

# 開放型分散環境でのソフトウェア部品検証システムの研究開発

## Research and Development on Software Component Verification System in Open Distributed Environment

二木 厚吉<sup>1)</sup>      森 彰<sup>2)</sup>      萩谷 昌己<sup>3)</sup>      澤田 寿実<sup>4)</sup>  
瀬尾 明志<sup>5)</sup>      加藤 賢次郎<sup>5)</sup>      石黒 正揮<sup>6)</sup>  
Kokichi FUTATSUGI   Akira MORI   Masami HAGIYA   Toshimi SAWADA  
Akishi SEO                      Kenjiro KATO                      Masaki ISHIGURO

- 1) 北陸先端科学技術大学院大学 情報科学研究科 (〒923-12 石川県能美郡辰口町旭台 15 E-mail: kokichi@jaist.ac.jp)
- 2) 独立行政法人 産業技術総合研究所 サイバーアシスト研究センター (〒135-0064 江東区青海 2-41-6 E-mail: amori@carc.aist.go.jp)
- 3) 東京大学大学院情報理工学系研究科 (〒113-0033 文京区本郷 7-3-1 E-Mail: hagiya@is.s.u-tokyo.ac.jp)
- 4) 株式会社 SRA 先端技術研究所 (〒160-0004 新宿区四谷 3-12 丸正ビル 5F E-Mail: sawada@sra.co.jp)
- 5) 日本ユニシス株式会社 (〒135-8560 江東区豊洲 1-1-1 E-Mail: Akishi.Seo@unisys.co.jp)
- 6) 株式会社 三菱総合研究所 (〒100-8141 千代田区大手町 2-3-6 E-Mail: masa@mri.co.jp)

**ABSTRACT.** We developed an integrated system of search engine for software component and software verification system based on behavioral specifications. Our system enables software developers to search software components on the Internet based on their interface or behavior by signature matching or refinement verification. Behavioral specifications based on hidden algebra is used to allow search by functionalities rather than syntactic features. It also provides model checking functionality for checking the safety properties of the software under development before its operation. As the system is implemented by using server-side web-based technologies and standard Internet protocols such as http, TCP/IP sockets, we can access to all the functionalities through ordinary web browsers.

## 1 背景

近年のインターネット技術の急速な発展・普及を背景に、ソフトウェア利用の形態が大きく変化しつつある。ユーザーの手にあるコンピュータに必要なソフトウェアをインストールし、当該マシン上で全ての計算・操作を閉じた形で行うのではなく、遠隔サーバーマシン上に提供されたアプリケーションをデータも含めてネットワーク経由で利用したり、異なるサーバー上に分散オブジェクトとして配置されたソフトウェア部品を複数組み合わせる利用したりといったことが日常的になりつつある。

一方、近い将来、多数の商用あるいは非営利のアプリケーションやソフトウェア部品がネットワーク上で利用できるものとなるものと期待されるが、利用者が自らの利用形態に合う望みのソフトウェア部品を捜し出すための検索・検証機能が提供されなければ、提供されたソフトウェア資源の利用価値は高まらず、ネットワークを介したソフトウェア利用に関する技術発展が促進されないことが懸念される。ネットワーク上で利用できるソフトウェアの包括的なディレクトリサービス [1] や、キーワード検索に基づくソフトウェア検索 [2] などの試みはなされているものの、そのほとんどがソフトウェアの構文情報に基づいており、各部品の正確な動作を知るにはソースコードを手にとって調べるほかないのが現状である。

近年、形式技法の分野において研究が始まった振舞仕様は、従来の代数仕様による抽象データ型の定義に加え、抽

象状態機械としての動的な振舞い定義するのに用いることができる。インターネット上で流通するソフトウェア部品やサービス等を、振舞仕様を用いて定義することにより、ソフトウェア部品およびサービスに対して、機能に基づく検索および検証を通じた再利用が可能となる。これを既存の遠隔実行機構と連動させることで、検索で得られた部品を即座に実行することも可能になる。このような技術は、ソフトウェア部品の流通と再利用を大いに促進するものと期待され、将来のソフトウェア産業の基盤技術の一角を占めるものと予想される。

## 2 目的

本テーマでは、開放型分散実行環境におけるソフトウェア部品の流通と再利用を促進することを目指し、部品あるいは部品の提供するサービスを仕様として定義することで、外部からの検索や利用に先立っての機能チェックといった検証作業を可能にするための基盤技術の開発を目的としている。

振舞仕様を用いて、インターネット上のソフトウェア部品あるいはサービスの動作仕様を定義する。これら定義された部品に対して、グラフィカルユーザインタフェースを通して、動作仕様、インタフェース等に基づく検索を可能にする。これにより、ユーザは、ソフトウェア開発に必要な部品を、ソースコードを調べることに無しに検索することができる。

さらに、検索された部品に対して、部品の仕様が要求仕

様を満たすかどうかを検証したり、仕様が決して危険な状態に陥ることがないかなどの不具合の検査を自動的に行う機能を提供する。これにより、開発するソフトウェアの利用に先だて、ソフトウェアの安全性、信頼性を確認することが可能になる。

### 3 研究開発の概要

本研究開発では、以下の機能の開発および実験を行った。

- 検証推論システム  
部品仕様が要求仕様を満たすかどうかを判定する詳細化検証機能、および、仕様に基づくシステムが決して異常な状態に陥らないこと (安全性) を自動検査するためのモデル検査機能である。
- リポジトリ管理システム  
GUI を通じたソフトウェア部品の登録、検索、検証起動等を行なうためのシステムである。部品間の参照関係は、XML を用いて管理され、モジュール名による検索、振舞仕様にに基づく検索インタフェースを提供する。振舞仕様にに基づく検索は、検証推論システムの機能と連携し実現される。
- 部品仕様シグネチャ変換システム  
リポジトリ管理システムに登録されるソフトウェア部品のシグネチャ部仕様を、EJB, JavaBeans 等の API を規定する JavaDoc から変換生成するためのシステムである。オブジェクト指向言語である Java のシグネチャ部仕様は、振舞仕様記述言語である CafeOBJ により定義する。
- 部品検索・検証に基づく開発プロセスにおける運用実験  
J2EE フレームワークに基づく WEB アプリケーションのリファレンス・インプリメンテーションとしてソースコードとともに公開される Java PetStore をベースとし、ソフトウェア部品の検索、変更によるアプリケーション開発において、上記のシステムの運用実験を行った。

### 4 システム構成と開発機能

本研究開発の開発システムの構成を示したものが図 1 (p. 2) である。

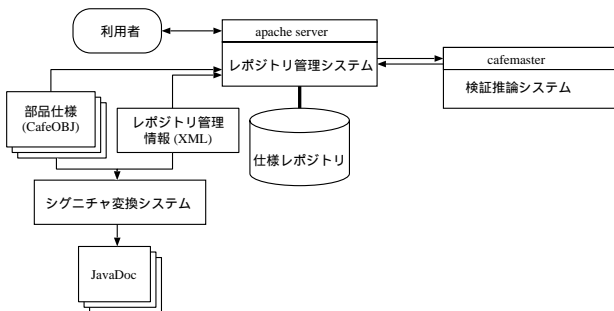


図 1: システム構成

利用者は、WEB ブラウザを介してレポジトリ管理システムにアクセスする。レポジトリ管理システムは、apache を介して、利用者から、部品仕様の登録、検索、検証等の要求を受け取り、処理結果を利用者に返す。あらかじめ部品仕様を一括して登録する場合には、シグネチャ変換システムにより生成した部品仕様である CafeOBJ モジュール定義と XML で記述されるレポジトリ管理情報に対するパスをレポジトリ管理システムの設定ファイルに設定する。レポジトリ管理システムから、検証推論システムへの検証

機能の実行は、TCP/IP ソケットを用いて実装された既存システムである cafemaster を介して行われる。

#### 4.1 検証推論システム

検証推論システムは、部品仕様が要求仕様を満たすかどうかの判定、および、仕様に基づくシステムが決して異常な状態に陥らないこと (安全性) の自動検査を行うシステムである。

以下では、検証推論システムを構成する機能の技術面についてまとめる。

#### シグネチャマッチング

シグネチャマッチング [14, 15] は仕様間のシグネチャ射を構成するための手法であり、これは仕様の等式部とは関係なく行われる。まず、隠蔽シグネチャ間の射の定義を与える。

定義 1 隠蔽シグネチャ射  $\varphi : \Sigma \rightarrow \Sigma'$  はシグネチャ射であって、隠蔽ソートを保存するものである。 $\varphi$  をビュー (view) と呼ぶことがある。

シグネチャマッチングの方式には、どのソートに対して代入を許すかによって様々なものが存在する。本研究開発では、ソフトウェア部品間で共有されるデータ型は同じ名前を持つべきであるという制約を設けることで、シグネチャマッチングを決定可能な文字列並び替え問題に帰着させ、効率の良い検索を実現する。この制約は、適切な名前空間の管理を行うことにより正当化されると考えている。この単純化されたシグネチャマッチングは CafeOBJ 上に実装されており、全ての可能なビューを生成する。

図 2 にモジュール M2 からモジュール ATM-CLIENT へのシグネチャマッチングを行っている CafeOBJ のセッションの様子が示されている。ここで ATM-CLIENT は文献 [5] に収められている ATM システム仕様の一部を成すものである。4 個の可能なビューが発見され、このうち V#4 が表示されている。

```

module* M2 {
  *[ E ]* [ Nat ]
  op elt : -> E
  bop ar1 : E -> Nat
  bop ar2 : E Nat -> E
  var I : Nat   var M : E
  eq ar1(ar2(M,I)) = I .
}
  
```

```

CafeOBJ> sigmatch (M2) to (ATM-CLIENT)
(V#4 V#3 V#2 V#1)
  
```

```

CafeOBJ> show view V#4
  
```

```

view V#4 from M2 to ATM-CLIENT {
  sort Nat -> Nat
  hsort E -> Atm
  hsort ?E -> ?Atm
  op (E : -> SortId) -> (Atm : -> SortId)
  op (Nat : -> SortId) -> (Nat : -> SortId)
  op (==_ : E E -> Bool) ->
    (==_ : _ HUniversal _ _ HUniversal _ -> Bool)
  op (elt : -> E) -> (no-atm : -> Atm)
  bop (ar1 : E -> Nat) -> (get-request : Atm -> Nat)
  bop (ar2 : E Nat -> E) -> (request : Nat Atm -> Atm)
}
  
```

```

CafeOBJ> check refinement V#4
yes
CafeOBJ>
  
```

図 2: CafeOBJ システム上でのシグネチャマッチングと詳細化検証

## 安全性モデル検査

ここでは、振舞仕様を対象として安全性モデル検査を抽象的に説明する。

まず状態集合は隠蔽ソート上の述語により定義することができる。述語により、無限個の要素を持つ状態集合に一つの記号表現を与えることができ、さらにメソッドによる述語変換 (predicate transformer) によって、状態遷移に関する前状態の集合を表す述語表現が得られる。全てのメソッドについて前状態の和をとること、すなわち

$$\text{pre}(P(X:h)_h)_{h'} \triangleq \sum_{\sigma:wh' \rightarrow h} \exists[V:w] P(\sigma(V,Y:h'))_h$$

によって、一回の遷移 (メソッド適用) によって述語  $P$  を満たす状態に達する可能性がある全ての状態の集合に対する述語が得られる。なお、上記の定義では多ソートの振舞仕様を厳密に取り扱うために、ソートにより指標付けられた述語の族を扱っていることに注意されたい。

以上を踏まえた上で、振舞 (安全性) モデル検査について考えてみる。抽象的にとらえた場合、安全性モデル検査は前状態関数  $\text{pre}$  の最小不動点の計算ととらえることができる。すなわち、述語  $P$  が安全であることを示すには、始状態を  $I$  として

$$I \not\subseteq \text{pre}^*(\neg P)$$

を示せばよい。ここで

$$\text{pre}^*(P) \triangleq \mu Z.P \vee \text{pre}(Z)$$

は  $P$  を含む  $\text{pre}$  の最小不動点であり、 $P$  を満たす状態へ到達する可能性のある全ての状態を表す述語を表す。

最小不動点  $\text{pre}^*(P)$  はよく知られた繰り返し計算によって計算可能であるが、状態数が無限の場合は有限回で収束するとは限らない。この問題は無限状態のシステムを扱う上で本質的に不可避の問題であり一般的な解法は存在しないが、我々の今までの経験では、安全であることが分かっている問題に対しては、期待通り有限回で収束する結果が得られている。

モデル検査のもう一つの特徴として、その反例を発見する能力が挙げられるが、ここでの抽象定式化では、不動点の繰り返し計算の過程で

$$I \subseteq \text{pre}^*(\neg P)$$

が判明した時点で反例を発見されることになる。

ここで説明したのは、後向きの安全性モデル検査として知られているアルゴリズム [8] の振舞仕様への応用であるが、述語計算により不動点  $\text{pre}^*(P)$  の繰り返し計算の収束判定が可能であることから、たとえ対象システムが無限状態であってもモデル検査を自動化することができるという事実が興味深い点である。これは、人手による仕様の変換や抽象化を経ずに安全性検査を可能な限り自動化するという点から特に重要であると考えられる。

以上の議論をまとめて、我々の (安全性) モデル検査手続きを以下に示す。この手続きは述語の族  $F^i$  ( $i \geq 0$ ) と自然数の変数  $N$  を維持しながら計算を行い、CafeOBJ システム上で定理証明器 PigNose を用いて実装されている。記法の簡略化のため以下ではただ一つの隠蔽ソート  $h$  が存在する場合を扱い、また  $F(X:h)$  により  $F^i$  ( $i \geq 0$ ) に含まれる全ての述語の論理和をとった述語を表すものとする。

Model checking procedure  $\text{MC}(I,P)$

( $I$ : initial state,  $P$ : safety predicate)

- 1)  $F^0 = \{\neg P(X:h)\}$ ;  $F^i = \phi$  ( $i > 0$ );  $N = 1$
- 2) for each formula  $Q \in F^{N-1}$   
for each method  $\sigma:wh \rightarrow h$ , let  
 $G(X:h)$  be  $(\exists V:w) Q(\sigma(V,X:h))$  and do  
(a) if  $(\forall X:h) G(X) \Rightarrow F(X)$  is provable then skip  
(b) if  $G(I)$  is provable then exit with failure  
(c) if  $\neg G(I)$  is provable then add  $G$  to  $F^N$  and skip  
(d) exit with unknown
- 3) if  $F^N = \phi$  then exit with success
- 4)  $N = N + 1$ ; go to Step 2

上記で論理式の証明可能性を検査する際に PigNose が起動される。定理証明は決定不能な問題であるので、一定の計算資源を消費したあとに PigNose は作業を中断し、結果を “unprovable” として返すものとしている。この計算資源の上限を定めるパラメータは複数用意しており、ユーザーが設定できるようになっている。

繰り返し一回で終わらない一般的な例として、Lamport による相互排除問題の古典的解である Bakery アルゴリズムがある。プロセスの相互排除の場合、4 回目の繰り返しまで進んで不動点に到達し、安全性検査に成功する。この問題は有限モデル検査で自動検証するのが不可能な問題であり、これが自動的に検証できるという点で、本研究での手法が従来より広範な安全性検査に応用できることが分かるであろう。検証に要した時間は PentiumIII 750MHz を搭載したマシンで約 1 分である。

## 詳細化検証

シグネチャマッチングにより生成されたビューが、実際に仕様の射になっているかを調べるのが詳細化検証である。これは、もとの仕様がビューによって保存されるかを調べることに相当し、機能に基づくソフトウェア部品検索の根幹となる部分である。純粋にモデル論的な振舞詳細化の定義を下に与える。

定義 2 隠蔽シグネチャ射  $\varphi: (\Sigma, E) \rightarrow (\Sigma', E')$  は、あらゆる  $(\Sigma', E')$ -代数  $M'$  について  $\varphi M' \models_{\Sigma} E$  が成り立つときに、振舞仕様  $(\Sigma, E)$  から  $(\Sigma', E')$  への振舞詳細化であると言われる。ただし、 $\varphi M'$  は  $M'$  を  $\Sigma$  代数とみなしたものであるとする。

$\varphi$  が振舞詳細化であることを示すには、 $(\Sigma, E)$  で成立する全ての可視等式が  $(\Sigma', E')$  で成立することを示せばよいことが知られている [16]。

補題 1 隠蔽シグネチャ射  $\varphi: (\Sigma, E) \rightarrow (\Sigma', E')$  は、全ての等式  $e \in E$  と全ての可視  $\Sigma$  文脈  $c$  について  $E' \models \varphi(c[e])$  が成り立つとき、振舞詳細化である。ただし、 $e$  を  $(\forall X) t = t'$  の形の等式としたとき、 $c[e]$  は等式  $(\forall X) c[t] = c[t']$  を表すものとする。

したがって、ビュー  $\varphi: (\Sigma, E) \rightarrow (\Sigma', E')$  が振舞詳細化であるかを調べるには以下の手順を踏めばよい。

- $\varphi$  を  $E$  の各等式  $e$  に適用し ( $e * \varphi$  と表す) 翻訳  $e'$  を得る。
- $e'$  が振舞等式でなければ、従来の等式推論により  $e'$  が  $E'$  から導けるかを調べる。
- $e'$  が振舞等式であれば、 $(\Sigma', E')$  において  $e'$  に対する余帰納法を、 $(\Sigma, E)$  の振舞演算子だけと考慮して行う。

この手順は CafeOBJ システム上での自動定理証明器である PigNose を用いて実装されている。図 2 においてシ

グネチャマッチングに続いて詳細化検証を行う CafeOBJ のセッションの一部が示されている。結果は正しく、get-request と request の演算子は M2 の等式を満たすことが分かる。

#### 4.2 リポジトリ管理システム

リポジトリ管理システムは、詳細化検証システム、モデル検査システム等を統合し、部品検索、検証のための統合された GUI を提供するものである。図 1 (p. 2) に示した通り、apache サーバ上で Servlet 等の Java を用いて実装され、WEB アプリケーションとして動作し、TCP/IP socket を用いた cafemaster を介して、検証推論システムと連携する。利用者は、WEB ブラウザのみによって、レポジトリ管理システムが提供する全機能にアクセスすることができる。

#### 4.3 部品仕様シグニチャ変換システム

部品仕様シグニチャ変換システムは、部品仕様のシグニチャについて、レポジトリ管理システムで扱う振舞仕様言語 CafeOBJ による定義に変換するものである。具体的には、EJB, JavaBeans 等の API を規定する JavaDoc から CafeOBJ の Module 定義を変換生成するシステムである。

シグニチャ変換手法の基本的な方針をまとめると以下のようになる。

- java クラスインスタンスの状態集合を hidden sort で表現する。  
(java インスタンスの状態を決めるインスタンス変数全体が、1 つの hidden sort でまとめて表現される。)
- java method は、当該クラスに対応する hidden sort の状態を変化させる振舞いオペレータとして表現する。(返り値が void の場合の基本パターン)
- java のクラス定義は、当該クラスに対応する hidden sort とその hidden sort を arity に含む振舞いオペレータ全体によって表現される。

以下では、WEB モールのための部品を例にあげ、Java API から CafeOBJ モジュール定義への変換法についてまとめる。例とする Java クラス定義におけるシグニチャは、および、変換後の CafeOBJ モジュールを先に示す。

```
public class PurchasedOrderLineImpl // (a)
    extends OrderLineImpl // (b)
    implements PurchasedOrderLine { // (f)
public SmartHandle giver;
public SmartHandle packingList;

public PurchasedOrderLineImpl();
public java.lang.Boolean getReturned(); // (e1)
public void setReturned
    (java.lang.Boolean returned); // (e2)
public void setGiver(Customer giver) // (e3)
    throws java.rmi.RemoteException; // (g)
    :
}
```

```
mod! PurchasedOrderLineImpl { -- (a)
    extending(OrderLineImpl) -- (b) class の継承
    * [ HOrderLineImpl < HPurchasedOrderLineImpl ] *
        -- (c)
        [ VPurchasedOrderLine VSmartHandle ... ] -- (d)

-- for the constructor
op purchasedOrderLineImpl : -> HPurchasedOrderLineImpl .

-- for the java method 'getReturned' -- (e1)
bop getReturned1 : HPurchasedOrderLineImpl
    -> HPurchasedOrderLineImpl .
bop getReturned2 : HPurchasedOrderLineImpl
    -> Vjava.lang.Boolean .

-- for the java method 'setReturned'
bop setReturned : HPurchasedOrderLineImpl Vjava.lang.Boolean
    -> HPurchasedOrderLineImpl . -- (e2)

-- for the java method 'setGiver'
bop attrCustomer : HCustomer -> VCustomer . -- (e3)
bop getGiver : HPurchasedOrderLineImpl VCustomer
    -> HPurchasedOrderLineImpl .
    :
}

view PurchasedOrderLine-Implimentation -- (f)
from PurchasedOrderLine to PurchasedOrderLineImpl {

hsort HPurchasedOrderLine -> HPurchasedOrderLineImpl,
sort Vjava.lang.Boolean -> Vjava.lang.Boolean,

bop getReturned1 -> getReturned1,
bop getReturned2 -> getReturned2,

bop setReturned -> setReturned
}
```

以下に、具体的な変換法を示す。

- Java 基本型および標準クラスに対応したソートを導入する。  
Java 基本型, Java 標準パッケージ内のクラスに該当するもの (EJB パッケージ内で絶対パスで指定されるクラス) を visible sort として宣言した CafeOBJ モジュール BASE を生成し、モジュールの輸入により、ソートを導入する。例においては、(b) に示すモジュールの輸入により、クラス継承関係に応じて、モジュールが輸入され、最終的に、BASE モジュールが輸入される。
- EJB パッケージ内の各クラスに対応した CafeOBJ Module を以下の通り生成する。ただし、
  - 当該クラスに対応するモジュール名を導入する。(例中 (a))
  - 継承される java class (,interface) を extending で import する。これにより method の継承を実現する。(例中 (b))
  - 当該クラスに対応する hidden sort を、継承するクラスの hidden sort の sub sort としての宣言する。インスタンス変数の継承を実現する。(例中 (c))
  - java method 引数において新規に出現するクラスを visible sort として宣言する。(例中 (d)) visible sort 導入のために以下のクラス集合を求める。
    - \* 当該クラス A の signature に出現する全てのクラスの集合: Sa
    - \* クラス A が継承するクラスを再帰的に辿りその中に出現するクラスの集合: Sb
    - \* 上記基本型, Java 標準クラスの集合: Sc
 クラス集合 Sa (Sb + Sc) に対応する visible sort を宣言する。
  - java method に対応する振舞いオペレータ宣言を生成する。当該クラスに対応する hidden sort を arity, coarity に追加する。(例中 (e))

- \* 戻り値を持つ method は、戻り値に対応する sort と当該クラス対応した hidden sort との内積を射影した bop の対で表現する。(例中 (e1))
- \* メソッド引数のクラスは全て visible sort とする。(例中 (e2))
- \* 継承関係にあるクラスに該当する hidden sort を引数にもつメソッドは、visible sort を返すオペレータを用意する。(例中 (e3))
- java interface を実装した class 宣言は、通常のモジュールとして生成し、interface に対応する CafeOBJ module から、class に対応する module への view 宣言により、必要とするインタフェースの存在に関する制約を与える。(例中 (f))
- java 例外ハンドリングのための throws は除外する。(例中 (g))

以上の方法により、Java クラス定義のシグニチャを CafeOBJ モジュールに変換することができる。一方、同様の変換手法を、以下の Java インタフェースに適用した場合を考える。

```
public interface PurchasedOrderLine extends OrderLine {
    public java.lang.Boolean getReturned();
    public void setReturned(java.lang.Boolean returned);
    :
    :
}
```

この場合、得られる CafeOBJ モジュールは以下の通りである。Java インタフェースでは、CafeOBJ のモジュール宣言において loose semantics を用いた mod\* により定義される。また、インタフェースの継承は、クラスの継承と異なり、複数継承できる点異なる。

```
mod* PurchasedOrderLine {
    extending(OrderLine) -- (b) interface の継承
    *[ HPurchasedOrderLine]*
    [ Vjava.lang.Boolean ... ]

    bop getReturned : HPurchasedOrderLine
        -> HPurchasedOrderLine .
    bop getReturned : HPurchasedOrderLine
        -> Vjava.lang.Boolean .
    bop setReturned : HPurchasedOrderLine Vjava.lang.Boolean
        -> HPurchasedOrderLine .
    :
}
```

Java のシグニチャと CafeOBJ のモジュール定義要素の対応関係を示したものが、表 1(p.5) である。

表 1: Java シグニチャと CafeOBJ の対応関係

java シグニチャ	CafeOBJ
class	module (hidden sort + bop)
instance 状態集合	hidden sort
instance 状態	hidden sort の term
method	当該クラスの hidden sort から hidden sort への bop
interface	(parameter) module
class 継承 (extends)	import (extending)
interface 継承 (extends)	import (extending)
implements	view (or parameterized module)

## 5 実験

### 5.1 開発プロセスにおける利用シナリオ

ソフトウェアの一部のモジュールに対する変更が、他のモジュールに与える影響が把握でき、その影響を小さく抑えることができれば、特定のソフトウェアから、機能変更、機能拡張を加え、目的に即した新たなアプリケーションを効果的開発することができる。

Smalltalk によるアプリケーション開発において提唱された MVC(Model-View-Controller) アーキテクチャは、Model に相当するアプリケーション・ロジック、View に相当するプレゼンテーション・ロジック、および、アプリケーション・ロジックと、プレゼンテーション・ロジックの相互作用に関する制御をおこなう Controller を分離することにより、ソフトウェアの開発および改良を効率化することを旨としたものである。

近年、J2EE (Java 2 Enterprise Edition) において、この MVC アーキテクチャに基づき、WEB アプリケーション・サーバに基づく WEB ショッピング・サービス・アプリケーションの Reference Implementation として、PetStore と呼ばれるシステムが公開された。このアプリケーションは、EJB(Enterprise JavaBeans), JSP, Servlet といった WEB アプリケーション・サーバの中核的な技術を用い、実装されたものである。

本テーマでは、この J2EE の Reference Implementation である PetStore をベースに、WEB ショッピング・アプリケーションを開発する方法論について検討を行い、統合化システムをもちいて、その方法論に基づく開発支援のための運用実験を行った。

図 3 (p. 5) は、本研究開発で提案する開発アプローチを明示したものである。

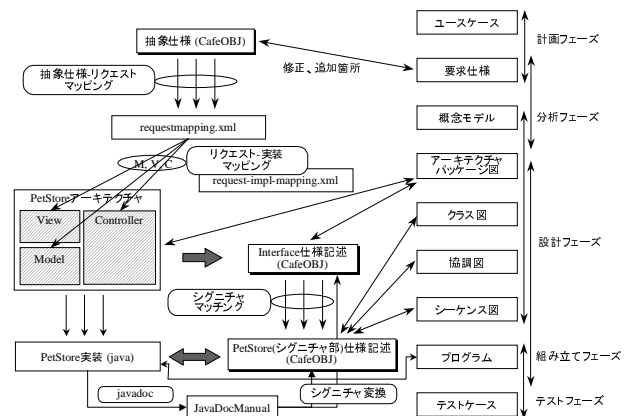


図 3: PetStore をベースとした影響解析に基づくソフトウェア開発提案手法

図左側には、本開発手法の流れを示し、右側には、UML 開発方法論におけるフェーズおよび、そのプロセスで利用されるダイアグラム、仕様を示し、本開発手法との対応関係を示したものである。

まず、本開発手法の前提条件をまとめると以下のようになる。

- 基本アーキテクチャの共通性  
開発のターゲットのアーキテクチャは、ベースにするシステムのアーキテクチャと、基本構造に共通性がある必要がある。WEB アプリケーション、ネッ

トショッピング等の場合、アーキテクチャの基本構造に共通性がある場合が多いため、この前提は、現実的である。

- ベースアプリケーションの抽象仕様  
ベースアプリケーションの抽象仕様を CafeOBJ で用意する必要がある。開発者が作成する必要はない。一つのベースアプリケーションから、複数の WEB アプリケーションを開発するためには、ベースとなるアプリケーションの抽象仕様を一つだけ用意すればよい。
- 提供される抽象仕様に対する知識  
開発者は、類似部品の検索のためには、提供されるベースアプリケーションに関する抽象仕様を理解できる必要がある。
- リクエストタイプと実装マップ  
ベースとなる WEB アプリケーションのリクエストタイプと、そのリクエストを処理するための実装コードへのマップを事前に用意する必要がある。
- 抽象仕様、クラス設計仕様レベルでの検証  
検証は、抽象レベルでは、動作仕様等の意味的な検証まで行えるが、クラス設計仕様 (Java クラス API に関する仕様) では、仕様に公理の追加しなければ、意味的検証は行えない。

本システムを用いた利用シナリオは以下のようになる。

- 1) ソフトウェア要件の抽出  
ユーザから、新規に開発したい WEB アプリケーションの要件を抽出する。
- 2) 要求仕様の作成  
事前に提供される抽象仕様の記述形式に合わせ、ユーザから抽出したソフトウェア要件を形式仕様として記述する。
- 3) 要求仕様と比較し、抽象仕様で変更が必要な場所の特定  
要求仕様と、抽象仕様との対応関係から、ベースとなる WEB アプリケーションに対して必要な変更、機能追加部を特定する。
- 4) リクエスト-実装マップ情報による実装変更部の特定  
リクエスト-実装マップ情報 request-impl--mapping.xml をもとに、リクエストの種類から、実装部を特定する。
- 5) アーキテクチャレベルから実装レベルのコードの特定  
リクエスト-実装マップ情報 request-impl--mapping.xml により、リクエストタイプから実装部を特定する代りに、アーキテクチャを規定する Java インタフェースをと規定する。
- 6) 再利用可能な類似コードの特定  
リクエストタイプに関わる Java インタフェースから、部品検索システムを用いて、類似部品、関連部品を検索取得する。
- 7) 実装コードの修正、追加  
上記仕様検索に基づく、類似部品、関連部品をもとに、必要な修正あるいは、新規追加を行う。

以上のプロセスを繰り返すことにより、ベースとする WEB アプリケーションから、開発者が目的とするアプリケーションに機能を変更あるいは追加する。

## 5.2 インタフェース仕様に基づく部品検索

Java PetStore は、MVC アーキテクチャに基づき実装されている。このうち、コントローラ部の実装は、各種のイベントハンドラーのインタフェース定義により、アーキテクチャが規定されている。

図 4 (p. 6) は、Controller 部の実装の構成をしめしたものである。

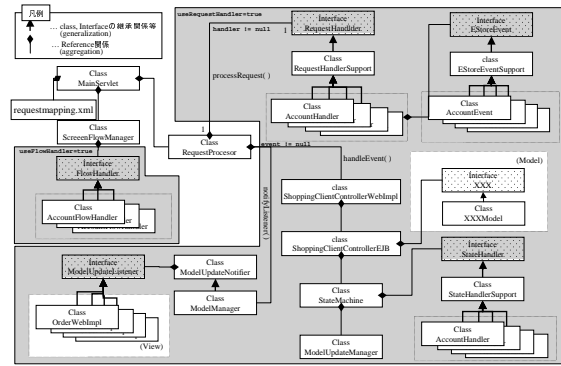


図 4: Controller 部の構成

Controller 部は、Model 部と View 部の連携と制御を行うものである。WEB アプリケーションのサーバに対するリクエストは、サーブレットコードの MainServlet.java により集中的に受け取られ、RequestProcessor.java により、リクエストの処理ごとに各ハンドラーに渡されるリクエスト・ハンドリング型の実装が行われている。リクエストの種類とハンドラーの対応関係、処理結果のスクリーンフロー等は、requestmapping.xml というファイルのテキスト情報で規定されている。

リクエストは、モデルの状態遷移が必要な場合には、RequestHandler によりリクエストタイプから、イベントタイプに変換される。この結果得られたイベントオブジェクトは、ShoppingClientControllerWebImpl および StateMachine を介して StateHandlerSupport によって実際のモデル状態遷移が引き起こされる。

さらに ModelManager, ModelUpdateNotifier を介して、view の変更のための通知が行われている。この通知は、ScreenFlowManager, FlowHandler 等によって受け取られ、処理後の適切な画面表示が行われる。

インタフェース仕様に基づく部品検索の例として RequestHandler インタフェースを検索条件として検索した結果を示す。図 5 (p. 6) は、RequestHandler を対象にした検索を行う画面を示している。



図 5: RequestHandler インタフェースによる検索開始画面

この検索画面による検索の結果、RequestHandler インタフェースに適合する仕様がすべて取得されたことを示す画面が、図 6 (p. 7) である。

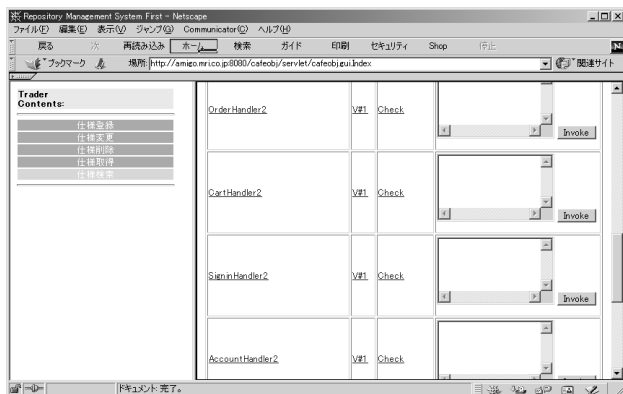


図 6: RequestHandler インタフェースによる検索結果



図 8: CONTAINER のシグニチャマッチによる検索結果

### 5.3 詳細化検証に基づく部品検索

詳細化検証に基づく部品検索実験の例として、データの一般的な入れ物を定義するCONTAINERの定義に関する検索結果を示す。検索対象としての要求仕様を規定するCONTAINERの定義は、以下の通りである。

```

mod* CONTAINER(X :: TRIV) {
  *[ Container ]*
  op empty : -> Container
  bop store : Elt Container -> Container
  bop val : Container -> Elt
  var E : Elt var C : Container
  eq val(store(E,C)) = E .
}

```

詳細化検証に基づく部品検索は、1) シグニチャマッチによるインタフェース適合性による検索、2) インタフェース適合部品に対して部品の動作を定義する振舞等式の充足性に基づく部品の絞り込み、といった2段階の手順で行う。

図 7 (p. 7) は、レポジトリ管理システムに登録された各種のデータ構造を規定する部品の一覧を示す画面である。



図 9: STACK が、CONTAINER の要求仕様を満たすか検証した結果の画面

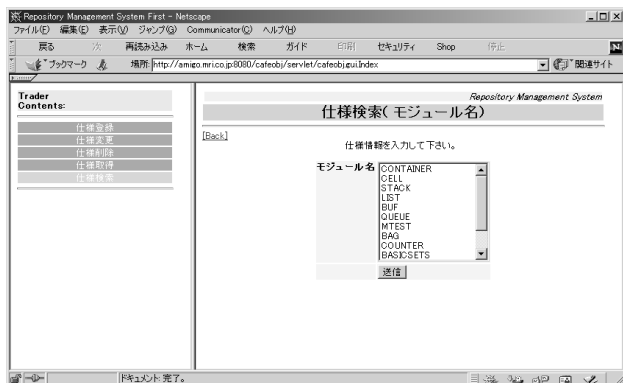


図 7: 各種データ構造を規定する部品の検索画面

CONTAINER のインタフェースに適合する部品をシグニチャマッチで検索した結果が図 8 (p. 7) である。

検索結果を示す図 8 では、First-In-First-Out 型の QUEUE, STACK, LIST 等の一般的なデータ型をもつ部品とインタフェースに関して適合することが示されている。この時、STACK とCONTAINER のインタフェースの対応を定義する写像関係のもとで、CONTAINER において規定される条件 (等式) が満たされるか検査した結果が図 9 (p. 7) である。

図 9 の画面において、STACK の行における refinement 列で “yes” と示されているのは、STACK が要求仕様を満たしていることが示されたことを示している。

一方、仕様検索により、インタフェースが適合した部品 MTEST について、同様に詳細化検証を行ったところ、図 10 (p. 7) に示す通り、以下の等式の証明に失敗したことが示されている。

$$eq \text{ val}(\text{store}(E,C)) = E$$



図 10: CONTAINER の機能のうち、MTEST によって満たされていないものを表示する画面

図 10 の例は、インタフェースは適合するが、要求仕様には適合しない部品であることを示しており、インタフェースの適合性に基づく検索で、多数の部品が見つかった場合の絞り込みの事例を示している。

## 6 まとめ

本研究開発では、インターネット上のソフトウェア部品およびサービスの流通と再利用の促進を目指し、部品あるいは部品の提供するサービスを仕様として定義することで、外部からの検索や利用に先立つ機能チェックといった検証作業を可能にするためのシステムを開発した。

これらを実現するための基盤技術として以下の機能を開発した。

- 検証推論システム  
部品仕様が要求仕様を満たすかどうかの判定、および、仕様に基づくシステムが決して異常な状態に陥らないこと (安全性) を自動検査をおこなう。
- リポジトリ管理システム  
ソフトウェア部品の登録、検索、検証起動等を行なう。
- 部品仕様シグニチャ変換システム  
リポジトリ管理システムに登録されるソフトウェア部品のシグニチャ部仕様を、EJB, JavaBeans 等の API を規定する JavaDoc から変換生成する。

これらの機能は統合化され、WEB ブラウザによる GUI として提供される。

## 7 参加企業及び機関

本研究開発は、北陸先端科学技術大学院大学二木研究室および株式会社 SRA 先端技術研究所が有する代数仕様記述言語に関する技術に基づき実施された。株式会社 SRA 先端技術研究所および独立行政法人 産業技術総合研究所サイバーアシスト研究センターは、仕様検証システムを開発し、日本ユニシス株式会社はレポジトリ管理システムを開発し、株式会社三菱総合研究所は、部品仕様シグニチャ変換システムの開発および、WEB アプリケーションをベースとした開発プロセスにおける運用実験を行った。北陸先端科学技術大学院大学二木研究室、東京大学大学院情報理工学系研究科萩谷研究室および、デザイナーズデンは、代数仕様に関する技術アドバイスの提供を行った。

## 8 参考文献

- [1] Universal Description, Discovery and Integration of Business for the Web, homepage at <http://www.uddi.org>
- [2] Robert Seacord, Scott Hissam and Kurt Wallnau, "Agora: A Search Engine for Software Components", Technical Report CMU/SEI-98-TR-011, Carnegie Mellon Software Engineering Institute, 1998
- [3] Joseph Goguen and Grant Malcolm, "A Hidden Agenda", in Theoretical Computer Science, Vol.245 No.1, 2000, pp.55-101
- [4] Răzvan Diaconescu and Kokichi Futatsugi, CafeOBJ Report. World Scientific, 1998
- [5] Kokichi Futatsugi and Ataru Nakagawa, "An Overview of CAFE Specification Environment", in the Proceedings of IEEE International Conference on Formal Engineering Methods'97, 1997
- [6] Amy Zaremski and Jeannette Wing, "Signature Matching, a Tool for Using Software Libraries", in ACM Transactions on Software Engineering and Methodology (TOSEM), Vol.4 No.2, 1995, pp.146-170
- [7] William McCune, "OTTER3.0 Reference Manual and Guide", Technical Report ANL-94/6, Argonne National Laboratory, 1994, available at <http://www-unix.mcs.anl.gov/AR/otter/>
- [8] Patrick Cousot and Radhia Cousot, "Refining Model Checking by Abstract Interpretation", in Automated Software Engineering Journal, Vol.6 No.1, 1999, pp.69-95
- [9] BEA Systems, WebLogic Commerce Server, <http://www.bea.com/products/weblogic/portal/-index.shtml>
- [10] Sun Microsystems, J2EE Reference Implementation Java PetStore, <http://java.sun.com/j2ee/>
- [11] Sun Microsystems, JavaCard Technology, <http://java.sun.com/products/javacard/>
- [12] Sun Microsystems, J2ME The Mobile Information Device Profile (MIDP), <http://www.sun.com/software/communitysource/midp/>
- [13] Masaki Ishiguro, Ataru Nakagawa: Proof Assistance for Algebraic Specifications Based on Proof Obligations in CafeOBJ, Cafe: An Industrial-Strength Algebraic Formal Method, Elsevier Science Ltd, 2000, 79-96
- [14] Zaremski, A.M., Wing, J.M.: Signature Matching: a Tool for Using Software Libraries. ACM Transactions on Software Engineering and Methodology (TOSEM) 4(2) (1995) 146-170
- [15] Goguen, J., Meseguer, J., Luqi, Zhang, D., Berzins, V.: Software Component Search. Journal of Systems Integration 6 (1996) 93-134
- [16] Malcolm, G., Goguen, J.: Proving Correctness of Refinement and Implementation. Technical Monograph PRG-114, Programming Research Group, University of Oxford (1994)