモデル化する手法を提案する．

- 多重割り込み
- 割り込みマスク

※ 上記 2 つが割り込み制御情報のモデル化を複雑にする原因である．

研究の概略

従来のモデル化

割り込み制御情報（HWマニュアル等）

ソースコード（ C言語 ）→ 割り込み制御有りモデル（ Promela ）

手動変換

（割り込み制御情報を考慮）

→ 割り込みの数が増加するにつれて，変換が複雑化

提案手法によるモデル化

割り込み制御情報（csv形式）

ソースコード（ C言語 ）→ 割り込み制御無しモデル（ Promela ）→ 割り込み制御有りモデル（ Promela ）

手動変換

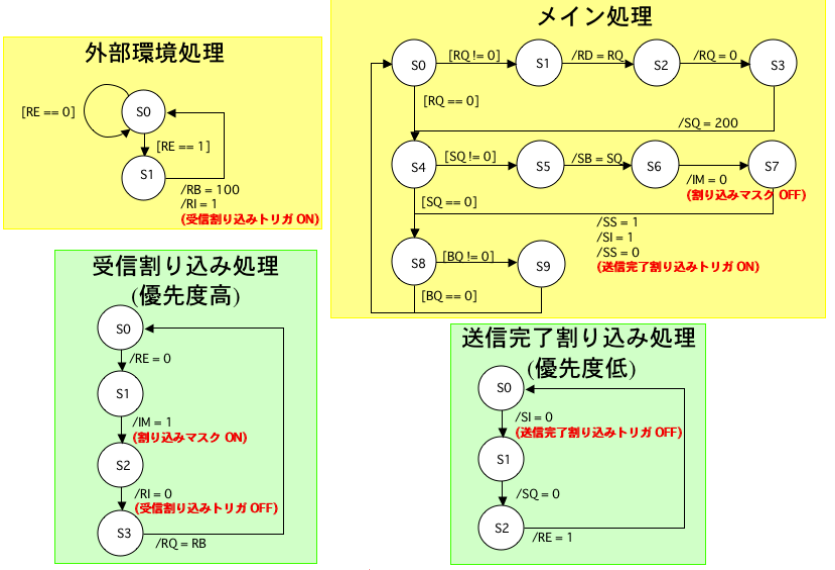
自動変換 本研究のスコープ

（割り込み制御情報を考慮せず）

→ 人手で割り込み制御情報を考慮せず，変換が可能

実システムへの適応

モバイル FeliCa IC チップを本手法によりモデル化



→ 検証項目「いつか必ず RE == 1 となること」を確認

まとめ

本研究で行ったこと

- 多重割り込み，割り込みマスクを考慮した割り込み制御情報を半自動でモデル化する手法を提案
- 提案した手法を自動にて行う，変換ツールを作成し，ツールの妥当性を確認
- 実システムへの適応による，有効性の確認

今後の展望

- 規模のより大きいモデルへの適応
- 複数割り込みが同時に動作する場合の考慮

トップエスイー修了制作

Webシステムにおけるデータ項目の抽出とチェックの仕組み検討

塚本　純一　　tukamoto@sysmic.jp

開発における問題点

Webシステム改修による品質の低下が発生することがあり、原因の一つとして同時に修正しなければならない部分の修正が漏れてしまう。
Webシステムには多くのデータ項目があり、データ項目の状態をチェックできればよいが、扱うデータ項目は多いことと、複数のデータ項目を持つオブジェクトが多く存在しているため、チェックするのが大変である。

手法・ツールの提案による解決

複数のデータ項目を持つオブジェクトを抽出し、チェックするツールを提案。
このツールを、データ項目をオブジェクトから使用する際に、アスペクトで挟み込む。

データ項目の抽出とチェックの仕組み

データ項目の抽出

JSPからデータ取得の：

```
Object obj = request.getAttribute("person");
```

このオブジェクトが保持しているデータの抽出を行う。

mapとして抽出したデータのイメージ

Key(項目名)	value(値)
name	name1
age	27
address	aaaa@xxx.jp

データ項目が持つ項目名、値をマッピングするオブジェクト(map)として抽出する。
このmap化を一度行うことによってチェックメソッドは専用のメソッドを用意せず統一してチェックすることができる。

データ項目のチェック

mapとして抽出したデータのイメージ

Key(項目名)	value(値)
name	name1
age	27
address	aaaa@xxx.jp

⇔

Xmlに定義されたデータ項目情報のイメージ

Item name (項目名)	data type (データ型)	Digit (桁)
name	string	40
age	number	3
address	string	80

チェックは、日付、数値、文字列等の各項目でチェックを行いたい内容のチェックメソッドを用意し、mapとして抽出したデータのデータ項目とXmlファイルで定義された内容を照らし合わせることでチェックを行う。

アスペクトの挟み込み

コードの例

```
public void execute(ActionMapping  map,
                    HttpServletRequest  request) {
   …
   if(request.getAttribute("update.flag") != null) {
      Object obj = request.getAttribute("person");
      …
   }
   …
}
```

オブジェクトが取り出される時にアスペクトによる処理の挟み込みを行う。

データ項目の抽出とチェック実行

データ項目の抽出とチェックをシステムの処理とは別に切り離して実行される。

まとめ

・達成されること

Webシステムでは、1つのオブジェクトに複数のデータ項目を入れることが多いので、そのデータ項目をチェックする考え方。

・達成できていないこと

現時点では「方針・考え方」であって、「ライブラリ・フレームワーク」までは作りこんでいない。
また、チェックをアスペクトで挟み込むタイミングは別途検討が必要。

トップエスイー修了制作

組込みシステムコア資産の作成手法の提案

所属 三菱電機マイコン機器ソフトウエア株式会社
名前 徳弘　雄亮　メールアドレスtokuhiro@mms.co.jp

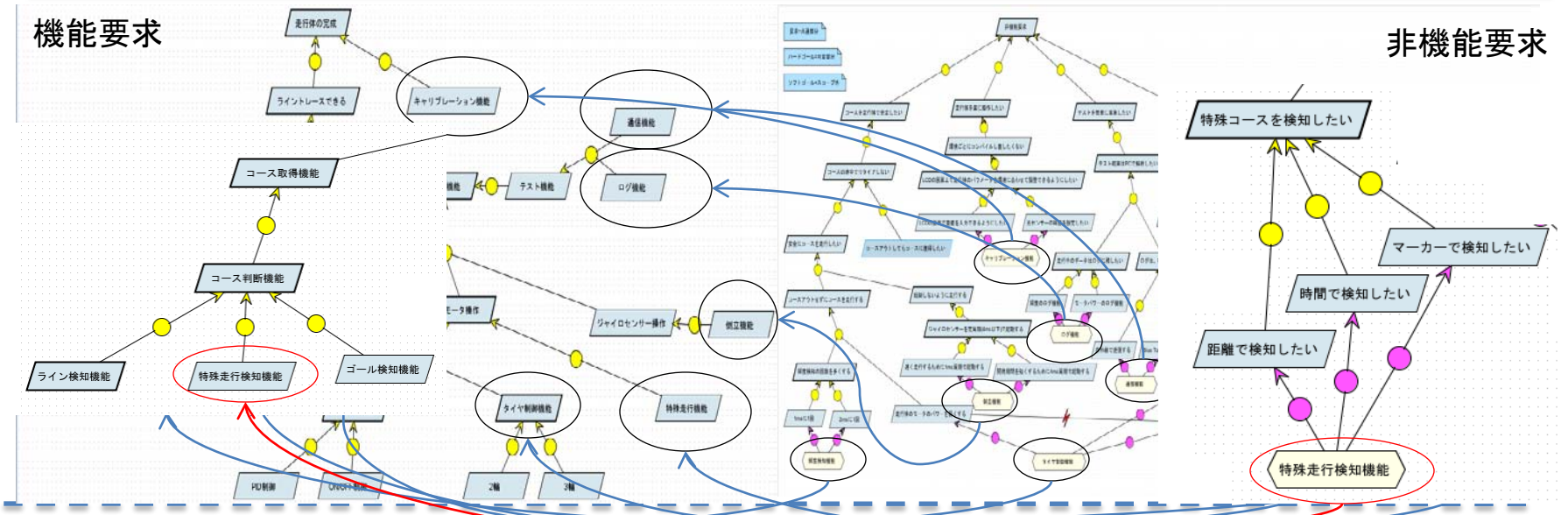
開発における問題点

プロダクトライン開発において、非機能要求の変更によりアーキテクチャの変更が発生した。アーキテクチャの変更は、コア資産の使用を制限され、プロジェクトの失敗を発生させる。非機能要求の変更に対応できる、アーキテクチャの開発が必要である。

手法・ツールの適用による解決

変更に対応するためには、詳細な部分で非機能要求と機能要求が対応しておく必要がある。詳細な部分まで要求を分析するために、ゴール指向要求分析であるKAOSを使用した。KAOSで分析した要求からフィーチャーモデルを作成し、コア資産を使いやすくするために、コンポーネントベース開発のKobrAを使用し、コンポーネント化した。

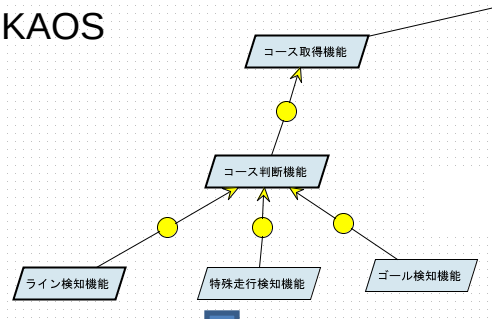
ゴール指向要求分析(KAOS)の成果物



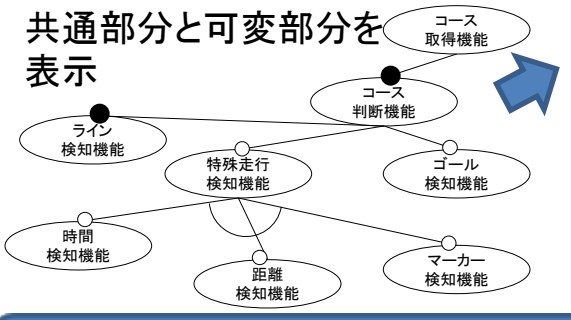
機能要求と非機能要求の対応付け

要求分析から設計までの対応

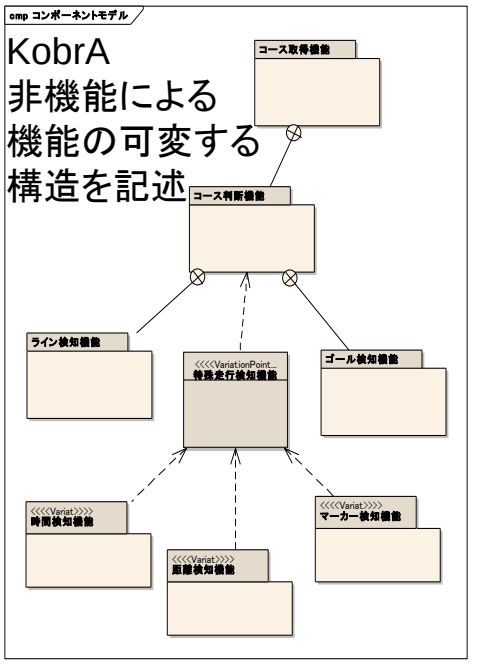
KAOS



フィーチャーモデル
共通部分と可変部分を表示



KobrA
非機能による機能の可変する構造を記述



非機能要求と機能要求の対応付けの手法
KAOSによる要求分析
　ゴール指向要求分析
　AND/ORによる階層構造
　共通部分と可変部分の分析が可能
　具体的なゴールをトレースできる分析が可能

KAOSからの拡張点
　非機能要求の具体的な実現手法の分析
　非機能要求を実現する機能要求の追加

非機能要求の対応前
　非機能要求の変更が、複数の機能要求に影響
　変更がアーキテクチャの構造に影響を与える
非機能要求の対応後
　非機能要求の変更が与える影響が明確
　機能コンポーネントの変更で対応が可能