Laboratoire Bordelais de
Recherche en Informatique

Équipe : Systèmes et
Objets Distribués

JCAT
An environment for attack and test on Java Card™

Serge Chaumette, Iban Hatchondo, Damien Sauveron

Damien Sauveron
sauveron@labri.fr
http:/www.labri.fr/~sauveron/

Plan

1) The Java Card Security project
   - ➤ Context of our work
   - ➤ Partners
2) Java Card
   - ➤ What is the Java Card Technology?
   - ➤ Applet Development
   - ➤ Architecture
3) JCAT Tools
   - ➤ Overview
   - ➤ Why developping JCAT Tools?
   - ➤ JCAT Emulator
4) Problems in the specifications
   - ➤ Problem of the heap
5) Conclusion and future work

Context of our work

In 2001, there was no evaluation methodology following Common Criteria (ISO 15408) for:

- the Java Card products;
- the applications running on these products.

The French government wanted to improve the security assurance level of theses new IT products and has accepted our Java Card Security project.

Partners (1/2)

The ***Distributed Systems and Objects*** team of the *Laboratoire Bordelais de Recherche en Informatique* (Bordeaux, FRANCE) provides software tools:

- to easily and securely develop applications based on distributed and mobile code;
- to use theses applications;
- that are proven to do what they pretend to.

Partners (2/2)

The ***ITSEF (Information Technology Security Evaluation Facility) center*** of *SERMA Technologies* (Pessac, FRANCE):

- is specialized in the ITSEC & Common Criteria security evaluation of smart card products and especially Java Cards.

The French gouvernment provides:

- from the French Ministry of Industry the label *Société de l'information* and the funding for the project *Sécurité Java Card*;
- from the French Ministry of Research a part of the funding for a doctoral grant.

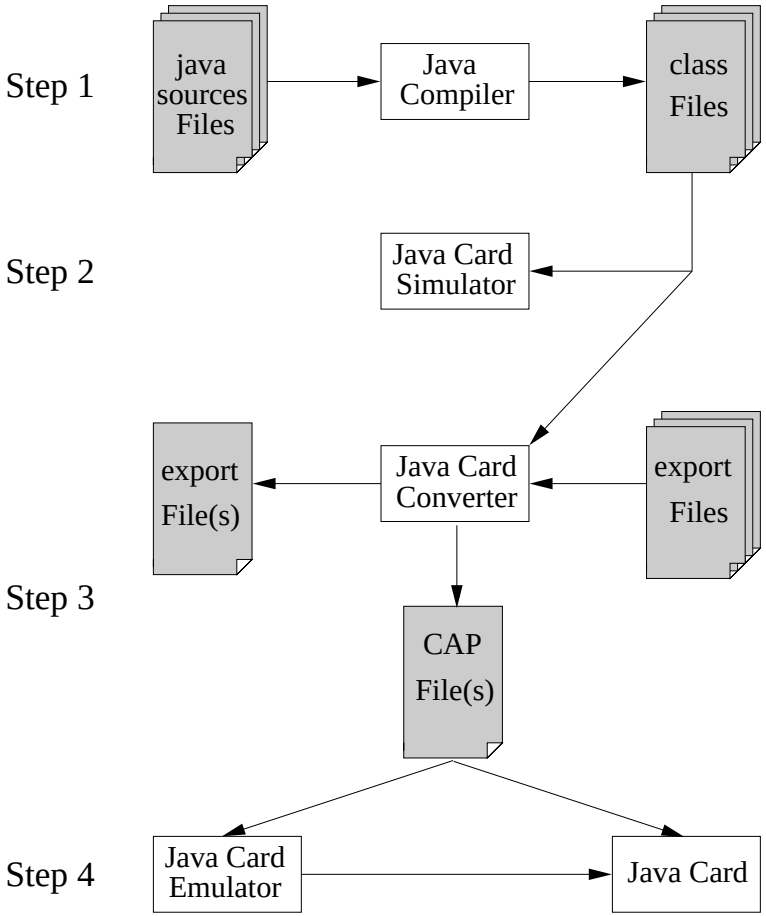
What is the Java Card Technology?

This technology enables programs written in Java programming language to run on:

- smart cards;
- other resource-constrained devices.

Java Card Technology provides smart cards with a secured, hardware independant and multi-application framework that includes many assets of the Java programming language.

Applet Development

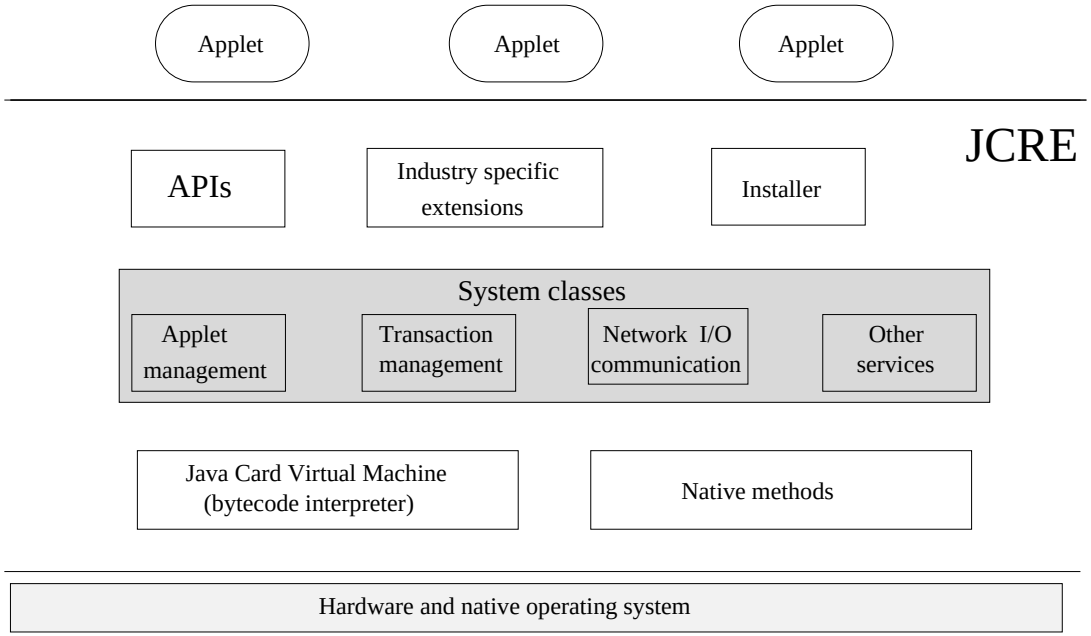


Architecture

Java Card is a ***super-set*** of a ***subset*** of the Java programming language.

Main additional features:

- atomicity;
- transaction;
- the firewall.



Overview

JCAT Emulator:
Tool to attack and test applets implementation.

JCAT View:
Tool that enables to parse the CAP file format of different manufacturers to view it (*e.g.* IBM, Oberthur Card Systems, Sun microsystems, etc.).

JCAT Converter:
Tool to convert a CAP file format from a manufacturer to an other.

Other JCAT Tools: for instance, a tool to help to modify the bytecode
$\Longrightarrow$ enable an easy creation of test and attack suites.

Developed in ***Java***.

Why developping the JCAT Tools? (1/2)

Java Card implementations are mainly commercial and closed solutions.
$\Longrightarrow$ low-level details of effective implementation are kept confidential.

Java Card simulators provided in toolkits are incomplete (no transaction, no firewall, etc.).

Security certification requires:

- the validation of Virtual Machine;
- the validation of parts of resident applets.

$\Longrightarrow$ Needs are contextual.

Why developping the JCAT Tools? (2/2)

Our goals:

- Doing a complete and open source emulator tool.
  $\Longrightarrow$ To be adaptable to different contexts;
- Enable both hardware and software attacks simulation.
  $\Longrightarrow$ To be as closed as possible to an embedded implementation;
  For instance an electromagnetic radiation just modifies the contents of the smart card memory cells, thus modifies the value of some system object.
- Get a tool allowing the usage of formal tools as plugins.

JCAT Emulator (1/3)

Complete implementation of the specifications Java Card 2.1.1 :

- VM and APIs
- JCRE (firewall and transaction)

Goal:

- To test and debug Java Card applets;
- To detect some problems of behaviour.

JCAT Emulator (2/3)

Features:

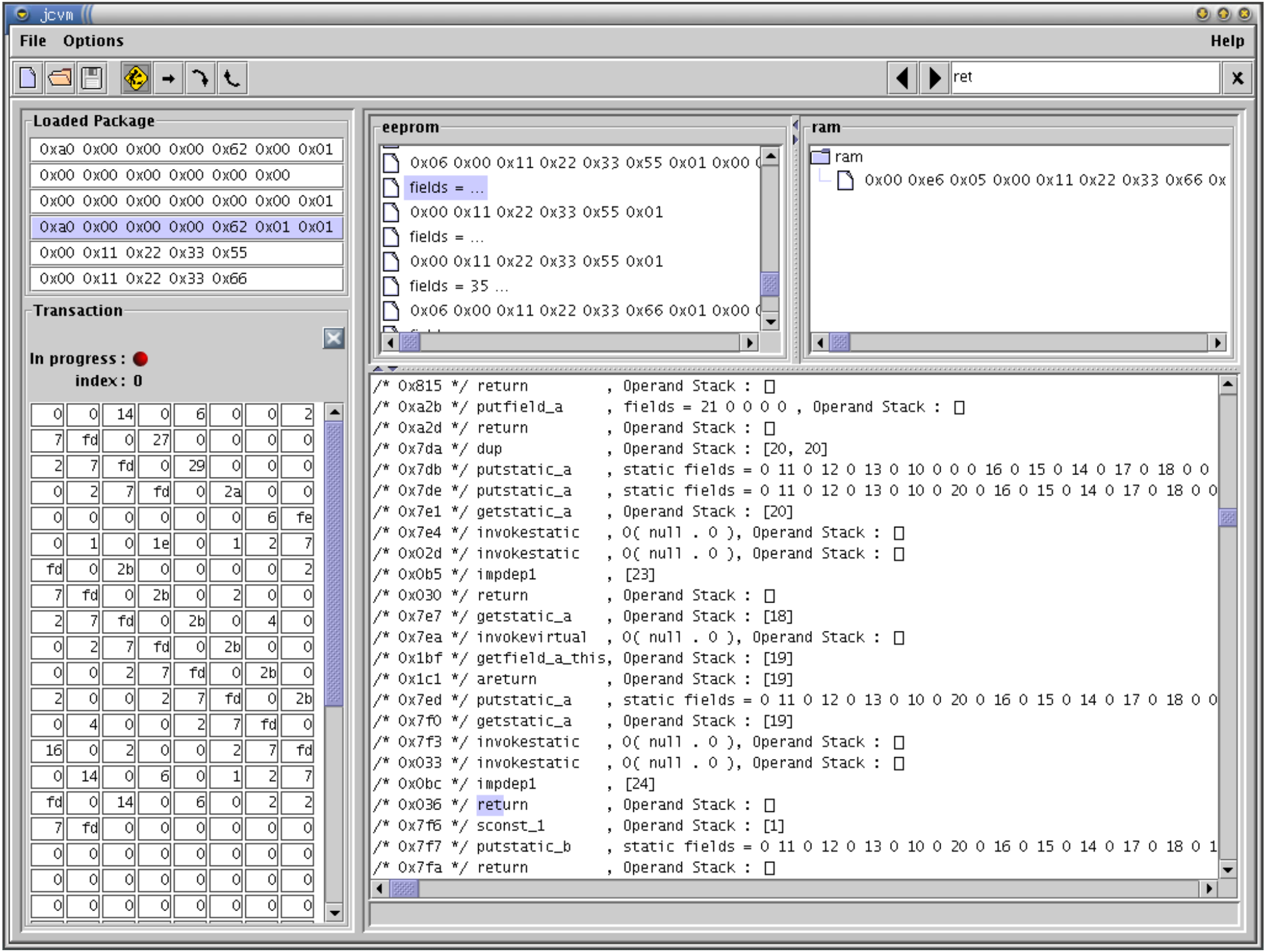
- execution step by step;
- view of the memories, objects, transaction buffer, frame stack, etc.;
- provides statistics on the execution;
- performs tearing and laser attacks;
- choices of memory size;
- available from PC/SC.

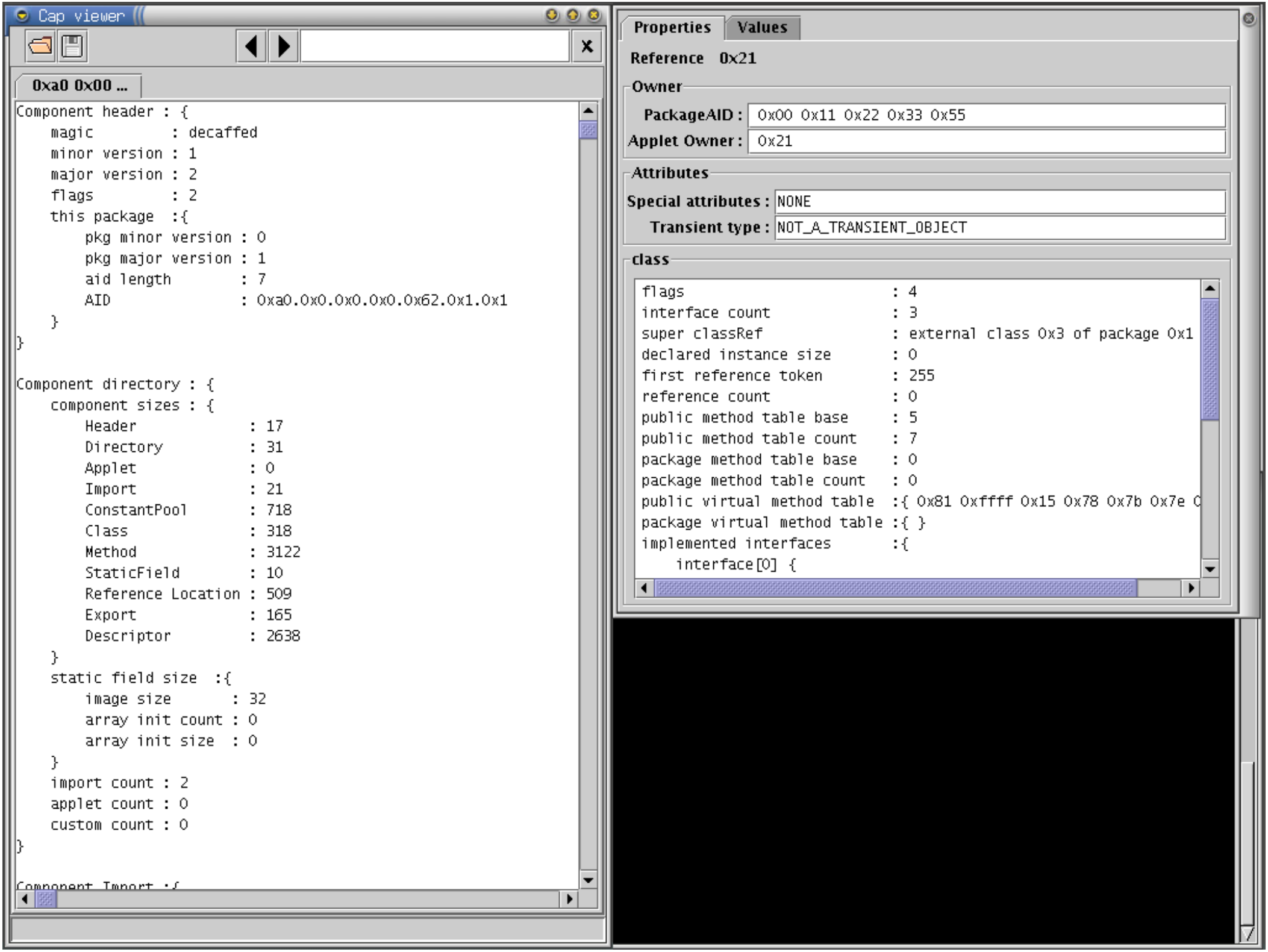
JCAT Emulator (3/3)

Towards an evaluation methodology for a Java Card platform:
Listing of the important security points in the specifications that are obscure.
Example : What is the behaviour of the platform when a transaction is aborted in `install()` after a call to `register()`.

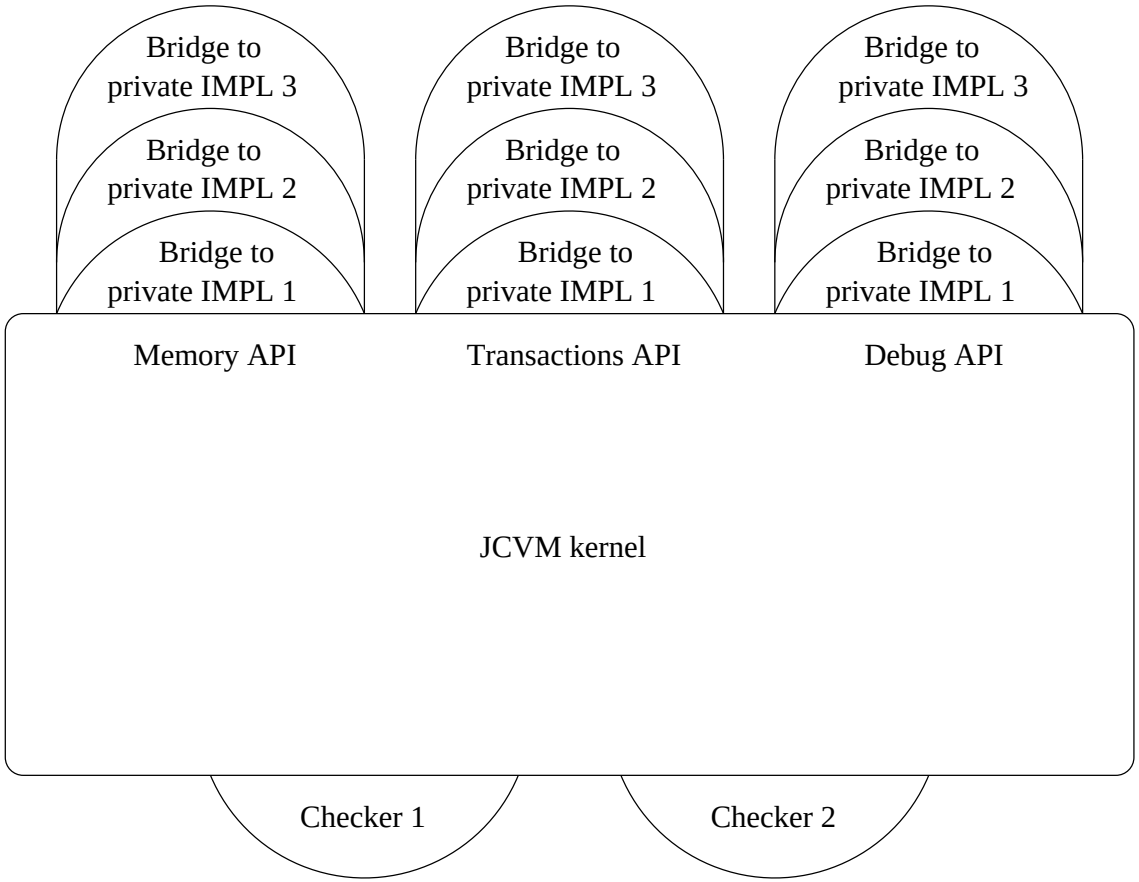
To do:

- improve the statistic reports;
- improve the simulation of physical attacks;
- improve the implementation of crypto APIs;
- add the support of Java Card 2.2 (RMI, multi-channels) and Open Platform.


jcvm
File Options Help
ret
Loaded Package
0xa0 0x00 0x00 0x00 0x62 0x00 0x01
0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x01
0xa0 0x00 0x00 0x00 0x62 0x01 0x01
0x00 0x11 0x22 0x33 0x55
0x00 0x11 0x22 0x33 0x66
Transaction
In progress :
index : 0
0 0 14 0 6 0 0 2
7 fd 0 27 0 0 0 0
2 7 fd 0 29 0 0 0
0 2 7 fd 0 2a 0 0
0 0 0 0 0 0 6 fe
0 1 0 1e 0 1 2 7
fd 0 2b 0 0 0 0 2
7 fd 0 2b 0 2 0 0
2 7 fd 0 2b 0 4 0
0 2 7 fd 0 2b 0 0
0 0 2 7 fd 0 2b 0
2 0 0 2 7 fd 0 2b
0 4 0 0 2 7 fd 0
16 0 2 0 0 2 7 fd
0 14 0 6 0 1 2 7
fd 0 14 0 6 0 2 2
7 fd 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
eeprom
0x06 0x00 0x11 0x22 0x33 0x55 0x01 0x00
fields = ...
0x00 0x11 0x22 0x33 0x55 0x01
fields = ...
0x00 0x11 0x22 0x33 0x55 0x01
fields = 35 ...
0x06 0x00 0x11 0x22 0x33 0x66 0x01 0x00
ram
ram
0x00 0xe6 0x05 0x00 0x11 0x22 0x33 0x66 0x
/* 0x815 */ return , Operand Stack : []
/* 0xa2b */ putfield_a , fields = 21 0 0 0 0 , Operand Stack : []
/* 0xa2d */ return , Operand Stack : []
/* 0x7da */ dup , Operand Stack : [20, 20]
/* 0x7db */ putstatic_a , static fields = 0 11 0 12 0 13 0 10 0 0 0 16 0 15 0 14 0 17 0 18 0 0
/* 0x7de */ putstatic_a , static fields = 0 11 0 12 0 13 0 10 0 20 0 16 0 15 0 14 0 17 0 18 0 0
/* 0x7e1 */ getstatic_a , Operand Stack : [20]
/* 0x7e4 */ invokestatic , 0(null . 0), Operand Stack : []
/* 0x02d */ invokestatic , 0(null . 0), Operand Stack : []
/* 0x0b5 */ impdep1 , [23]
/* 0x030 */ return , Operand Stack : []
/* 0x7e7 */ getstatic_a , Operand Stack : [18]
/* 0x7ea */ invokevirtual , 0(null . 0), Operand Stack : []
/* 0x1bf */ getfield_a_this, Operand Stack : [19]
/* 0x1c1 */ areturn , Operand Stack : [19]
/* 0x7ed */ putstatic_a , static fields = 0 11 0 12 0 13 0 10 0 20 0 16 0 15 0 14 0 17 0 18 0 0
/* 0x7f0 */ getstatic_a , Operand Stack : [19]
/* 0x7f3 */ invokestatic , 0(null . 0), Operand Stack : []
/* 0x033 */ invokestatic , 0(null . 0), Operand Stack : []
/* 0x0bc */ impdep1 , [24]
/* 0x036 */ return , Operand Stack : []
/* 0x7f6 */ sconst_1 , Operand Stack : [1]
/* 0x7f7 */ putstatic_b , static fields = 0 11 0 12 0 13 0 10 0 20 0 16 0 15 0 14 0 17 0 18 0 1
/* 0x7fa */ return , Operand Stack : []


Cap viewer
0xa0 0x00 ...
Component header : {
magic : decaffed
minor version : 1
major version : 2
flags : 2
this package :{
pkg minor version : 0
pkg major version : 1
aid length : 7
AID : 0xa0.0x0.0x0.0x0.0x0.0x62.0x1.0x1
}
}
Component directory : {
component sizes : {
Header : 17
Directory : 31
Applet : 0
Import : 21
ConstantPool : 718
Class : 318
Method : 3122
StaticField : 10
Reference Location : 509
Export : 165
Descriptor : 2638
}
static field size :{
image size : 32
array init count : 0
array init size : 0
}
import count : 2
applet count : 0
custom count : 0
}
Component Import :{
Properties
Values
Reference 0x21
Owner
PackageAID : 0x00 0x11 0x22 0x33 0x55
Applet Owner : 0x21
Attributes
Special attributes : NONE
Transient type : NOT_A_TRANSIENT_OBJECT
class
flags : 4
interface count : 3
super classRef : external class 0x3 of package 0x1
declared instance size : 0
first reference token : 255
reference count : 0
public method table base : 5
public method table count : 7
package method table base : 0
package method table count : 0
public virtual method table :{ 0x81 0xffff 0x15 0x78 0x7b 0x7e
package virtual method table :{ }
implemented interfaces :{
interface[0] {

JCAT Emulator Kernel



Damien Sauveron
sauveron@labri.fr

JCAT Emulator extensibility

Factory design patterns is used several times:
For instance, the CAP file reader uses it to be extensible. Indeed, in the next specifications, the CAP file may be augmented with new components and it is an easy way to support them.

Native interfaces:
Theses calls are forbidden for the end-user but needed for the Java Card APIs implementors (i.e. Transaction).
The specifications do not impose a specific way to deal with native interfaces.
Choices: Using the *impdep1* private bytecode.

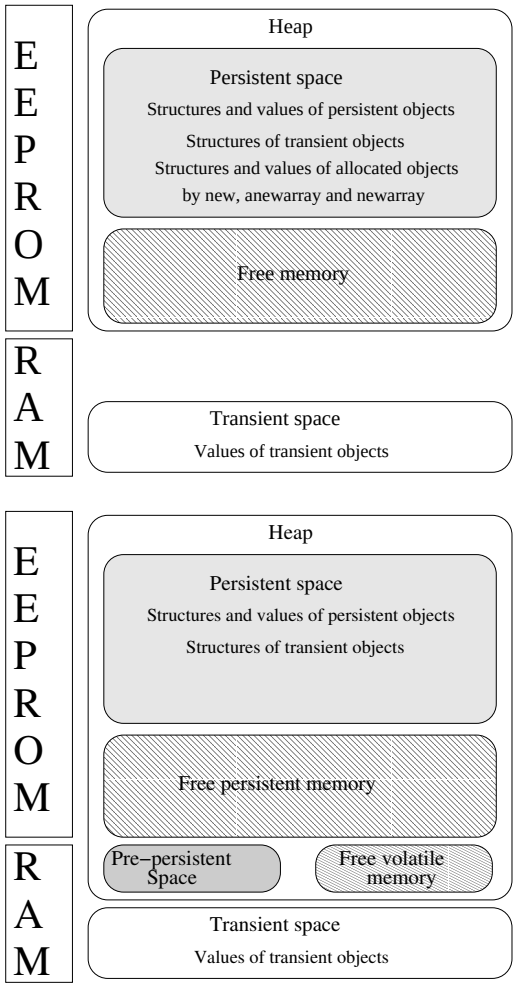
JCAT Emulator adaptability

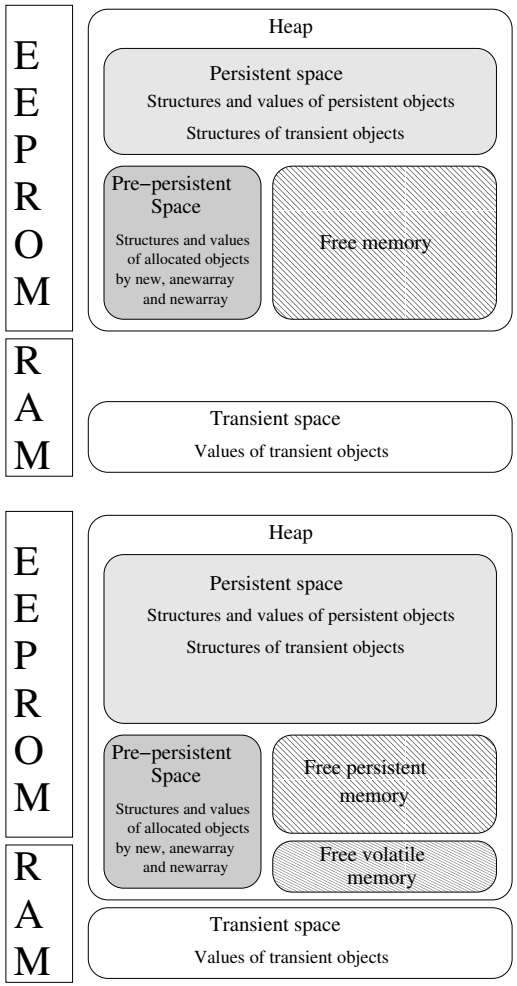
In the Java Card specification, we have discovered some unclear parts regarding:

- the persistence of objects;
- the heap location.

Thus we have designed the JCAT Emulator to be adaptable to the choice that will be done by the Java Card Forum.

Problem of the heap





Conclusion and future work

JCAT Emulator is designed as a flexible an open architecture.

It allows the suppport of external plugins to solve the problem of confidentiality of the majors of the smart card industry.

- Gemplus
- Oberthur Card Systems
- SchlumbergerSema
- SERMA Technologies
- Sun microsystems

Informal contacts:

- Datacard
- Giesecke & Devrient
- IBM

Certifying our JCVM by Sun microsystems.