Smart Card Modeling

Gemplus Research Lab

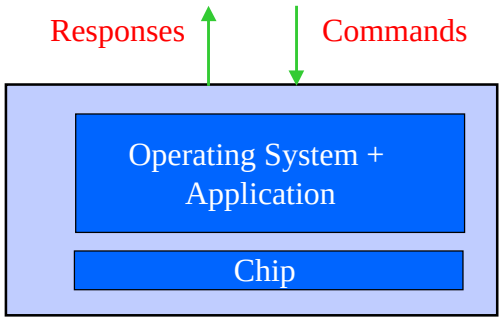
Saint Malo, 8-9 July 1999
jean-louis.lanet@gemplus.com

Outline

- Motivations
- The B Method
- Java Card Mechanisms:
  - Verifier
  - Interpreter
  - Firewall
- Conclusions

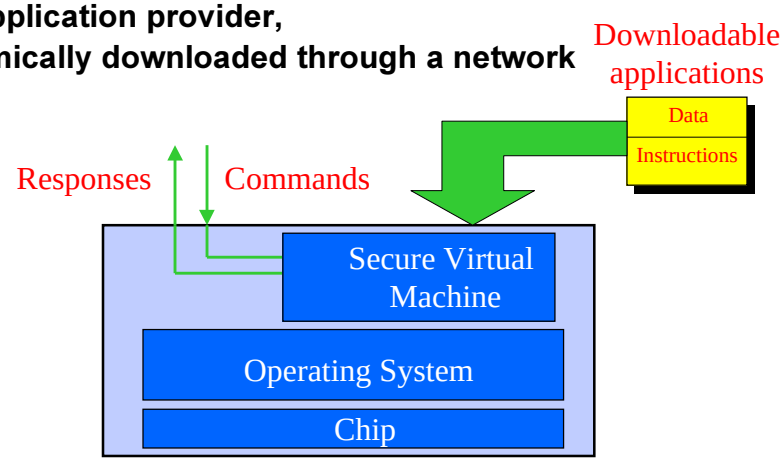
Motivations

- Applications are developed by the card provider in a secure environment,
- Drawbacks:
  - time consuming
  - costly



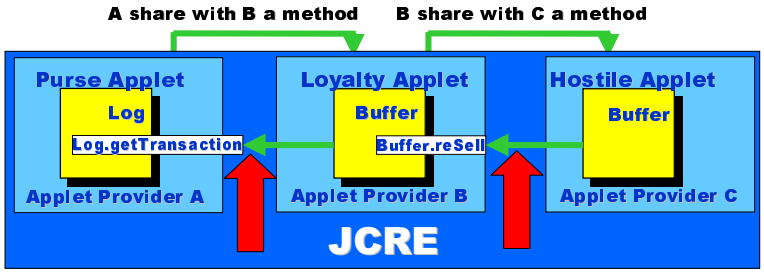
Open cards...

- Applications developed by the customer or any application provider,
- Dynamically downloaded through a network



...and the sharing mechanism

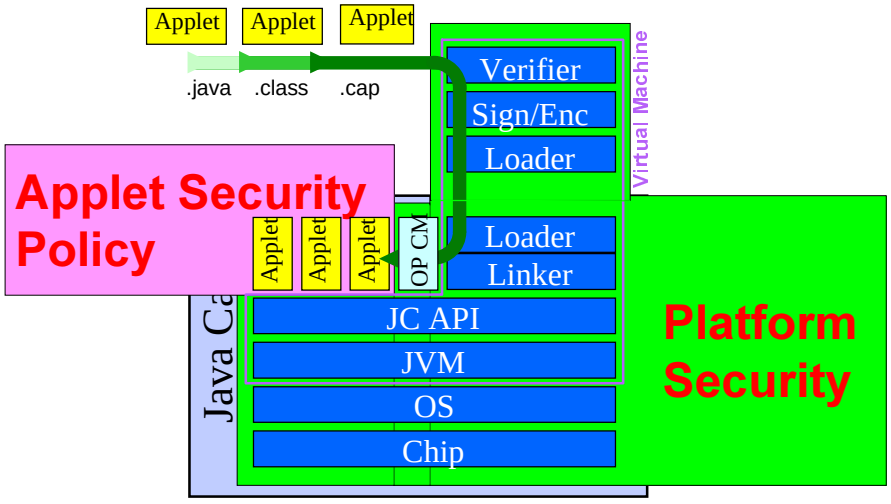
- The Java Card specification provides a mechanism to share data between several applets,
  - For example: a purse and a loyalty applet can share methods and/or objects,
  - Due to the limited resources of the smart cards new services or libraries will be offered.



New security problems

- Applications are no more developed under card issuer control,
  - Naïve implementation can ease DPA attacks,
  - Any application provider can introduce a Trojan Horse in the card,
- New attacks can arise (denial of services…),
  - Information can be exchanged between application,
  - Use faulty platform implementation

Java Card Security Chain



Two security levels

- Platform security
  - Traditional means,
  - Use of formal methods.

  => Models of the platform security modules
- Application security
  - There is a need for a global security policy
  - Flow control (data and/or code sharing)
  - Resources consumption (memory, CPU, method calls...)

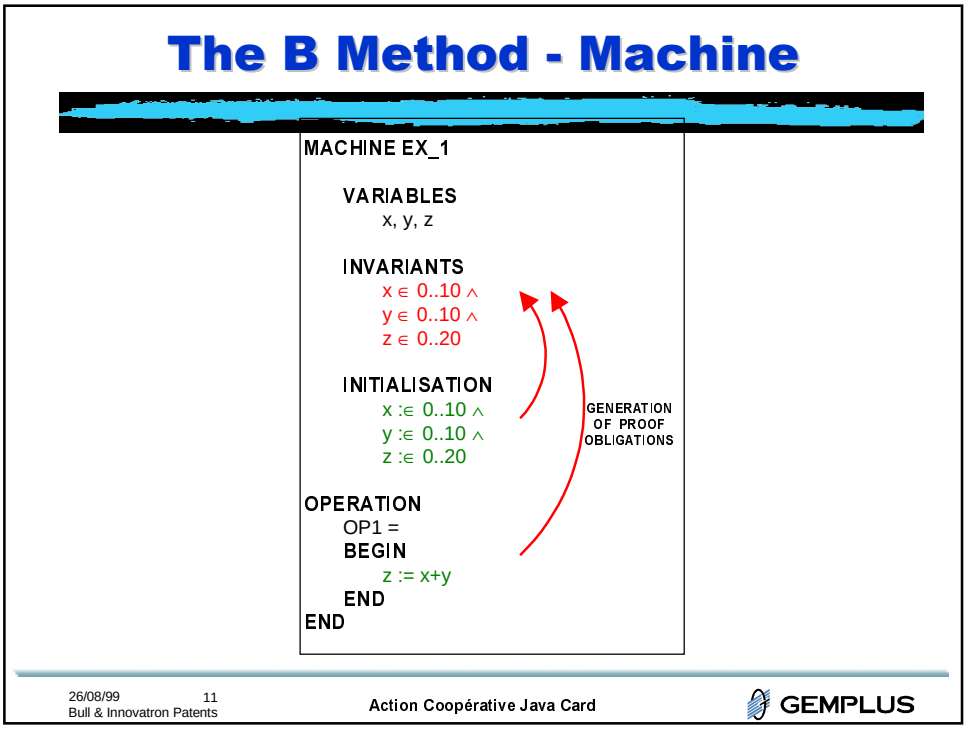
  => Static analysis of applet configurations (part of the CMS)

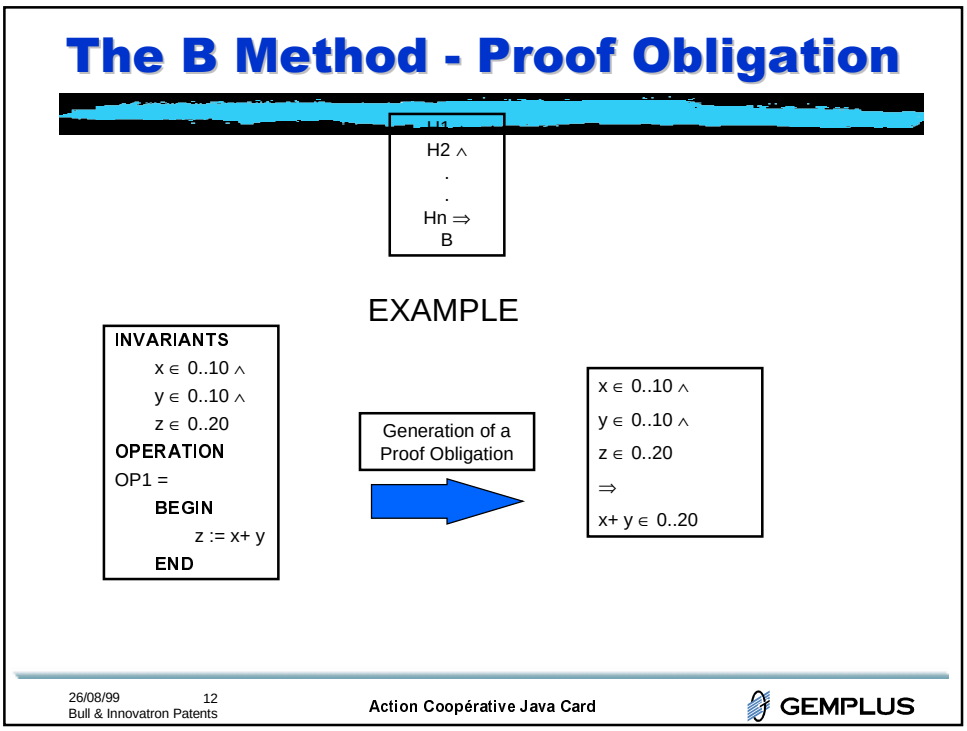
Smart Card Modeling

- B Method
- Verifier
- Interpreter
- Firewall

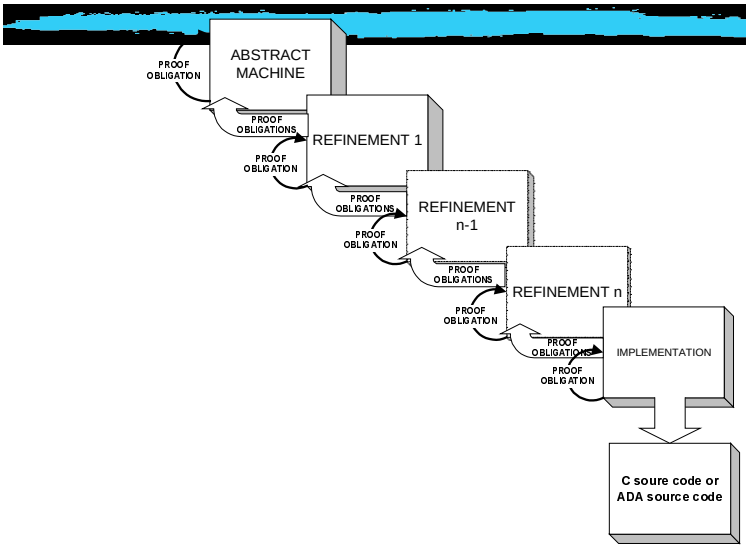
The B method

- A formal Method
  - Based on the mathematical set theory (variables, sets, relations, etc..),
  - Generation of proof obligations,
  - Theorem prover
- Supported by CASE tools (AtelierB, B Toolkit..)
- Used in industrial applications (RATP Meteor automatic subway, SNCF TGV Speed train control


The B Method - Machine
MACHINE EX_1
VARIABLES
x, y, z
INVARIANTS
x ∈ 0..10 ∧
y ∈ 0..10 ∧
z ∈ 0..20
INITIALISATION
x :∈ 0..10 ∧
y :∈ 0..10 ∧
z :∈ 0..20
OPERATION
OP1 =
BEGIN
z := x+y
END
END
GENERATION OF PROOF OBLIGATIONS
26/08/99 11
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS


The B Method - Proof Obligation
H1 ∧
H2 ∧
.
.
Hn ⇒
B
EXAMPLE
INVARIANTS
x ∈ 0..10 ∧
y ∈ 0..10 ∧
z ∈ 0..20
OPERATION
OP1 =
BEGIN
z := x+ y
END
Generation of a Proof Obligation
x ∈ 0..10 ∧
y ∈ 0..10 ∧
z ∈ 0..20
⇒
x+ y ∈ 0..20
26/08/99 12
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS

The B Method - Refinement



Smart Card Modeling

- B Method
- Verifier
- Interpreter
- Firewall

The Byte Code Verifier

- The Java byte code is compiled for the Java Virtual Machine.
- The Java byte code may be corrupted intentionally or not.
- Need to perform checks before its execution by the interpreter:
  - Flow controls
  - Type correctness

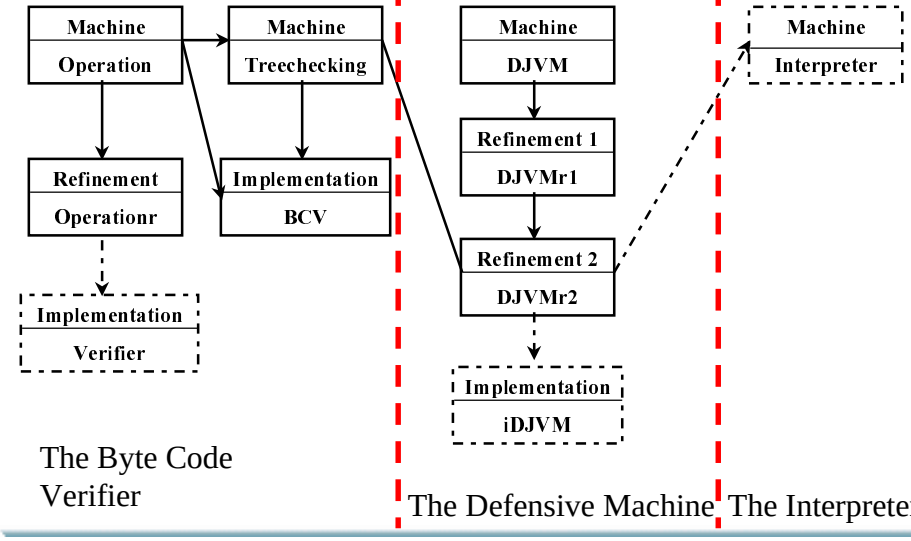
Flow Control and Type Correctness

- A state is defined by:
  - The pc (program counter)
  - The type stack
  - The type frame
- The properties to be checked are
  - Confinement
  - Stack access
  - Initialization
  - Type correctness

Our Approach of the Model

- Model a Defensive Machine.
- Extract runtime checks by successive refinements.
- De-synchronize verification and execution process.
- Split the defensive machine in two parts:
  - The verifier
  - The interpreter

The Model



The Defensive Machine

- Performing tests on byte code and then executing it.

```
ins_push0 = SELECT(methode(apc)=push0)
  THEN
    IF  (apc < size (methode) ∧
    top_stack < max_stack)
     THEN
      apc := apc + 1
      || top_stack := top_stack +1
      || types_stacks :=
    types_stacks{top_stack+1 ∈
INTEGERS}
     ELSE
       unchecked:=TRUE
     END
   END;
```

- No need to perform test on byte code, just executing it.

```
ins_push0 =
SELECT(methode (apc) = push0 ∧ unchecked
    = FALSE)
 THEN
     apc := apc + 1
    || top_stack := top_stack +1
    || types_stacks:= types_stacks{top_stack+1∈
INTEGERS}
   END;
```


The Freund & Mitchell Bytecode Instructions

- A subset of the Java bytecode language: Inc, Push0, Pop, If L, Istore x, Iload x, Halt, New, Init, Use.
- A static semantics and an operational semantics.
- A subset sufficient to study object initialization, flow and data-flow controls.

Construction of static stacks

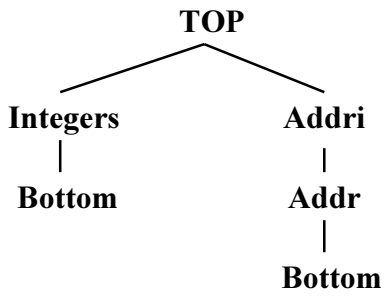
- The equation to be verified :

$$\forall pc,\ type_stack[pc] = \Pi\{\ f_i(type_stack[i] \ / \ i \text{ belongs to } Preds(pc)\},$$

- A fixed point search for each static stack.

Types

- The types subset:



- The lattice
  - A partial-order
  - A binary operator Meet

Conclusion

- The defensive machine is entirely proved
- The integration of the fixed point calculus is proved at 98%.
- We proved the soundness of our approach.

Smart Card Modeling

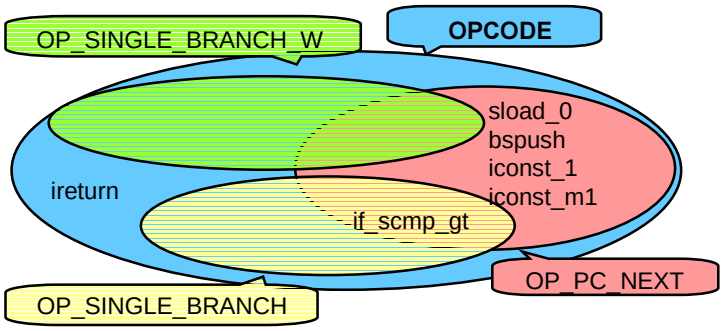
- B Method
- Verifier
- Interpreter
- Firewall

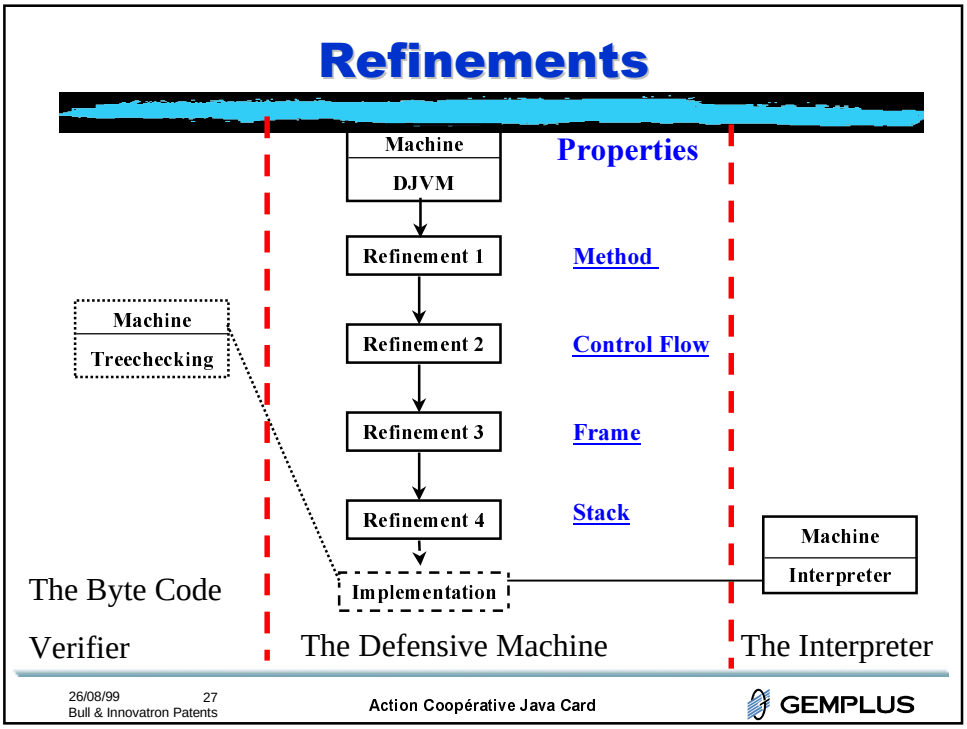
Java Card Entire Subset

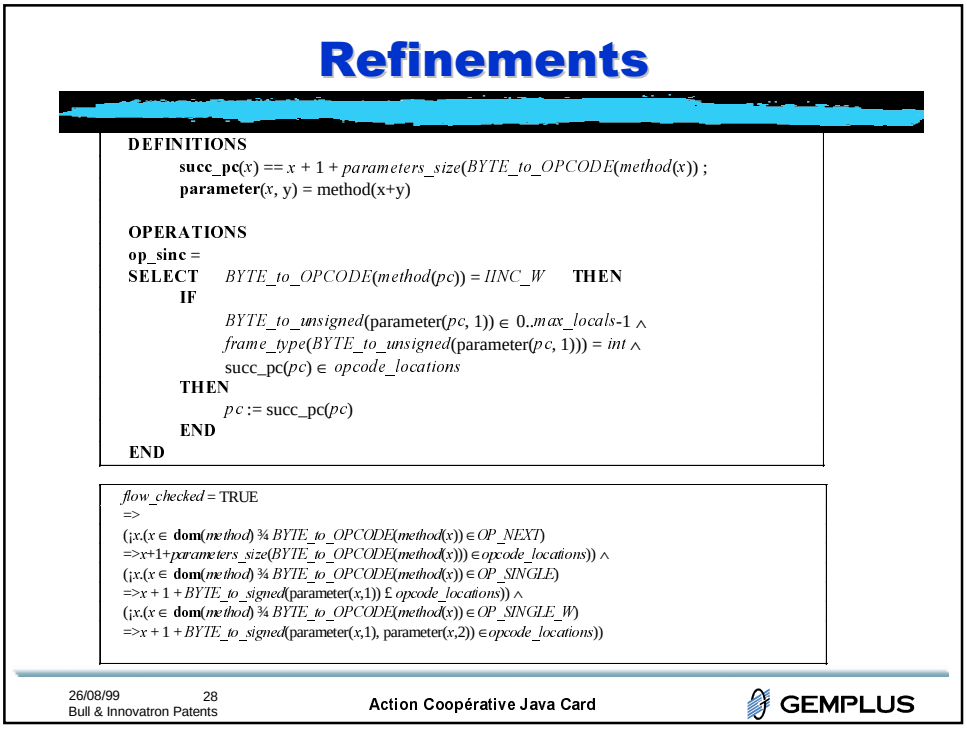
- Not taken into account
  - Constant Pool
  - Subroutines verification
  - Exception
  - Heap
- Instructions specification according to their properties:
  - Ease the specification
  - Ease the proof (cf. A. Requet)

Byte Code Properties

- Byte code accessing :
  - The stack (bspush)
  - The frame (sload_0)
  - The program counter (if_scmp_gt, bspush).




Refinements
Machine
DJVM
Properties
Refinement 1
Method
Machine
Treechecking
Refinement 2
Control Flow
Refinement 3
Frame
Refinement 4
Stack
Machine
Interpreter
Implementation
The Byte Code
Verifier
The Defensive Machine
The Interpreter
26/08/99 27
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS


Refinements
DEFINITIONS
succ_pc(x) == x + 1 + parameters_size(BYTE_to_OPCODE(method(x)) ;
parameter(x, y) = method(x+y)
OPERATIONS
op_sinc =
SELECT BYTE_to_OPCODE(method(pc)) = IINC_W THEN
IF
BYTE_to_unsigned(parameter(pc, 1)) ∈ 0..max_locals-1 ∧
frame_type(BYTE_to_unsigned(parameter(pc, 1))) = int ∧
succ_pc(pc) ∈ opcode_locations
THEN
pc := succ_pc(pc)
END
END
flow_checked = TRUE
=>
(¡x.(x ∈ dom(method) ¾ BYTE_to_OPCODE(method(x)) ∈ OP_NEXT)
=>x+1+parameters_size(BYTE_to_OPCODE(method(x))) ∈ opcode_locations)) ∧
(¡x.(x ∈ dom(method) ¾ BYTE_to_OPCODE(method(x)) ∈ OP_SINGLE)
=>x + 1 + BYTE_to_signed(parameter(x,1)) £ opcode_locations)) ∧
(¡x.(x ∈ dom(method) ¾ BYTE_to_OPCODE(method(x)) ∈ OP_SINGLE_W)
=>x + 1 + BYTE_to_signed(parameter(x,1), parameter(x,2)) ∈ opcode_locations))
26/08/99 28
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS

Refinements

```
op_sinc =
SELECT    frame_checked = TRUE ∧
          stack_checked = TRUE ∧
          flow_checked = TRUE
THEN
     pc := succ_pc(pc)
END
```

```
frame_value ∈ 0..max_locals-1 +→ INT

/* Gluing invariant */
dom(frame_type) ⊆ dom(frame_value)
```

```
op_sinc =
SELECT    frame_checked = TRUE ∧
          stack_checked = TRUE ∧
          flow_checked = TRUE
THEN
     VAR oldfvalue, newfvalue IN
          oldfvalue := frame_value(parameter(1)) ;
          newfvalue ← jah_sadd(oldfvalue, parameter(2)) ;
          frame_value(parameter(1)) := newfvalue
     END ;
     pc := pc + 3
END
```

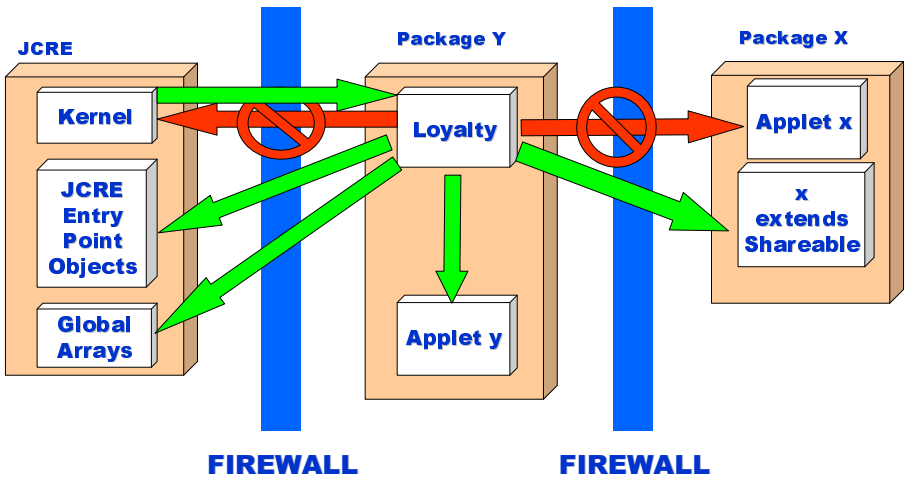

Status

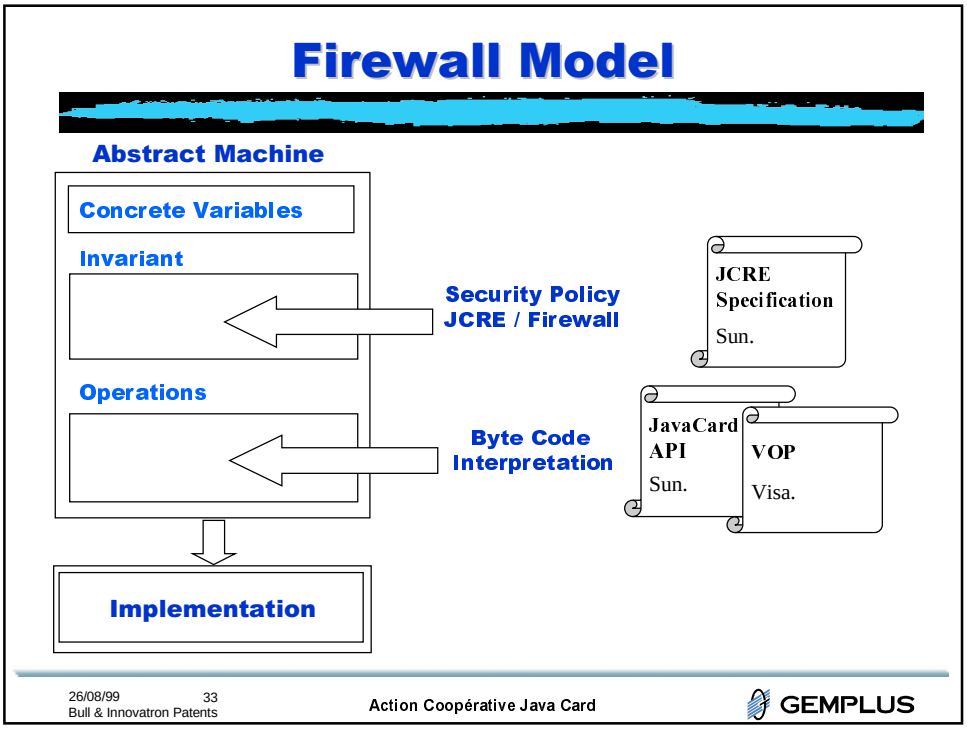
- All the byte codes are specified
- Proof of the properties per byte code sets is possible
- Proof Obligation resolution is difficult BUT generic
- 100% proved until refinement 3

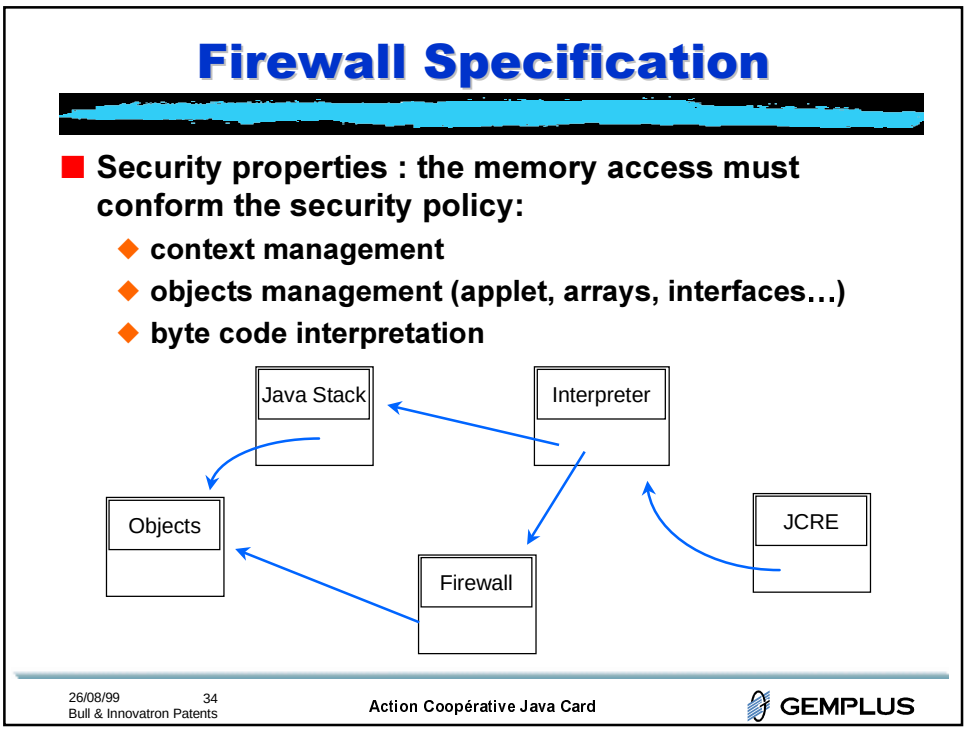
Smart Card Modeling

- B Method
- Verifier
- Interpreter
- Firewall

Object Sharing



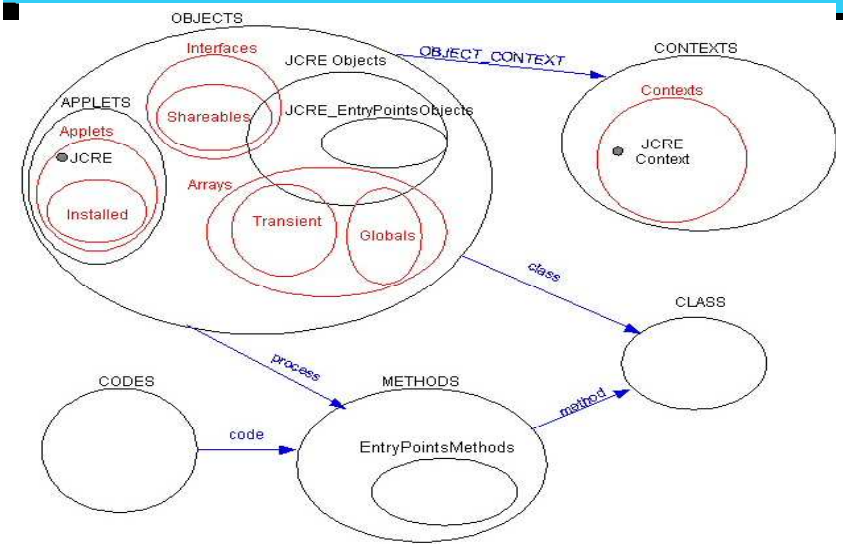

Firewall Model
Abstract Machine
Concrete Variables
Invariant
Operations
Implementation
Security Policy
JCRE / Firewall
Byte Code
Interpretation
JCRE
Specification
Sun.
JavaCard
API
Sun.
VOP
Visa.
26/08/99 33
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS


Firewall Specification
Security properties : the memory access must conform the security policy:
context management
objects management (applet, arrays, interfaces…)
byte code interpretation
Java Stack
Interpreter
Objects
Firewall
JCRE
26/08/99 34
Bull & Innovatron Patents
Action Coopérative Java Card
GEMPLUS

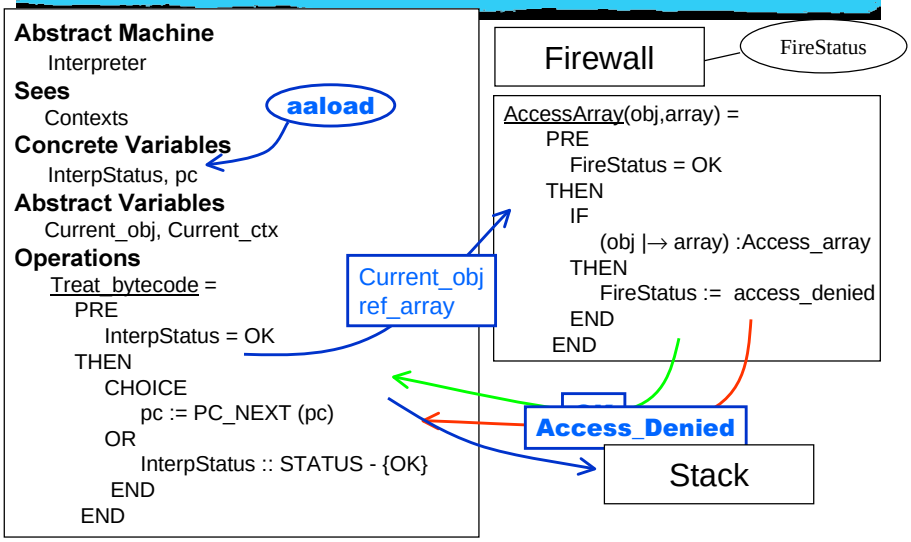
Byte Code Interpretation

- **Methods access**
  - ✔ invoke_interface, invoke_static, invoke_virtual
- **Array access**
  - ✔ aaload, iaload, baload, …
- **Context_switch**
  - ✔ invoke_interface

Components



Interpreter and Firewall



Firewall Refinement

- **Specification :** *" The acces to an array is allowed if*
  - ✔ *current object is element of the JCRE, or*
  - ✔ *array is global, or*
  - ✔ *array is not a transient clear_on_deselect, and is element of the current package"*

Access_Array =	Access_JCRE ∪ Access_global ∪ {Access_Package-Access_Transient_COD}

Access_JCRE : {ObjectContext⁻ (JCREContext)} ∗ ObjectsOnCard
Access_Globals : ObjectsOnCard ∗ GlobalArrays
Access_Package : {ObjectContext ; ObjectContext⁻}
Access_Transients : ObjectsOnCard ∗ Transients_COD

Firewall Implementation

- ***Access_Array*** **implementation**

IF

{curr_obj |→ array_ref} : Access_JCRE ∪ Access_global
∪ {Access_Package-Access_Transient_COD}

IF

ObjectContext (curr_obj) = JCREContext
OR is_global_array (array_ref) = TRUE
OR (same_package (curr_obj, array_ref) = TRUE
and not (is_transient_cod(array) = TRUE))

Conclusion

- **The Firewall is integrated is the virtual machine.**
- **100% proof of the model until the implementation**
- **Optimisation are mandatory on the additional test**
- **Help to understand the relation between the JCRE, the Firewall and the interpreter.**

Smart Card Modeling

- B Method
- Verifier
- Interpreter
- Firewall
- Conclusion

Conclusion

- We specified and/or implemented a large part of the virtual machine:
  - the verifier (spec only)
  - the interpreter
  - the firewall
  - the JCRE
- The implementations do not fit with smart card constraints
- Specification of the VOP module
- The complete interpreter (sub routine, exception…)

Publications

- *Using B Method to Model Protocols* **by J.-L. Lanet.** In Proceeding of the Workshop AFADL 98, Poitiers, Oct. 1998.
- *Formal Proof of Smart Card Applets Correctness* **by J.-L. Lanet and A. Requet.** In Proceedings of the Third Smart Card Research and Advanced Application Conference (CARDIS'98), Louvain-la-Neuve, Belgium, Sept. 1998.
- *The use of the B formal method for the design and the validation of the transaction mechanism for smart card application* **by P. Lartigue and D. Sabatier,** FM'99, Toulouse sept. 99
- *Formal Specification of the Java Bytecode Semantics using the B method,* **by L. Casset, J.-L. Lanet,** ECOOP workshop, Lisbon, Jun. 99
- *Formal Specification of the Java Byte Code Semantics Coherence for an Embedded System,* **by L. Casset, J-L. Lanet and G. Mornet,** submitted to ASIAN'99, Phuket Dec. 99
- *Formal Model of the Firewall,* **by S. Motre,** submitted to AFADL 2000, Grenoble, Fev. 00