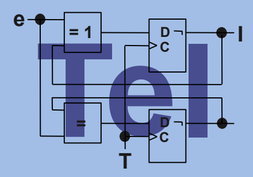
Implementierung des dynamischen Ladens von Java-Klassen für die Java-Plattform SHAP

Robert Dietrich

s1128611@mail.inf.tu-dresden.de



Institut für Technische Informatik
http://www.inf.tu-dresden.de/TeI/

Überblick

- Einleitung
- Statischer SHAP-Linker
- Dynamischer SHAP-Linker
- Test
- Ausblick

Einleitung

- Motivation
  - Agentensysteme
  - Flexibilität und Erweiterbarkeit
- Secure Hardware Agent Platform (SHAP)
  - SHAP-Mikroarchitektur – ISA basierend auf Java Bytecode
  - JVM in Microcode implementiert
  - Moderne Features wie Ausnahmeverarbeitung und automatische Garbage Collection (GC in Hardware implementiert)

Laden von Klassen

- ClassLoader - Hierarchien
  - BootClassLoader
    - Zentraler ClassLoader
    - Vertrauenswürdige Klassen (z.B. JVM System- und API-Klassen)
  - ClassLoader-Objekte
    - benutzer- bzw. applikationsspezifische ClassLoader
    - z.B. zur dynamischen Erweiterung einer Java-Applikationen zur Laufzeit

⇨ Namensräume

- Ladevorgang
  1. Übergabe der vollständigen Bezeichnung des Typs, Einlesen einer Binärdatenfolge (z.B. Klassendatei)
  2. Parsen der Binärdatenfolge in die interne Datenstruktur
  3. Bilden einer Instanz der Klasse java.lang.Class

Linken

- Verifikation
  - Java Bytecode
  - Semantik der Sprache Java
  - 4 Durchgänge
    1. grundlegendes Format
    2. grundlegende Konzepte der Java-Objektorientierung
    3. Bytecode-Verifizierung (Datenflussanalyse)
    4. Verifizierung symbolischer Referenzen

- Vorbereitung
  - Übertragung der `Class`-Objekte in implementierungsspezifische Datenstruktur
  - Speicher für Klassenvariablen reservieren und Initialwerte setzen
  - zusätzlichen Speicher für Datenstrukturen reservieren

Linken

- Resolution
  - Auflösen symbolischer Referenzen
  - vor erstem Zugriff auf entsprechende Referenz
    - Erstellen einer neuen Klasseninstanz
    - Aufrufen einer statischen Methode einer Klasse
    - Zugreifen auf ein nicht konstantes statisches Feld einer Klasse
    - Initialisieren einer Unterklasse einer Klasse
  - Early Resolution
    - im Anschluss an das Laden der Klasse
  - Late Resolution
    - Direkt vor erstem Zugriff auf symbolische Referenz

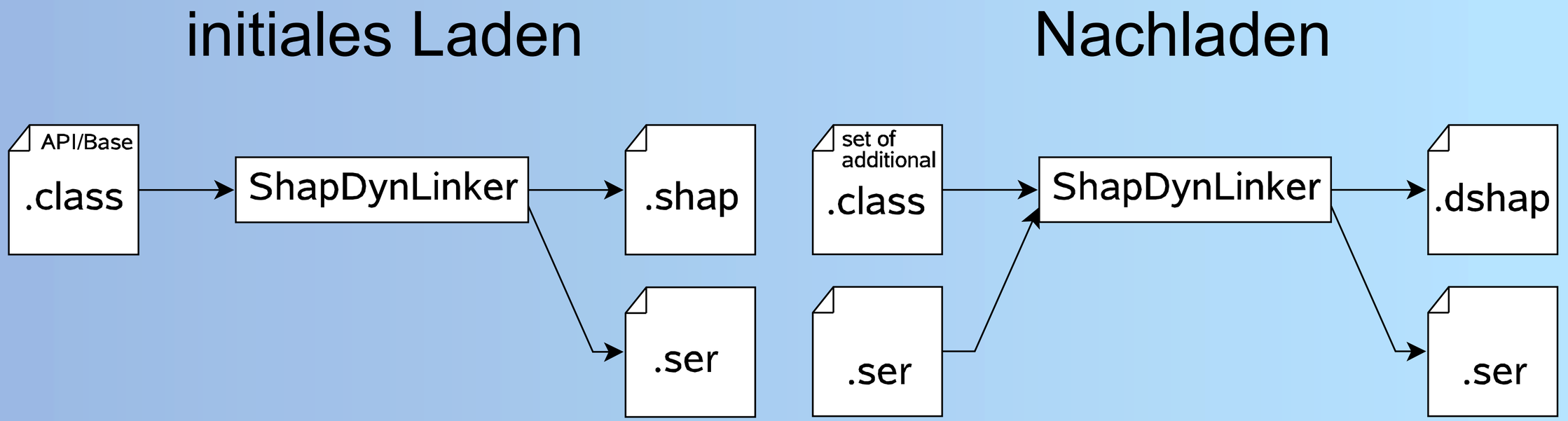
Statischer SHAP-Linker

- **Input:** komplette Anwendung inkl. System- und benötigten API-Klassen
- **Output:** SHAP-Datei

- Klassen: `ShapLinker`, `ShapProject`, `ShapClass`

- ASM-Framework
  - Bytecodemanipulation
  - Vorbereitungsphase
  - Verifikation

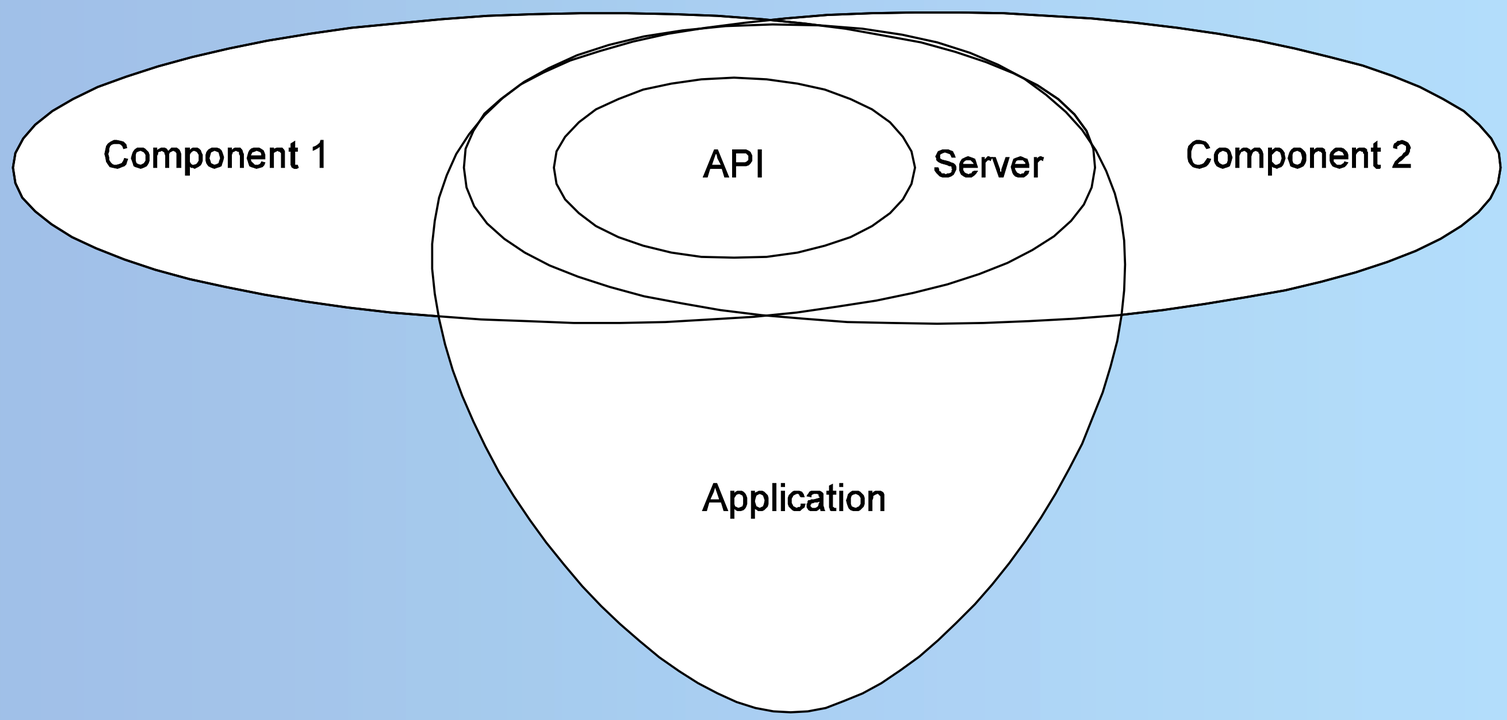
Dynamischer SHAP-Linker



- ⇨ Persistenz von Projektdaten
- ⇨ Erweiterung der Verifikation

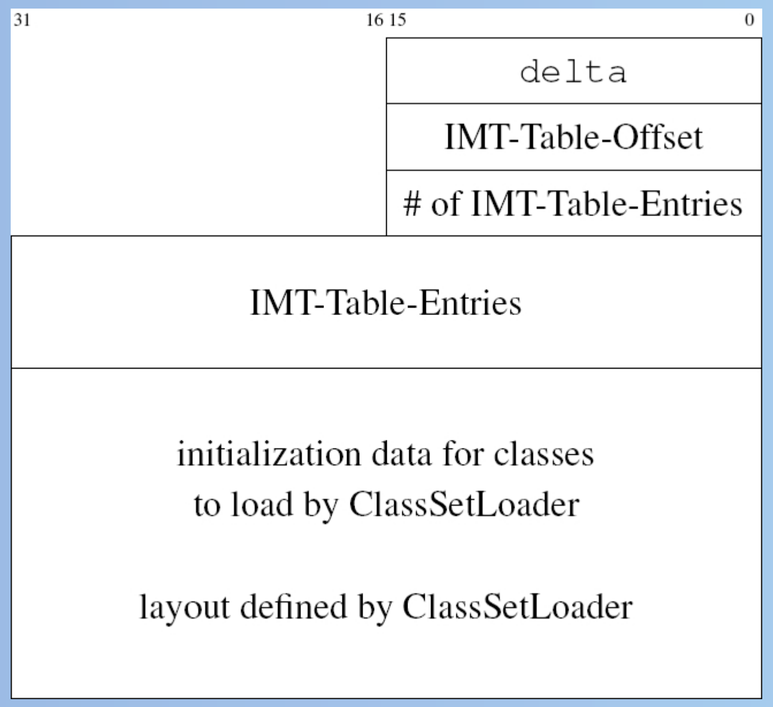
Entwurfsentscheidungen bzgl. des dyn. Nachladens

- Laden von kompletten Klassensätzen
- Early Resolution

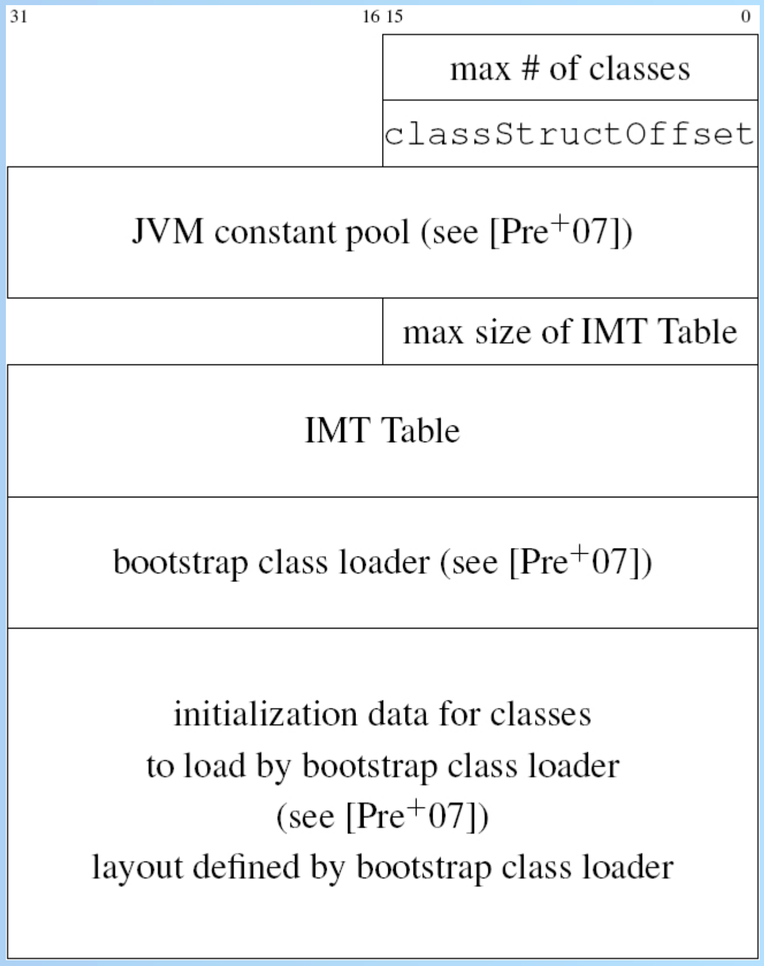


SHAP Dateien & ClassSetLoader

Differentielle SHAP-Datei

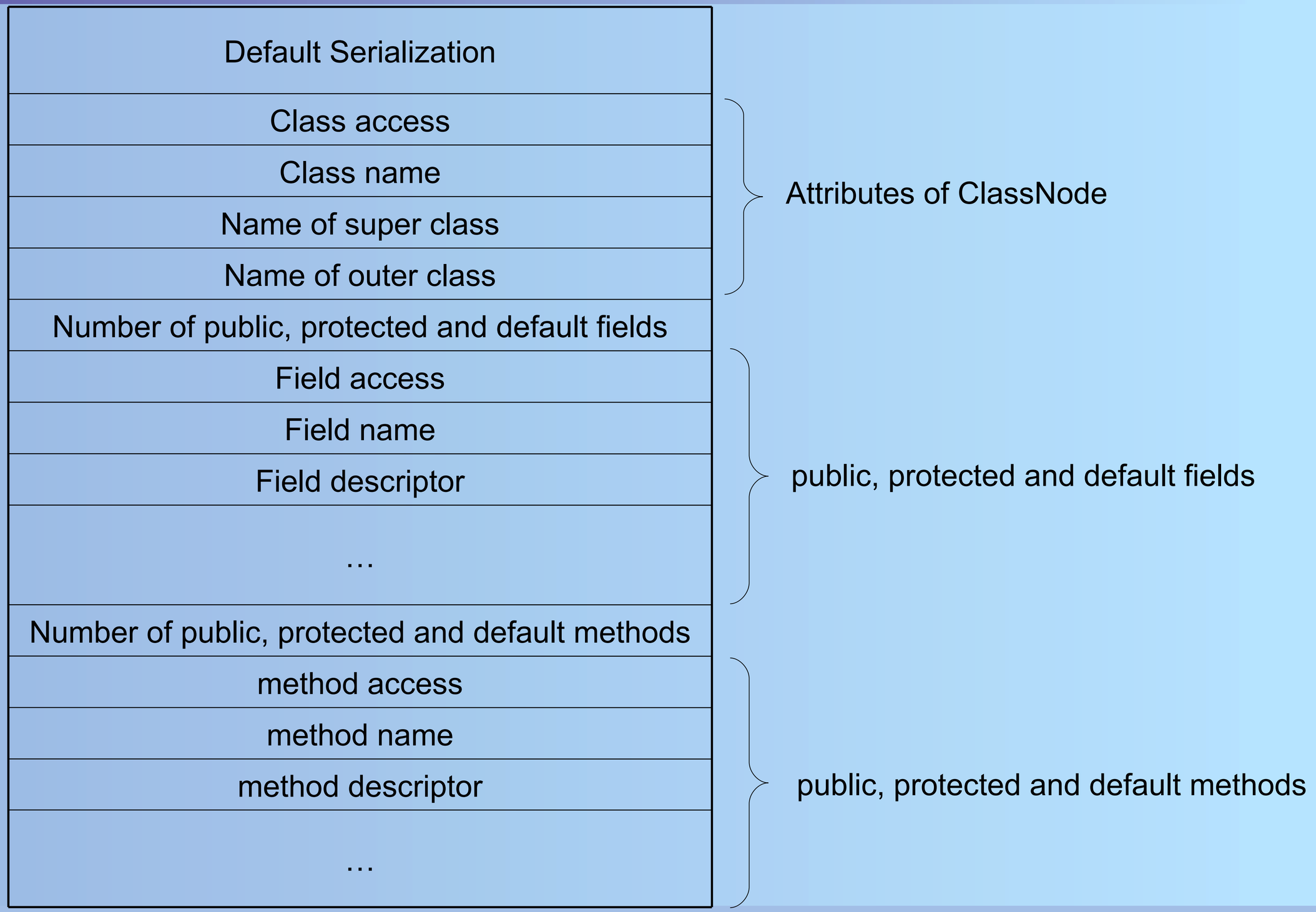


Initiale SHAP-Datei



- ClassSetLoader
  - Teil der SHAP-JVM
  - Liest .dshap-Dateien ein
  - Schreibt eingelesene Klassen in vorhandene Klassenstruktur
  - Schreibt IMTLookup-Table Einträge in vorhandene Tabelle (zukünftig)

Serialisierung von ShapClasses



Verifikation

Änderung von Zugriffmodifizierern

	Innerhalb Klasse	Paket-Klassen	Unterklassen	Sonstige Klassen
private ⇨ default		Exception	Compiler	Compiler
private ⇨ protected		Exception	Exception	Compiler
private ⇨ public		Exception	Exception	Exception
default ⇨ protected			Exception	Compiler
default ⇨ public			Exception	Exception
protected ⇨ public				Exception

Betrachtung von:

- Klassen
- Feldern (Attributen)
- Methoden

	Innerhalb Klasse	Paket-Klassen	Unterklasse	Sonstige Klassen
private	✓	-	-	-
default	✓	✓	-	-
protected	✓	✓	✓	-
public	✓	✓	✓	✓

=> ShapClassVerifier

ShapClassVerifier

- Durchsucht nachzuladende Klassen nach Referenzen auf bereits geladene Klassen

- Überprüfung gefundener Referenzen
  1. *public* -> korrekt
  2. Beide Klassen im selben Paket - > korrekt (*default*)
  3. Nachzuladende Klasse ist Unterklasse bereits geladener und Modifier ist *protected*
  4. Keine Bedingung trifft zu -> Exception

Testanwendung für

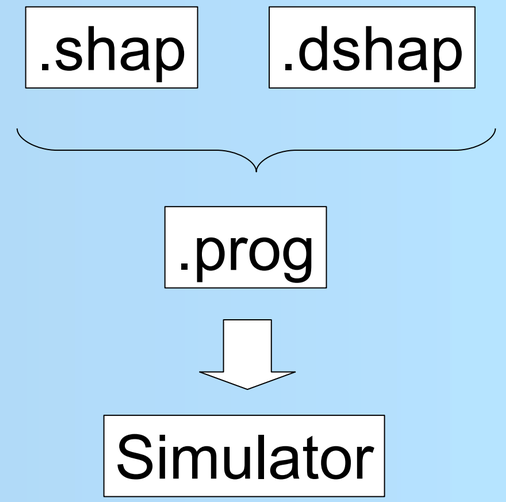
- Tests zum dynamischen Nachladen
  - Interfaces
  - Vererbungsstrukturen
  - Nachladen bereits geladener Klasse
  - …

- Test des ShapClassVerifiers
  1. Laden und Linken einer Klasse
  2. Änderung eines Modifiers der geladenen Klasse
  3. Nachladen einer Klasse, welche auf geänderte Struktur zugreift
  4. ggf. Fehlerauswertung

Test (2)

- Testumgebung: ndoprosim (Prozessorsimulator)

- Testablauf
  1. Erzeugung der .shap-Datei
  2. Erzeugung der .dshap-Datei(en)
  3. Zusammenfassen zu .prog-Datei
  4. Simulation der .prog-Datei mit ndoprosim



Ausblick

- IMTLookup-Table der SHAP-JVM
- Erweiterung des ShapClassVerifiers
- Optimierung der Serialisierung

- Bedienbarkeit - Frontend

Literatur

[Brun07] BRUNETON, E.: ASM 3.0 - A Java bytecode engineering library, 2007. – http://asm.objectweb.org

[LiYe99] LINDHOLM, T.; YELLIN, F.: The Java Virtual Machine Specification. Second Edition. Prentice Hall PTR, 1999. – http://java.sun.com/docs/books/vmspec/

[Pre+07] PREUSSER, T. B.; ZABEL, M.; REICHEL, P.: The SHAP Microarchitecture and Java Virtual Machine. Fakultät Informatik, Technische Universität Dresden, Forschungsbericht TUD-FI07-02, 2007. – ISSN 1430–211X

[Preu03] PREUSSER, T. B.: Entwicklung eines Prozessorsimulators unter besonderer Berücksichtigung des Organic Computing, 2003