# 実行環境に依存する部分を含めた
# ソフトウェア部品の形式的仕様の記述

北陸先端科学技術大学院大学
海谷 治彦

Dec. 3, 1998
日立製作所 泉山ホールにて

# 発表の概要

- 背景 〜 なぜ形式的な部品仕様が必要か?
- どのような仕様が必要か? シグニチャ, 事前後条件 ....
- 部品の機能や振る舞いに影響を与える**環境**の仕様も記述.
- どのような環境を記述すべきか?
- 仕様記述例: Java の RMI 部品.
  - 環境の仕様なしでは実際のプログラムの振る舞いと異なる仕様となる例.
  - 環境の仕様によって実際のプログラムの振る舞いと仕様が一致する例.
- まとめと今後の課題.

# 背景

- ソフトウェア部品を (再) 利用には，その仕様が必要.
- 「形式手法を用いた場合にのみソフトウェアの再利用は成功し，かつ，再利用部品の開発に用いた場合にのみ形式手法は成功を納める.」[1]
  - 正確かつ簡潔な仕様がなければ部品の誤再利用が起る [2].
  - 繰り返しの利用によって，形式記述や手法のコストを合理的な額に抑えることができる.

---

Bertrand Meyer. The Next Software Breakthrough. COMPUTER, Vol. 30, No. 7, pp. 113–114, Jul. 1997.

Jean-Marc Jezequel and Bertrand Meyer. Design by Contract: The Lessons of Ariane. COMPUTER, Vol. 30, No. 1, pp. 129–130, Jan. 1997.

## 形式的に記述されている仕様内容

- 部品利用の際の名前や型の検査: シグニチャ
  (例) `public static Remote lookup(String name) throws ....`
- 部品呼び出しに関わる意味: Design by Contract(DBC)
  - 事前条件: 部品がクライアントに期待する性質.
  - 事後条件: 部品が結果として保証する性質.
  - 不変表明: 部品が維持する包括的な条件.

  基本的には部品の内部状態と入出力値のみを用いて定義されている.

## 他にどのような仕様が必要か?

計算機環境 (の状態) も部品の振る舞いを規定する.

- 異なるプラットフォームで部品をコンパイル & 利用するための環境分析: (例) `autoconf` の `configure`.
- 他の環境との関係と責任分担: モーバイルコードは, プログラム自身だけでなく,

  実行時環境. ブラウザ, OS, ネットワークなどの環境と共に機能している [3] .

  (例) Java のクラスロードなどでは, ファイルシステム (OS) 内のロードモジュール配置と, Java VM の両方が実行に関わる.

---

Tommy Thorn. p.231, Programming languages for mobile code. *ACM Computing Surveys*, Vol. 29, No. 3, pp. 213–239, Sep. 1997.

## 本研究の目的

環境の変化によって，部品の機能や振る舞いがどのように変化するかを形式的に仕様化.

- 部品を適切な環境下で利用するための簡潔なマニュアル.
- 記述対象: 今後，利用が増加する分野の部品 – 分散アプリのための言語 / 部品，Java 言語の RMI クラス群.
- アップレットではなく，アプリケーションを対象とする.
- 仕様記述言語: Z 記法 – 広く知られており，OO 部品などとも相性は良い.
- 記述方針: Meyer の DBC – 事前 / 事後条件，不変命題.

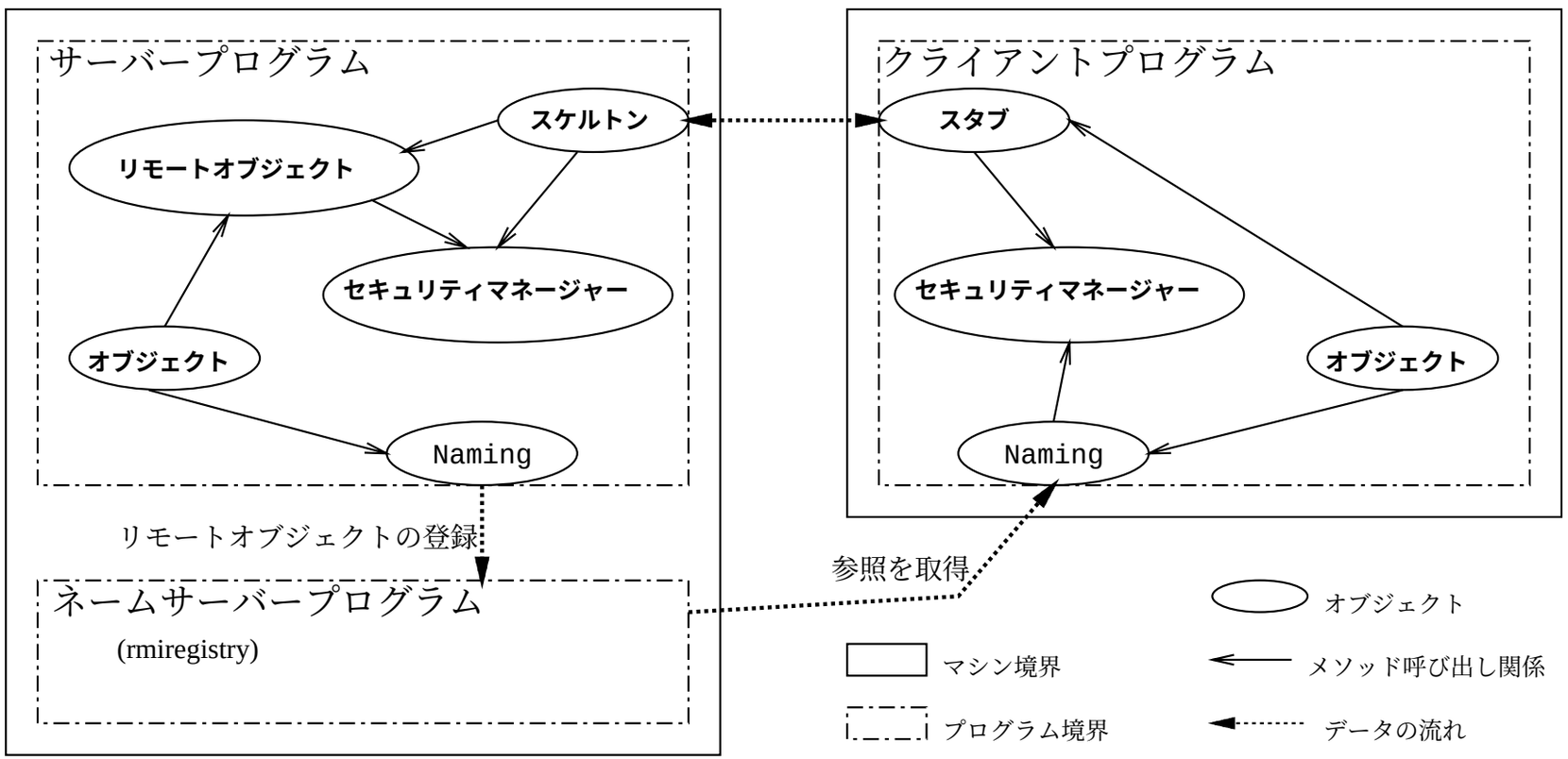
## 環境の何が部品に影響を与えるか? – Java の場合

- クラスファイルの配置: 動的ロード機構によりクラスファイルがどこ (ローカルファイルシステム or ネットワーク越し) からロードされるか? 振る舞いが変る.
- 他のサーバー類の動作状態: ネームサーバーが正常に動作しているか?
- ネットワークの性質: トランスポート層が直接ソケット通信か, ファイヤウォール越しか?
- ブラウザの違い: アプレット下で部品を利用した場合, そのブラウザの性質によって動作が異なる場合がある [4] .

---

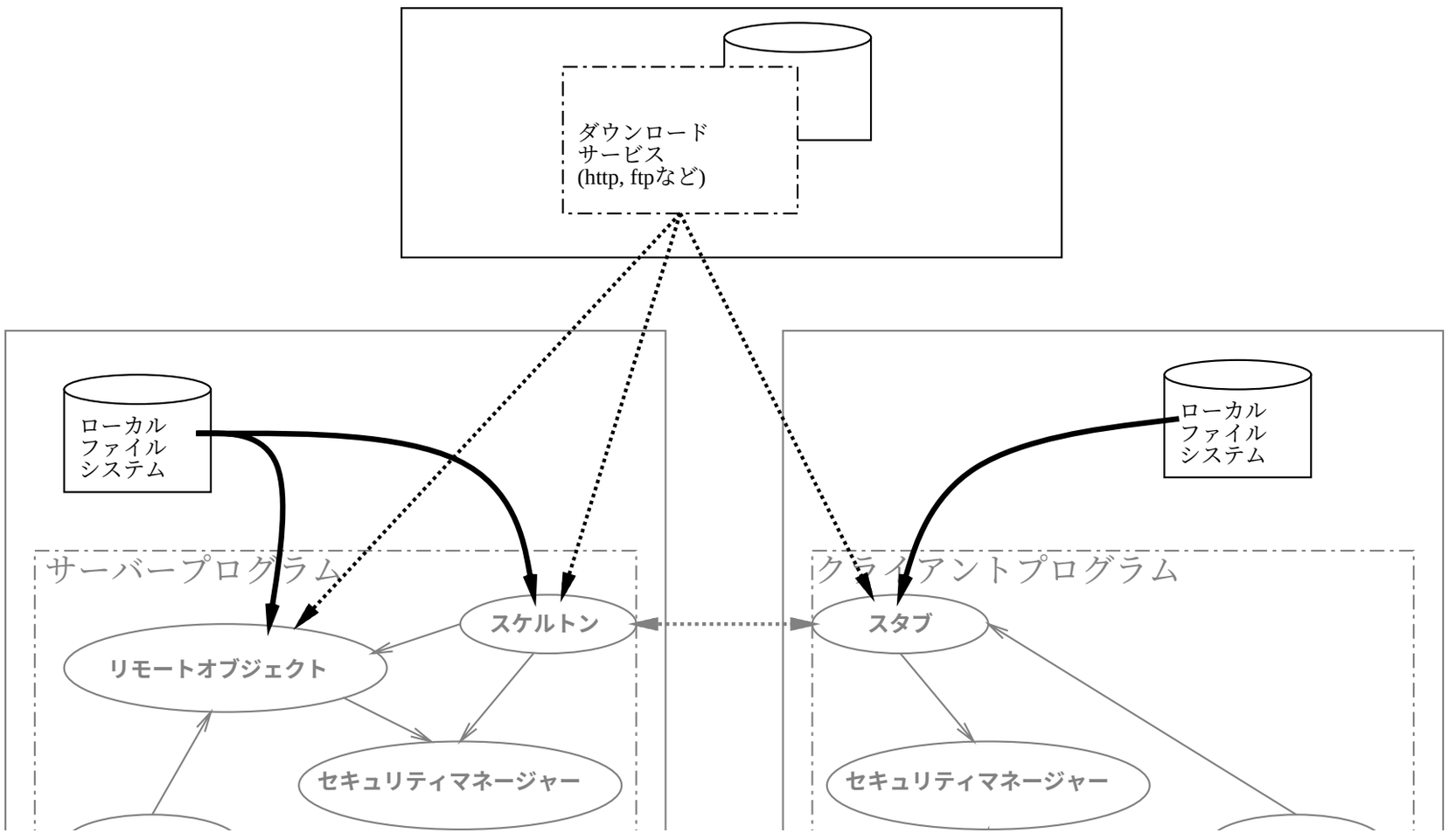
Drew Dean, Edward W. Felten, and Dan S. Wallach. Java Security: From HotJava to Netscape and Beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996.

# RMI システムの概要




RPC のオブジェクト指向版.

# 着目する実行環境: 動的ロード 機構



# 実行環境の仕様記述の方針 (1 / 2)

動的ロード機構に関する環境は,

$$LoadPlace ::= Local \mid Network$$

という型の値をもつ，以下のような状態スキーマとする.

環境

$$impl, skel : LoadPlace$$

∵ 振る舞いの変化は，ファイルのロード元に依存するため．これらが，それぞれの部品をどのように制約するかは部品 (メソッド) の仕様の方に書く.

# 実行環境の仕様記述の方針 (2 / 2)

*pre*.オブジェクト名.メソッド名

オブジェクト名; 入力値?
参照する他のオブジェクト; 環境

環境による部品の利用制約

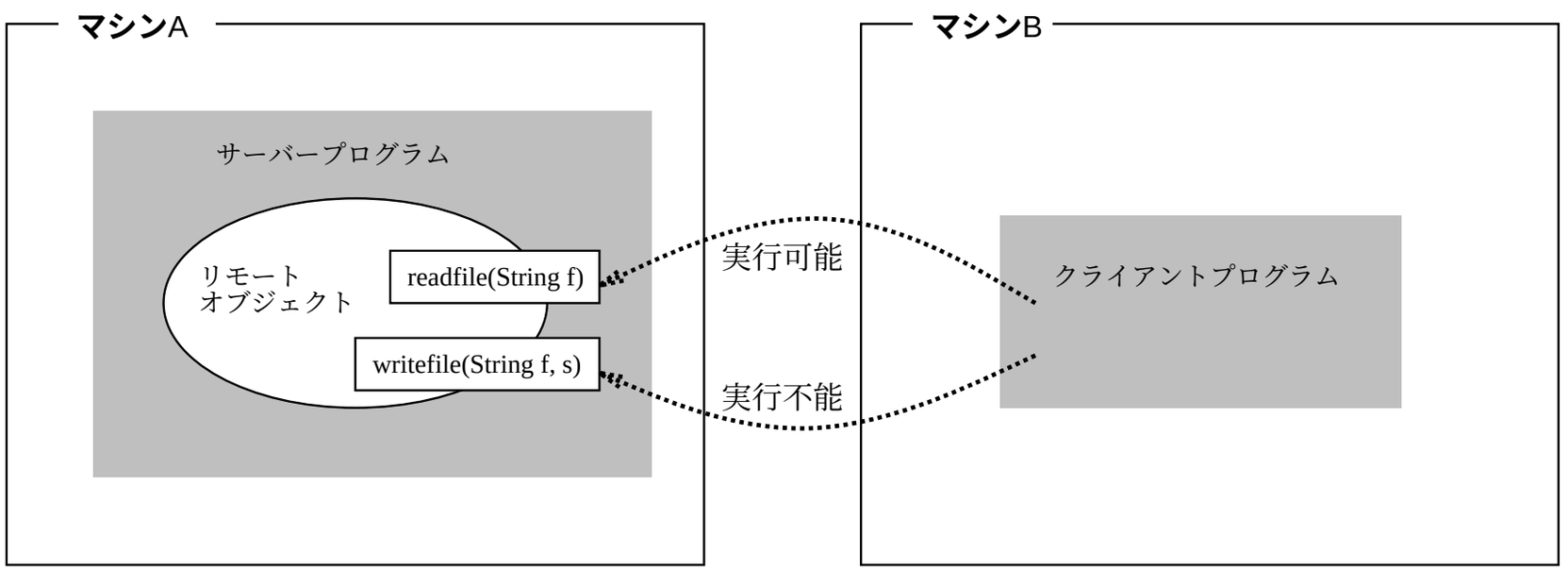
オブジェクト名.メソッド名

Δオブジェクト名; 入力値?; 出力値!
Ξ参照する他のオブジェクト; Δ環境

事後条件

# 例題

サーバープログラム側のファイルを，リモート側から読めるが，書けないプログラムをセキュリティマネージャーを緩めることで実現する．



# 部品のソース

セキュリティマネージャー:

```
class MyManager extends RMISecurityManager{
  public synchronized void checkRead(String file){}
  public synchronized void checkWrite(String file){
    super.checkWrite(file);
  }
}
```

リモートオブジェクト:

```
public class MyRObject
 extends UnicastRemoteObject implements MyInterface{
  public MyRObject() throws RemoteException{ super(); }
  public String readfile(String s) throws RemoteException{
      return new String(SimpleIO.readfile(s));
  }
  public boolean writefile(String f, String s)
    throws RemoteException{
      return SimpleIO.writefile(f, s);
  }
}
```

# 従来的な部品の仕様 (1 / 2)

**MyManager**

$checkTable$ : 制限される動作 $\rightarrow \mathbb{B}$

**InitMyManager**

$MyManager'$

---

$checkTable'$ 読 $=$ true

$checkTable'$ 書 $=$ false

$\vdots$

# 従来的な部品の仕様 (2 / 2)

**MyRObject**

$serverFS$ : ファイル名 $\pfun$ ファイル内容

**MyRObject.writefile**

$f? :$ ファイル名; $s? :$ ファイル内容; $rep! : Report$;
$\Delta MyRObject$; $\Xi MyManager$

---

$checkTable$ 書 $\Rightarrow$ $serverFS'\ f? = s?$
$\quad \mathrm{dom}\ serverFS \cup \{f?\} = \mathrm{dom}\ serverFS'$; $rep! = OK$
$\neg checkTable$ 書 $\Rightarrow$
$\quad \theta MyRObject = \theta MyRObject'$; $rep! = Exception$

## プログラムの実行結果と異なる推論結果

ここでのプログラムは書き込みはできないことが期待され，部品仕様をもとにすると

$$MyRObject.write \mid serverFS\ f? \neq s? \vdash serverFS = serverFS'$$

である性質が推論できる.

しかし，**実際には**上記が成り立たない場合がある，なぜなら，**実行環境によっては**書き込めてしまう場合があるため.

**ここでの実行環境とその状態**: RMI のスケルトンと実装の双方がローカルファイルシステムからロードされると，セキュリティ制限はかからない.

# 実行環境を加えた仕様

$$LoadPlace ::= Local \mid Network$$

**LoadMap**

$$impl, skel : LoadPlace$$

**pre.MyRObject.writefile**

$$MyRObject;\ MyManager;\ LoadMap$$

---

$$impl = skel = Local \lor$$
$$(impl = Network \lor skel = Network) \land chakTable \text{ 書}$$

**MyRObject.writefile**

$$f? : \text{ファイル名};\ s? : \text{ファイル内容}$$
$$\Delta MyRObject;\ \Xi MyManager;\ \Xi LoadMap$$

---

$$serverFS'\ f? = s?;\ \mathrm{dom}\ serverFS \cup \{f?\} = \mathrm{dom}\ serverFS'$$

## 適切な性質

ローカルファイルからスケルトンと実装をロードした場合は,

$$MyRObject.write \\ \quad | \; impl = skel = Local \wedge serverFS\ f? \neq s? \\ \quad\quad \vdash serverFS \neq serverFS'$$

となり，ネットワークからどちらからかをロードした場合は,

$$MyRObject.write \\ \quad | \; skel = Network \wedge serverFS\ f? \neq s? \\ \quad\quad \vdash serverFS = serverFS'$$

となる.

# おわりに

- どのような環境が Java の RMI 部品の振る舞いに影響を与えるかを整理.
- その中で動的ロードに関わる実行環境の仕様化.
- 環境の記述を追加することで，実際の部品の振る舞いをより正確に仕様化.

  環境の違いによって動作が変ることを仕様に明示的に記述することで，環境が変化した場合の再利用の誤用を防止.

# 今後の課題

- 他の環境変化についても仕様を記述: プロキシを介したファイアウォール (ネットワーク) 環境越しの RMI の仕様化.
- 部品の誤用個所を指摘する方法の検討.
- 適切な環境下で実行しているかどうかを判断する方法の検討.
- Java に限らず計算機環境に依存するようなモーバイルコード部品の仕様化の方法を検討.

# REFERENCES

[1] Bertrand Meyer. The Next Software Breakthrough. *COMPUTER*, Vol. 30, No. 7, pp. 113–114, Jul. 1997. IEEE/CS.

[2] Jean-Marc Jezequel and Bertrand Meyer. Design by Contract: The Lessons of Ariane. *COMPUTER*, Vol. 30, No. 1, pp. 129–130, Jan. 1997.

[3] Tommy Thorn. Programming languages for mobile code. *ACM Computing Surveys*, Vol. 29, No. 3, pp. 213–239, Sep. 1997.

[4] Drew Dean, Edward W. Felten, and Dan S. Wallach. Java Security: From HotJava to Netscape and Beyond. In *Proceedings 1996 IEEE Symposium on Security and Privacy*, 1996.

尚，海谷の研究活動は，

**http://www.jaist.ac.jp/~kaiya/**

から，html, pdf, ps などの形式で参照できます.