(Ab)using foreign VMs: Running Java Card Applets in non-Java Card Virtual Machines

Michael Roland

2 December 2013 • MoMM 2013 • Vienna, Austria

This work is part of the project "High Speed RFID" within the EU program "Regionale Wettbewerbsfähigkeit OÖ 2007–2013 (Regio 13)" funded by the European regional development fund (ERDF) and the Province of Upper Austria (Land Oberösterreich).

NFC Research Lab Hagenberg • www.nfc-research.at
A research group of the University of Applied Sciences Upper Austria

Outline

- Introduction
  - Java Card
  - Open environment for debugging, testing and rapid prototyping
- Running Java Card Applications in non-Java Card VMs
  - Why do we want this?
  - What issues do we face?
  - Proposed solution
- Conclusion

Java Card Platform

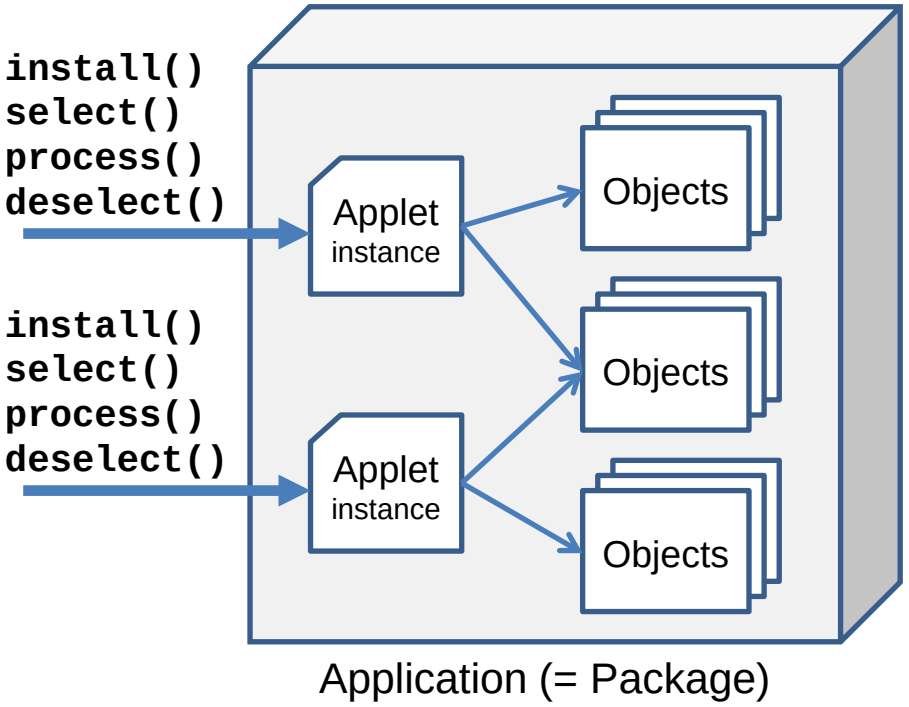
- Java Card Runtime Environment
  - Java Card Virtual Machine
  - Java Card API
- Java specifically designed for smartcards
  - Small footprint designed for tiny devices
    - Limited memory & processing power
  - Limited subset of Java language
    - Reduced set of primitive data types: **`boolean`**, **`byte`**, **`short`**, **`int`** (optional)
    - Some Java language constructs not supported
    - Most of Java core API not supported
    - No multi-threading
  - Smartcard-specific classes for application life-cycle management, APDU processing, cryptography, …

Java Card Virtual Machine

- All applications run in one VM
- VM lifetime = smartcard lifetime
  - VM runs from card production until card destruction
  - Code and data storage backed by persistent memory
  - Applications run across power-cycles of the card (from installation until deinstallation)
- Security: application firewalling
  - Strict separation between application contexts
  - Applications cannot access each other's data (unless explicitly granted permission)

Java Card Applications

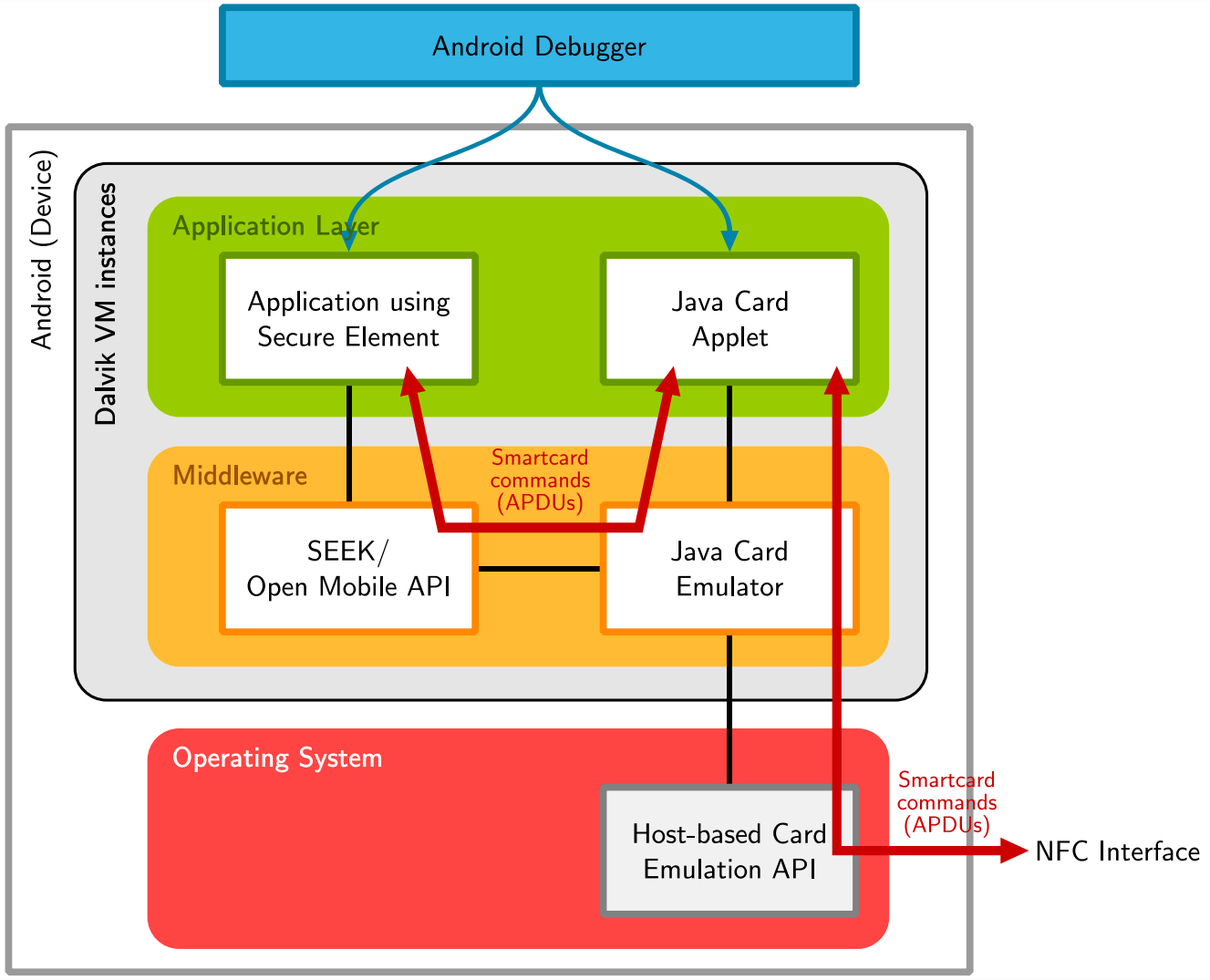
- One application consists of one or more applets
- Applet
  - entry-point object
  - life-cycle methods
    - invoked by JCRE
    - **`install()`**: create and initialize applet instance
    - **`select()`**: notify applet that it has been selected for command exchange
    - **`process()`**: forward received command APDU to applet
    - **`deselect()`**: notify applet that it has been deselected



Environment for Debugging, Testing and Rapid Prototyping

- Smartcard security prevents in-circuit emulation
  - Dedicated emulator environment necessary
- Emulation of complete run-time environment
  - Complete Java Card API (at least comparable to real cards)
  - Same application life-cycle as with real card
- In-place testing, debugging and prototyping
  - Test and debug in combination with other application components
  - Drop-in replacement for secure element for rapid prototyping (→ open but less secure)
- Emulation in Java/Dalvik virtual machine
  - Java Card language = subset of Java
  - Exisiting VM, existing tools for source-level debugging
  - Same tools as for debugging regular Java/Android applications

Emulator Integration with Android NFC Devices



Issues with Emulation in Java/Dalvik Virtual Machine

- Java Card atomic transaction mechanism
  - Java does not have an atomic transaction mechanism by default
  - Variables/objects involved in transactions cannot easily be rolled-back to a defined boundary
- State of JCRE and applications is not persistent
  - Significant differences between lifetime of Dalvik/Java VM and Java Card VM
  - If emulator environment is terminated (e.g. app closed, device rebooted) the state of the JCRE and all Java Card applets is lost
  - Upon restarting the emulator all applets start at the beginning of their life-cycle

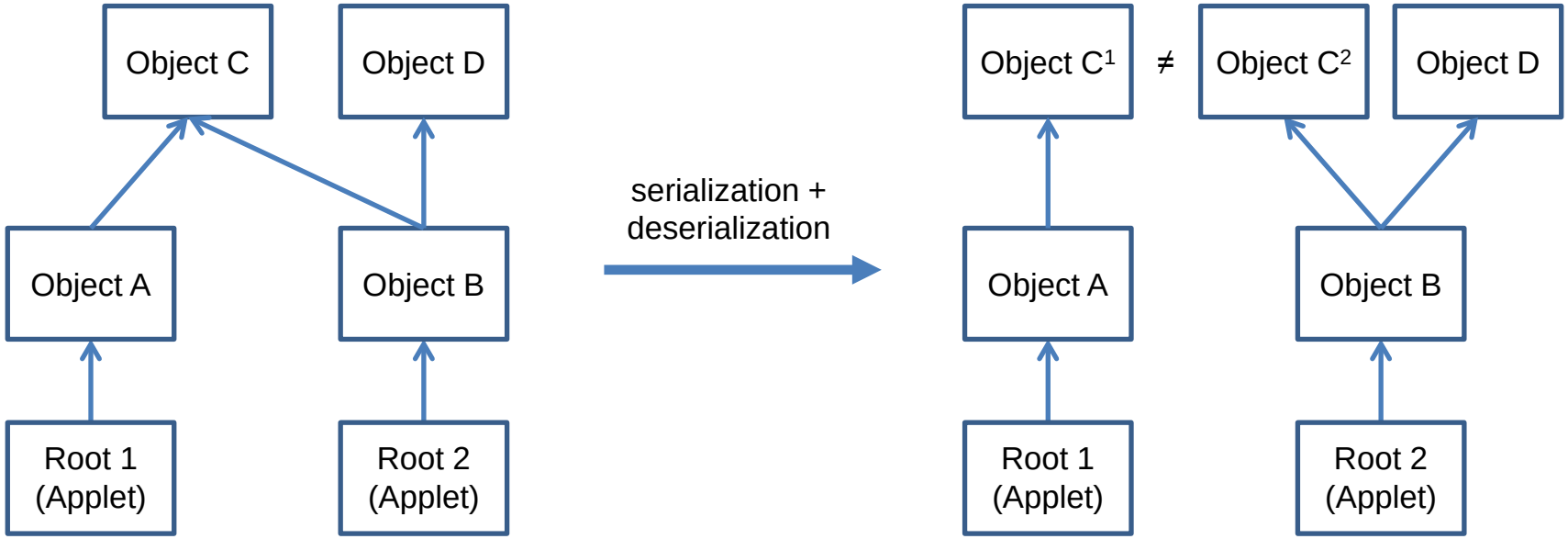
→ *Methods for extracting and re-implanting application state necessary!*

Goals

- Store and load networks of objects from persistent memory
  - Start at one or more root nodes (Java Card applet instances)
  - No duplication of objects referenced from multiple locations
- Store and load static fields of classes
  - Based on a list of classes or within a package scope
- Java Card applications should be used as-is
  - No modifications to source code should be required
  - No pre- or post-processing of application source code should be required (source-level debug-ability; same code as run on card)
- Should work within typical VMs (e.g. Oracle Java SE VM, Android Dalvik VM)

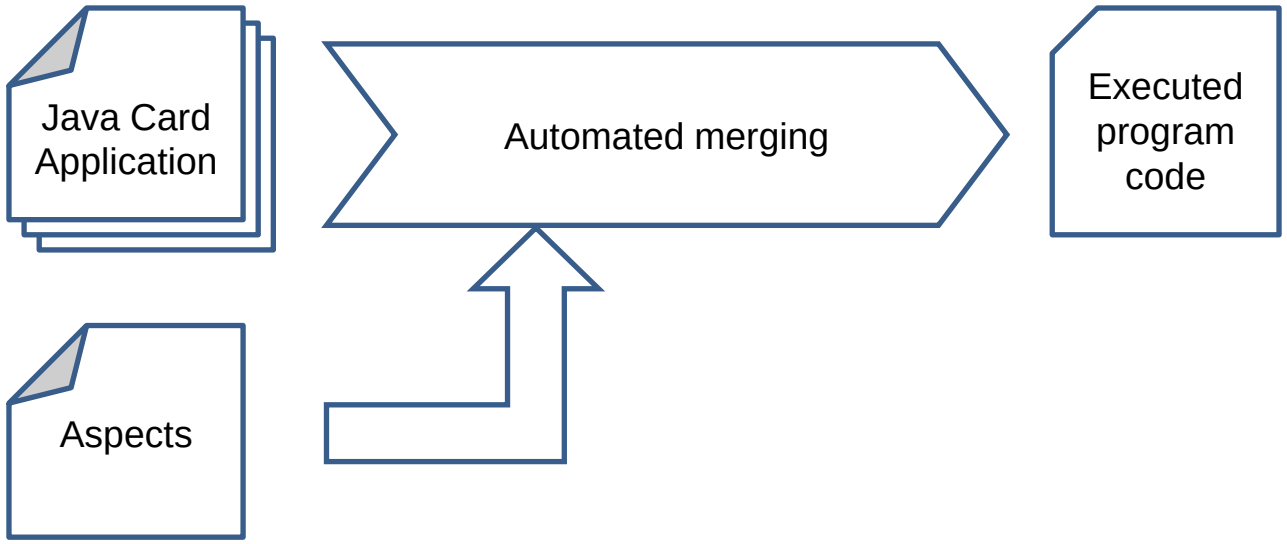
Existing Methods

- Java serialization
  - Serialize/deserialize objects into byte stream
  - Code modifications necessary
  - Serialization can only have one root object
  - Only complete object graph can be serialized/deserialized



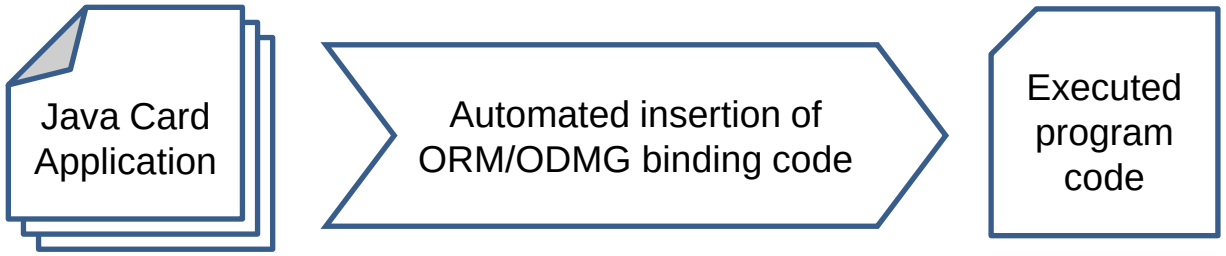
Existing Methods

- Aspect-oriented programming (AOP)
  - Add aspects that intercept read/write access to data
  - Automated pre-/post-processing of program code necessary
  - No source-level debugging of original application



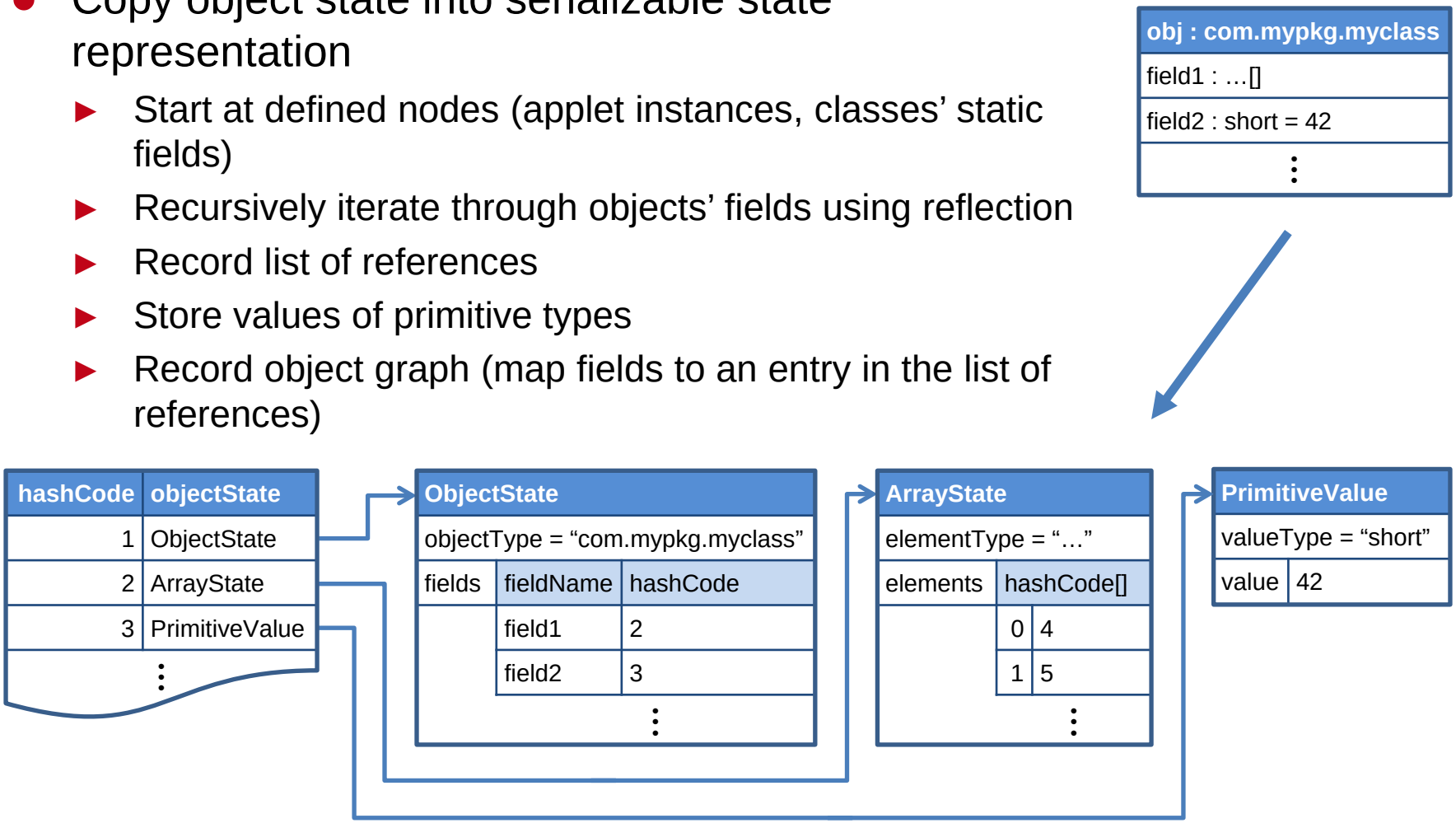
Existing Methods

- Object-relational mapping (ORM) / ODMG binding
  - Map objects to database (relational or object-oriented)
  - Code modifications (annotations, special constructors, getter/setter methods) or automated pre-/post-processing of program code necessary
  - No source-level debugging of original application



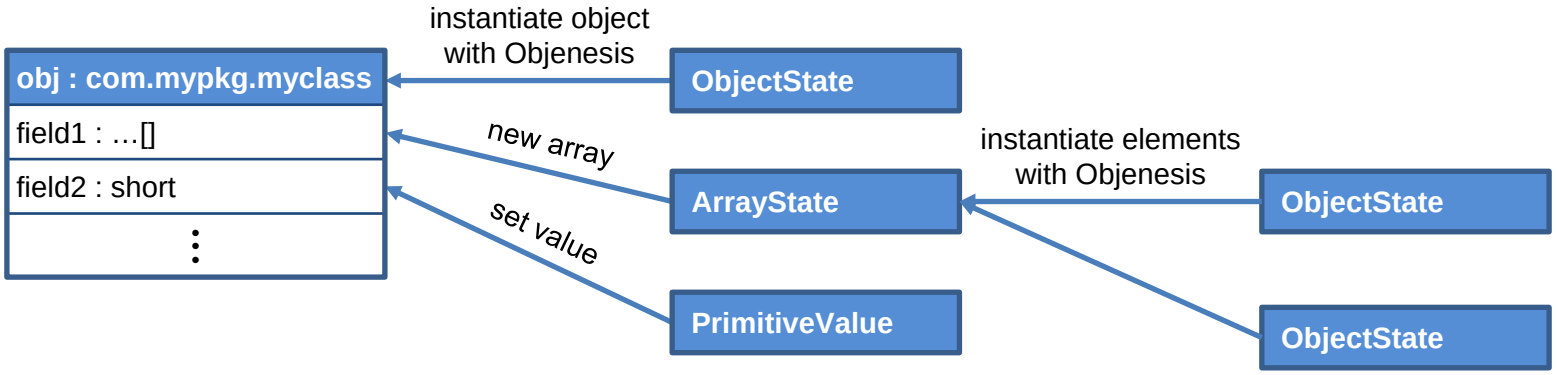
Proposed Solution

- Copy object state into serializable state representation
  - Start at defined nodes (applet instances, classes' static fields)
  - Recursively iterate through objects' fields using reflection
  - Record list of references
  - Store values of primitive types
  - Record object graph (map fields to an entry in the list of references)



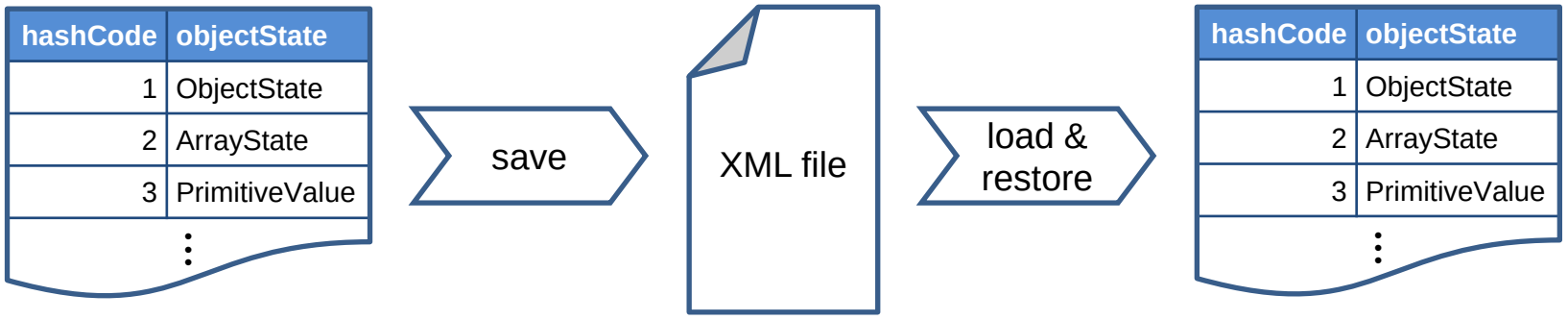
Proposed Solution

- Recreate object graph from object state representation
  - Start at defined node (possible to restore only sub-graphs)
  - Use Objenesis library to instantiate objects without calling their constructor (no special constructors or tagging interfaces necessary)
  - Recursively fill objects' fields with stored primitive values or restored object references



Proposed Solution

- Store/load object state representation to/from persistent memory
  - State representation designed for easy export to XML and easy import from XML
  - Save to XML file when application is about to close
  - Load and restore from XML file upon start of application



Conclusion

- Running Java Card applications on standard Java VMs or the Android Dalvik VM permits easy source-level debugging using standard debugger tools
- Problem: Life-cycle of Java Card VM is different from other VMs
  - Applications live in persistent memory
- Created proof of concept to introduce Java Card-style persistence to other Java VMs

Dr. Michael Roland

Research Associate, NFC Research Lab Hagenberg
University of Applied Sciences Upper Austria

michael.roland (at) fh-hagenberg.at

This work is part of the project "High Speed RFID" within the EU program "Regionale Wettbewerbsfähigkeit OÖ 2007–2013 (Regio 13)" funded by the European regional development fund (ERDF) and the Province of Upper Austria (Land Oberösterreich).

NFC Research Lab Hagenberg • www.nfc-research.at
A research group of the University of Applied Sciences Upper Austria