ダウンロード可能な
ソフトウェア部品の仕様化法

海谷 治彦　海尻 賢二

信州大学 工学部

2000年 9月 25日

発表の概要

- 部品仕様の役割
- ダウンロード可能な部品とは(Java1の例より)
- **ダウンロード部品の性質**
  - コードの配置，部品ロード，セキュリティ管理
- ダウンロード部品をどのように仕様化するか？
- 本仕様の利点
  - クラッククコードを含むRMIの例を通じて.
- まとめと今後の課題
- 議論 (某査読のコメントをもとに)

部品仕様の役割

- 適切な(再)利用のためのマニュアル
- 部品の機能と制限の理解を助ける
- 開発環境等の部品ブラウザの基になる

- 自然言語による仕様:
  - 分かりやすいが長く曖昧な場合がある.
- 形式的記法
  - メリット: 小さく，矛盾なく推論などにも便利
  - デメリット: 読み書きが困難な場合が多い
  - しかし，繰り返し利用で，メリットはデメリットを超える.

伝統的な形式的機能仕様

- **シグニチャ: 引数と返り値の型・数とメソッドなどの名前 – 機能の意味は示せない**
- **メソッドの事前・事後条件**
- **クラスの普遍命題**

引数，返り値，内部状態などをもとに記述

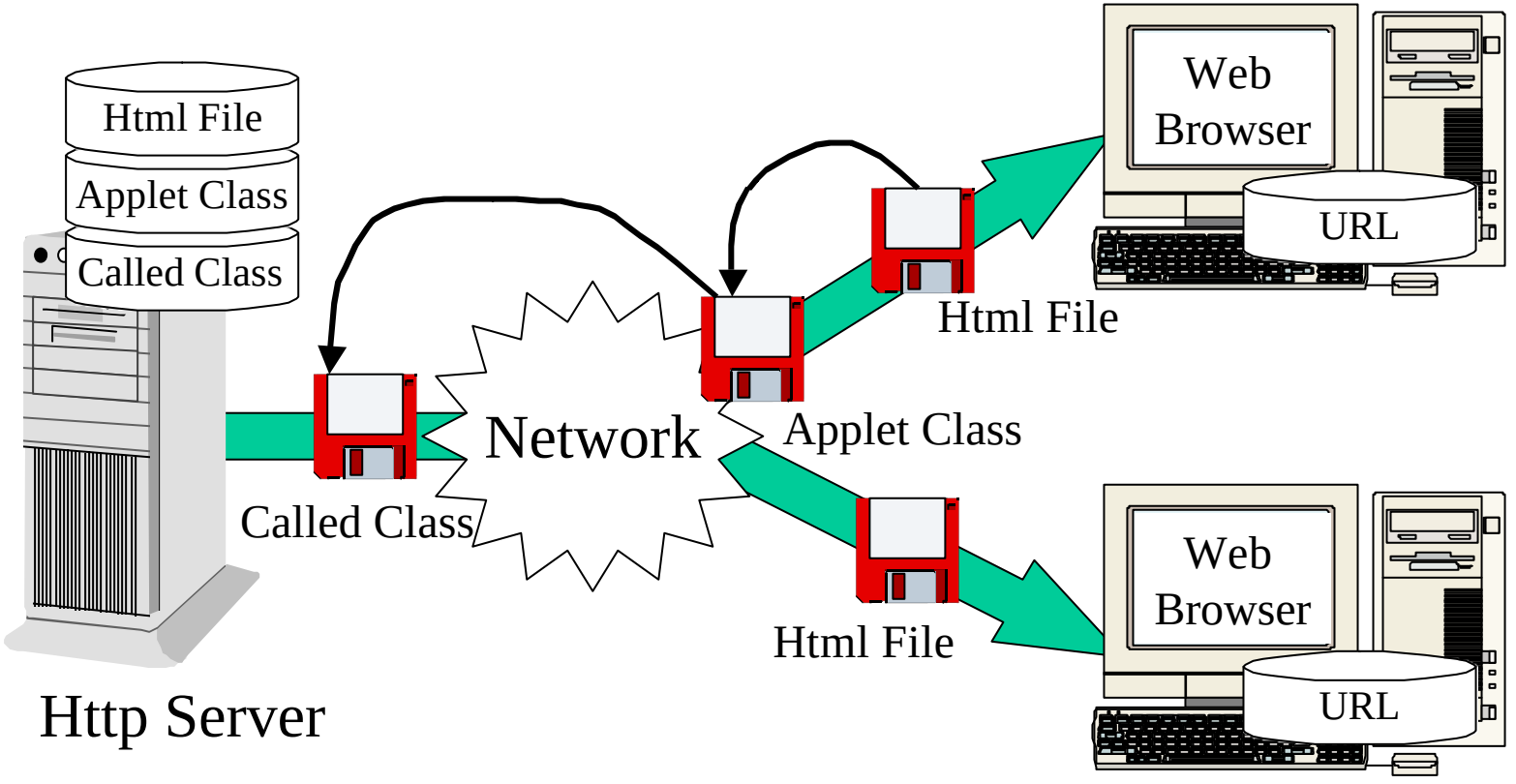
- 普通の機能部品を仕様化するには十分
- ダウンロード部品には不十分？

ダウンロード可能な部品とは？

- 自分のマシン外からロード可能
- 実行時に動的にリンク可能
- 例: JavaのアプレットやRMIのスタブ，スケルトンなど.

- 完全に信頼できる部品ではない.
- システム外のサービスや環境に大きく影響を受ける.

例: JavaのApplet



Java1ダウンロード部品のための機能仕様

- 伝統的な仕様 : シグニチャ，事前・事後条件，普遍命題

＋

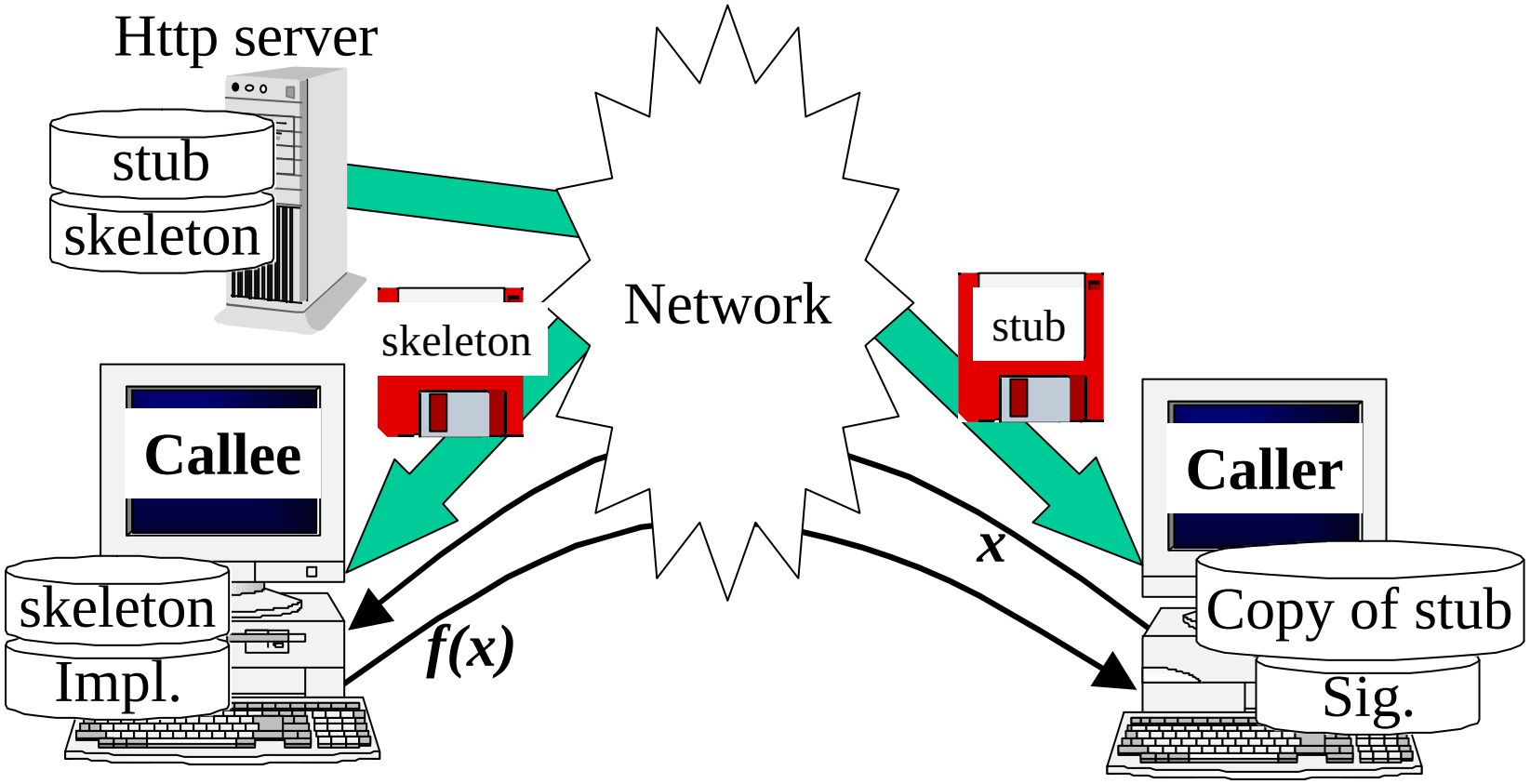
- システム外の**バイトコード**の配置
- クラスロードのポリシィ : バイトコードのサーチパス
- **セキュリティマネジャ**: システム資源へのアクセス制限チェックリスト

例を通した
仕様とその利点の説明

- 利用者の状況
  - **スタブにクラッキングコードを含むRMIを想定する.**
  - **利用者はそれが分かっているが使わざるを得ない.**
  - **しかもスタブのクラックは進行する可能性がある.**

- セキュリティマネジャによる自資源の保護
- 現在のスタブのコピーをローカルにおく

例題の概要


Http server
stub
skeleton
Network
skeleton
stub
Callee
Caller
skeleton
Impl.
f(x)
x
Copy of stub
Sig.

不十分な仕様1

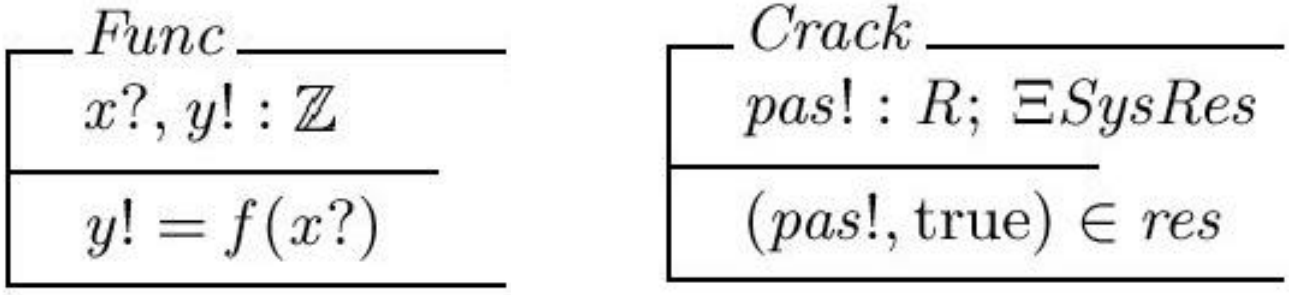
- セキュリティマネジャ以外考慮していない

$$
\begin{array}{|l}
\hline
SysRes \\
res : R \pfun Bool;\ limit : \mathbb{P}\, R \\
\hline
limit \subseteq \mathrm{dom}\, res;\ \forall x : limit \bullet (x, \mathrm{false}) \in res \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
SetLimit \\
\Delta SysRes;\ l? : \mathbb{P}\, R \\
\hline
limit \neq \emptyset \Rightarrow l? = limit' \\
\hline
\end{array}
$$

不十分な仕様2

- クラッククコードを含むメソッドの仕様



$$F \equiv (Crack \fatsemi Func \land \Xi SysRes) \setminus \{pas!\}$$

形式推論

- 以下は「セキュリティマネジャがセットされていても，クラックされてしまう！」ということを表す.

$$SetLimt \fatsemi Crack \fatsemi (Func \wedge \Xi SysRes) \mid pas! \in l?$$

- これは矛盾がある → クラックされない.
  - *(pas!*,true) と *(pas!*,false) がともに *res* の要素
  - *res* は関数

不十分な仕様に関する議論

- 実際にはこの状況下でもクラックされてしまう → 推論は現実をモデル化していない！
- コピーされたローカルなスタブにはセキュリティマネジャは無力.
- ロードは通常，ローカルから先に見る.

- バイトコードの配置が無視されている.
- クラスロードのパスが無視されている.

十分な仕様1

- バイトコードの配置

$$| \; deploy : Loc \pfun \mathbb{P}\, ByteCode$$

- 現在のマシン位置を含む *SysRes*

SysRes

$$res : R \pfun Bool;\ limit : \mathbb{P}\, R;\ here : Loc$$

$$limit \subseteq \mathrm{dom}\, res$$
$$\forall x : limit \bullet (x, \mathrm{false}) \in res$$
$$here \in \mathrm{dom}\, deploy$$

十分な仕様2

- 部品とその素(バイトコード)の関係

Class

$$birth : Loc;\ byte : ByteCode$$
$$lslctr : \text{seq}\ Loc$$

$$birth \in \text{ran}\ lslctr$$
$$birth \in \text{dom}\ deploy$$

SetLoader

$$sl?;\ \text{seq}\ Loc;\ \Delta Class$$

$$lslctr' = sl?$$
$$\forall x, y : \mathbb{N} \bullet byte \in deploy\ lslctr'\ x \land$$
$$x \in \text{dom}\ lslctr' \land lslctr'\ y = birth' \Rightarrow y \leq x$$

十分な仕様3

- クラック部分の仕様を以下のように変更

$$Crack \mathrel{\widehat=} [pas! : R;\ \Xi SysRes;\ \Xi Class \mid here \neq birth \Rightarrow (pas!, \mathrm{true}) \in res]$$

形式推論

- 以下も「セキュリティマネジャがセットされていても，クラックされてしまう！」ということを表す.

$$SetLimit \fatsemi (SetLoader \land \Xi SysRes \fatsemi Crack \fatsemi Func \land \Xi Class \land \Xi SysRes) \setminus Class$$
$$\mid pas! \in l? \land sl? = \langle here, there \rangle$$

- 以下のバイトコード配置では上式は矛盾しない.

$$deploy = \{(here, \{byte, \cdots\}), (there, \{byte, \cdots\}) \cdots\}.$$

まとめ

- Java1のダウンロード部品に合わせるように既存の機能仕様の書き方を拡張した.
- 追加した概念.
  - セキュリティポリシィと管理
  - クラスロード
  - バイトコードの配置
- 実際に部品を利用する場合の自体をよりよく反映している.

今後の課題

- Java2への拡張
  - 著名付コード (JDK1.1より)
  - Permisson とAccessController クラス
  - 拡張サンドボックスモデル
- 他のセキュリティ技術の考慮: Proof Carryingなど
- スクリプト言語の考慮: VBscript, JavaScript ….
- 言語に依存しないダウンロード部品仕様
- 同仕様をもとにしたクラス・ブラウザの開発