Specifying Downloadable Properties for Reusing Software Components: A Case Study of Java

Haruhiko Kaiya & ***Kenji Kaijiri***

Shinshu University, JAPAN

Sep. 13, 2000

Outline

- The **Role of Spec**ification for Components
- **Downloadable Components** in Java1
- **Properties** of Downloadable Components
  - code deployment, class load, security manage.
- **How to specify** Downloadable Components
- The **Advantages** of our Specification
  - An example: RMI with Cracking Code
- Conclusion and Future Works

Role of Spec. for Components

- Spec. as **Manual** for suitable reuse
- Understanding its functionality and limitations
- Base for component browser in IDE

- Natural Languages: long, vague.
- Formal Notation
  - Compact, consistent, formal reasoning
  - Hard to write
  - Retrieve the cost by using it repeatedly

Traditional Functional Spec.

- **Signatures**: type of arguments, returns and method name – No semantics!
- **Pre and Post Condition** for each method
- **Invariant** for a class

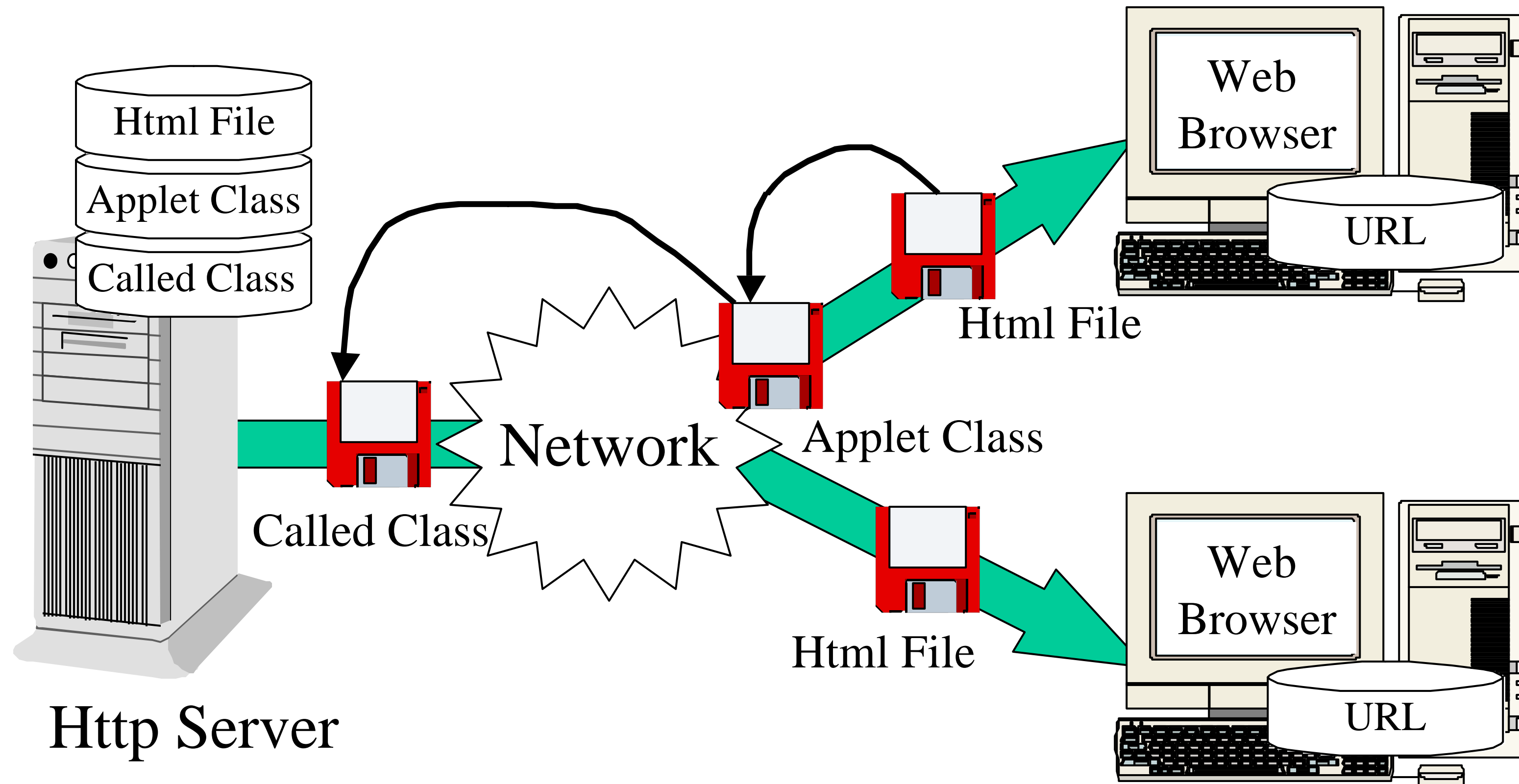
- Enough to specify normal components.
- Not enough for downloadable components

Downloadable Components

- Loaded from ***remote machines***
- ***Dynamically*** loaded and linked.
- Examples: Java Applets, RMI stub and skeleton.

- ***Not*** fully ***trusted***, i.e. in the ***sandbox***.
- Depending on the **services and environments** ***outside*** the system.

An example: Applet



Func. Spec. for D.L. Components

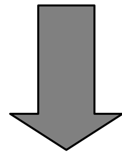
- Traditional Spec: signatures, pre/post conditions, invariants.

+

- **Deployment of bytecodes** over the network
- **Class loader's Policy**: search path for bytecodes.
- **Security Manager** for a Machine: checklist for accessing system resources.

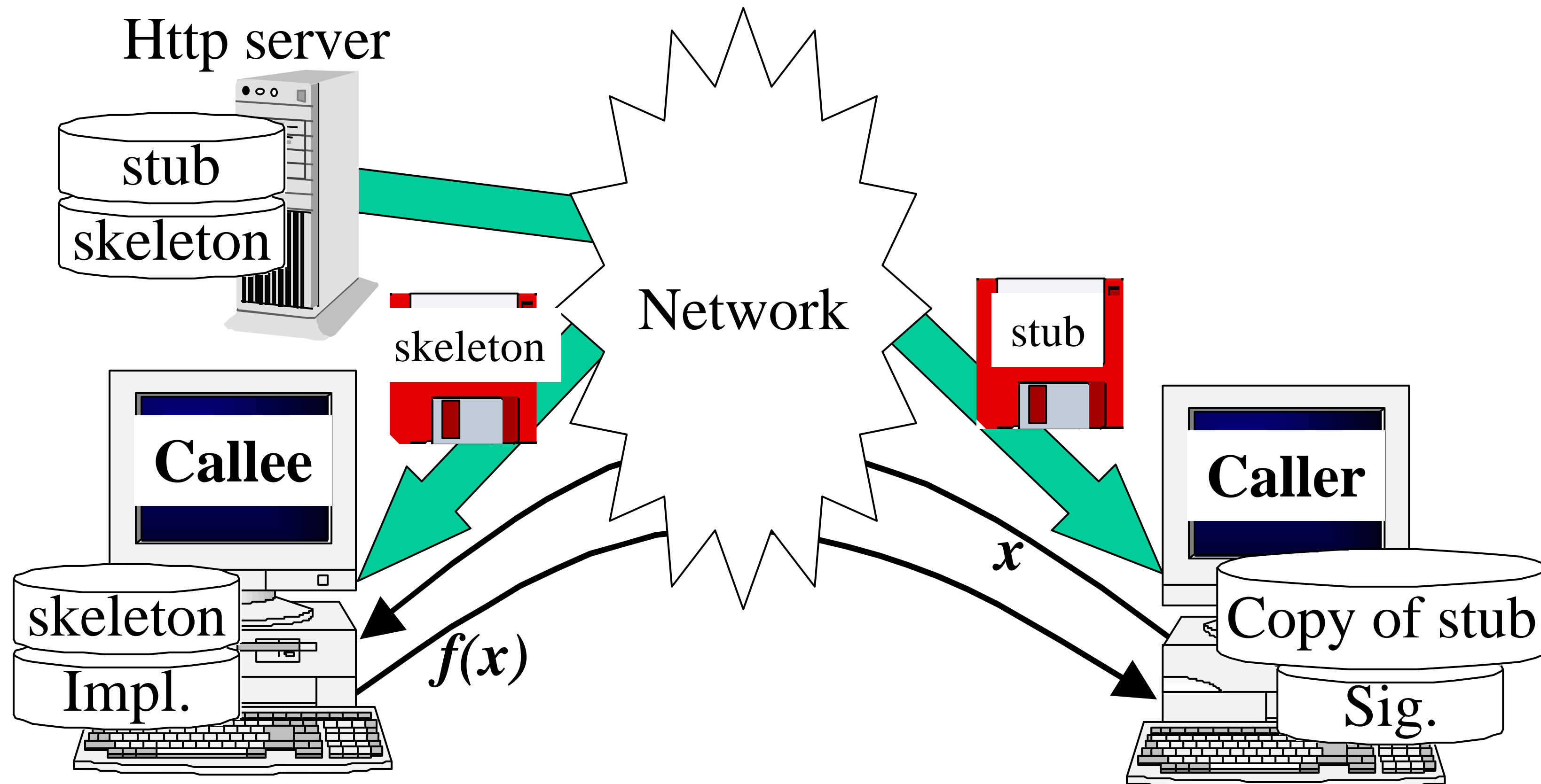
The Story of this Example

- RMI ***with cracking code*** in its stub!
- We know, but we *should use* it.
- Risk for ***the progress of cracking***.



- Set Security Manager.
- Deploying the copy of current stub in the local.

Overview of this Example



Insufficient Specification 1

- Only account for security manager.

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

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

Insufficient Specification 2

- Method body and Cracking Code

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{Func}} \\ \hline x?, y! : \mathbb{Z} \\ \hline y! = f(x?) \\ \hline \end{array}$$

$$\begin{array}{|l} \multicolumn{1}{l}{\textit{Crack}} \\ \hline pas! : R;\ \Xi SysRes \\ \hline (pas!, \mathrm{true}) \in res \\ \hline \end{array}$$

$$F \mathrel{\widehat{=}} (Crack \mathbin{;} Func \land \Xi SysRes) \setminus \{pas!\}$$

Formal Reasoning

- This schema tells ***`Cracking is established even if the security manager is set***'.

$$SetLimt \mathbin{⨟} Crack \mathbin{⨟} (Func \land \Xi SysRes) \mid pas! \in l?$$

- This is inconsistent.
  - Both *(pas!,*true*) in res* and *(pas!,*false*) in res*
  - *res* is function

Discussion for insufficient spec.

- In fact in this stage, ***Cracking is established***!
- ***Security manager is helpless*** because of the copy of ***stub*** **in the local** system.

- ***Deployment of bytecodes*** is ignored.
- ***Class loader's policy*** is also ignored.

Sufficient Specification 1

- Deployment of bytecodes

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

- *SysRes* with current location.

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

Sufficient Specification 2

- Rel. between the component and its src

Class

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

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

SetLoader

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

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

Sufficient Specification 3

- Spec. of Cracking code.

$$
\begin{aligned}
Crack \equiv [&pas! : R;\ \Xi SysRes;\ \Xi Class \mid \\
&here \neq birth \Rightarrow (pas!, rmtrue) \in res]
\end{aligned}
$$

Formal Reasoning

- This schema also tells ***`Cracking is established even if the security manager is set'***

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

- The above becomes consistent under the deployment:

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

Conclusion

- We extend traditional functional spec. for fitting downloadable components in Java1.
- Additional Concept:
  - Security Policy and Management.
  - Class loading
  - Deployment of bytecodes.
- Suitable reasoning for component use.

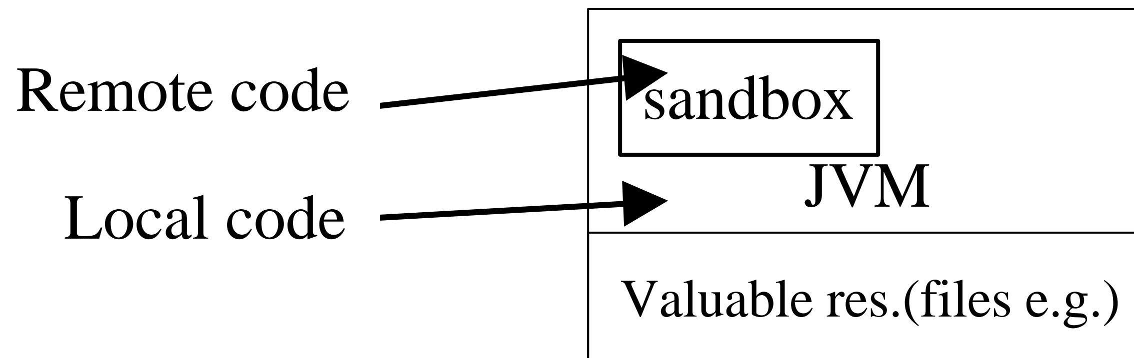
Future Works

- Extending our Spec. for Java2
  - Code signing (since JDK1.1)
  - Permisson and AccessController Class
  - Extended sandbox model
- Other Security Techniques: Proof Carrying...
- Spec. for Scripting, VBscript, JavaScript ….
- Prog. Lang. Independent Spec.
- Embedding our spec. into component browsers

Appendix

Sandbox Security Model (Java1)

- ***Local code*** is trusted to have ***full access*** to vital system resources
- While ***downloaded remote code*** is not trusted and can ***access only the limited resources*** provided inside the *sandbox*



Class Loader (Java1)

- System for loading ByteCodes into JVM
- System Class Loader:
  - From local file system
  - Always trusted.
- User Defined Class Loader:
  - From any sources, e.g. remote systems, byte streams, databases.
  - Normally untrusted.
  - Customizable by programmers.

Security Manager

- Check List for accessing system vital resources.
  - Read/write files system, create net. connection, create other class loaders……..
- Implemented as a class.
- Methods for accessing res. in API refer the lists.