Convergence OSGi-JavaCard : Fine-grained dynamic update

Agnès C. Noubissi, Julien Iguchi-Cartigny, Jean-Louis Lanet

Department of Computer Sciences, XLIM Labs, University of Limoges

23/09/2010

Dynamic Software Update

What is it?

Update either applications or system components while running without restarting the system or stopping the application.

Goals

- Fix bugs or correct some vulnerabilities,
- Improve perfomances by adding, deleting or modifying some functionnalities,
- Increase system security.

Case Study

Application fields . . .

1. Contact cards
   - Sim cards (infinite lifespan),
   - Bank cards (two years)
2. Contactless cards
   - Electronic passports (five years, renewable),
     $\Rightarrow$ Dynamically update system components while passing through the reader terminal.

Traditional update vs HotSwUp

SwUp

- Traditional update : stop, apply update and restart,
- Loss of the system state ⇒ Loss of execution contexts,
- Stopping services offer and associated services.

HotSwUp

- HotSwUp : apply update while running
- Don't stopping any application or any system component ⇒ Don't stop services offer

Concerning Java Card

Java Card

- **Post-issuance** ⇒ Ability to update applications,
- But for system components ⇒ creation of a new card.

And . . .

Java Card Virtual Machine never stops ⇒ Need of dynamic update of API components (cryptographic algorithms)

Limitations of HotSwUp technics for smart cards

- Implemented and tested for servers and desktops,
- Resource and security constraints of smart cards.

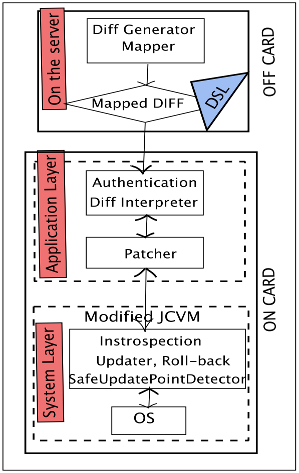
Our solution

Constraints

- Reduce download overhead,
- Reduce memory footprints,
- Reduce energy consumption,
- And ensure security.

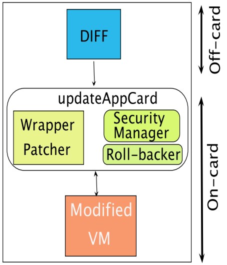
- Extends the existing Java Virtual Machine (JVM),
- EmbedDSU $\Rightarrow$ A JVM with HotSwUp mechanism,
- Solution based on off-card and on-card mechanisms.

General workflow approach

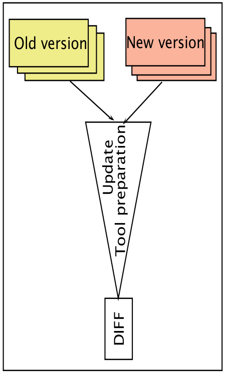


Two parts . . .

- Off-Card : prepare update,
- On-Card : effective update of components.



Off-Card : prepare the update



Off-card process

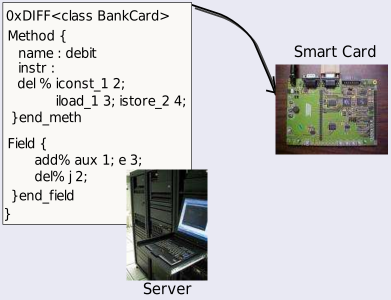
- Conception of a Domain Specific Language (DSL) used to express changes beetween class files (DIFF),
- Implementation of the DIFF generator.

DIFF generator

- Input : Two class files,
- Output : Express changes into a dedicated language,
- Goal : Restrict update process to those parts of the program that are affected by modifications.

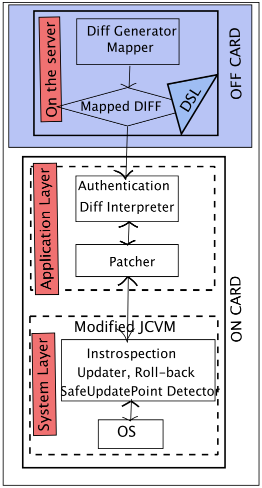
On-Card : download into the Card

Process



```
0xDIFF<class BankCard>
Method {
   name : debit
   instr :
  del % iconst_1 2;
           iload_1 3; istore_2 4;
 }end_meth
Field {
      add% aux 1; e 3;
      del% j 2;
 }end_field
}
```

On-Card : update process

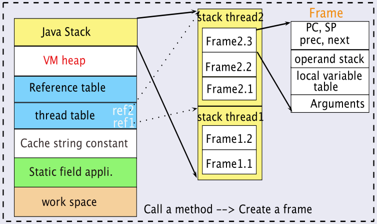


Process based on JVM modification

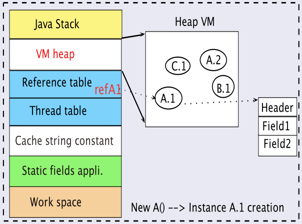
- JVM modification
  - JVM instrospection,
  - Detection of Safe Update Point.
- Wrapper Module implementation
  - Authentication,
  - DIFF Interpreter.
- Updater Module implementation
  - Update class byte code,
  - Update class objects in the JVM heap,
  - Update frames in the JVM stack.

Run through the JVM

Stack View



Heap View

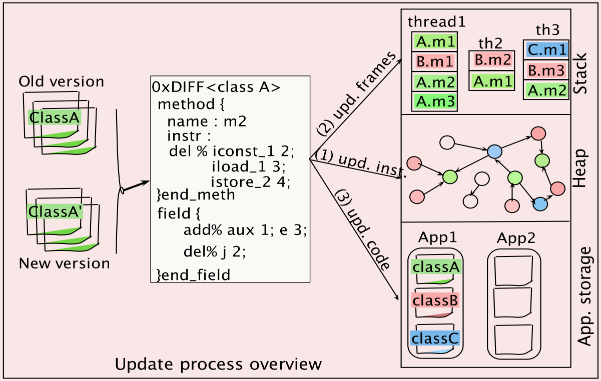


Modification of JVM ⇒ EmbedDSU

- To interpret DIFF,
- To introspect heap, stack frame,
- To update data instances, frames and byte code of updated class.

Update process overview

Global view on update process



When can we apply the update process ?

⇒ Detects the *safe update point* . . .

On-Card update process : Safe Update Point

Goals

- Update must be atomic,
- Update must happen at point call *safe point*,
- To ensure coherence of execution context after update.

How to detect it?

- Obtain all methods changed by the update (signature, bytecode, local variables, etc),
- Check if one of those methods are in the stack frame,
- If yes, then delay the update.

Dynamic Update Process ...

The proposed solution for JavaCard

- Works properly,
- Test on an evaluation board (AT91 EB40A) which caracteristics is near to JavaCard classic target,
- Metrics are presented in another paper.

OSGi seems to be an other perfect target for adapting that solution.

What about DSU in OSGi?

- Presentation of OSGi,
- What does an OSGi module looks like?
- DSU in the context of OSGi : Process, advantages and weaknesses,
- Proposed solution : Architecture, and process

Presentation of OSGi

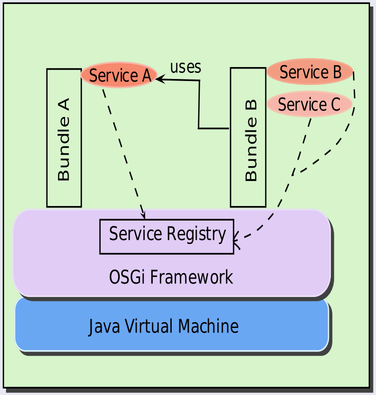
OSGi Definition

The Open Services Gateway Initiative is a programming model to develop Java Applications from modular units called Bundles

OSGi : Two pieces

- OSGi Framework ⇒ For deploying and execution service-oriented applications
- And service interfaces : set of standard service (bundle) definitions which can run on the framework.

OSGi Framework

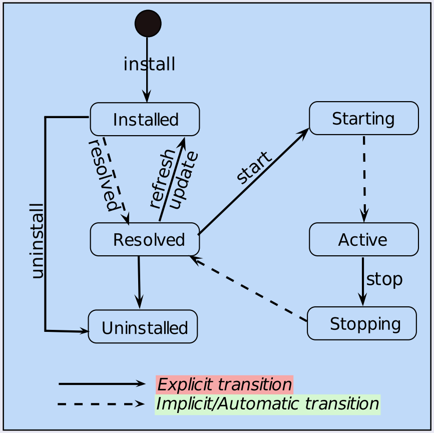


What does an OSGi module look like?

OSGi Bundle : A JAR file

1. It is a set of classes,
2. With a special file called MANIFEST.MF containing metadata informations like
   - Name of the bundle
   - version,
   - list of imports and exports (services),
   - Minimum Java version that the bundle needs to run on, etc.

OSGi Bundle Lifecycle



DSU in the context of OSGi

Bunlde Update Process

- De-activate the old version of the bundle :
  - Remove listeners,
  - Unregister exported services,
  - Remove the service objects,
  - Release the bounded service,
  - Release all resources used by the bundle objects.
- Load and install the new bundle,
- And activate the new one.

OSGi Dynamic Update

- Apply at bundle level,
- Don't stop the OSGi Framework,
- Don't stop the Virtual machine,
- But stop the bundle itself.

DSU process on OSGi : Simplistic?

Advantages

- Component bundles can be added and updated at runtime,
- Powerful event mechanisms are supported by the framework.

Weaknesses

- Update process of bundle
  - Deactivates of the bundle to be updated,
  - Loads of the corresponding new classes,
  - And calls of the start method
- Loss the state of the bundle component when it is updated $\Rightarrow$ Stops the running associated objects and release the resources it holds after installed the new version.

Proposed solution

Goals

- Don't destroy the execution state of the bundle during update $\Rightarrow$ Existing instances continue to be running during the update process,
- Dynamic update at the level of the bundle component that include the execution state transfer of the old version to obtain the new version ones.

Approach

JVM-Based approach $\Rightarrow$ Modify the virtual machine in order to introspect the OSGi Framework and virtual machine data structures to offer dynamic update at the class and bundle level without loss the execution state.

Proposed solution : Architecture

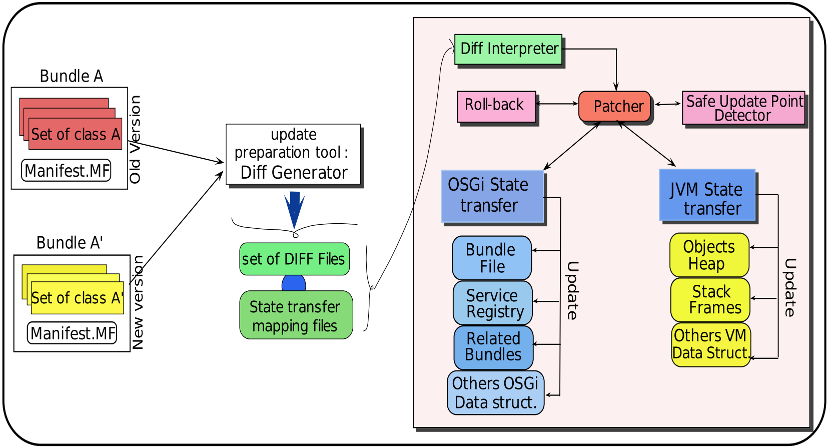


Figure: Proposed DSU Architecture

Proposed solution : Update Process (1)

The proposed Update Module : Two parts

1. The first part
   - is encapsulated as an OSGi Bundle,
   - registers the update service to the OSGi Framework,
   - and should be started before the others bundles except the system bundle.
2. The second part is link to the modification of the VM in order to offer some features like
   - Safe Update Point detector,
   - Introspection of the VM and OSGi data structures,
   - Roll-back when detects non-atomicity of the update,
   - And VM state transfer.

Proposed solution : Update Process (2)

Proposed Update process : First phase

- Preparation of the Update (Diff Generator),
- Interpretation of the Diff files and state transfer files and send neccessary intructions to the patcher,
- Detection of Safe Update Point,
- Dispatch information by the patcher to OSGi State transfer and JVM State transfer module,

Proposed solution : Update Process (3)

Second Phase

- OSGi State transfer perfoms
  - Architecture adaptation ⇒ Update the structure of the application. It can be applied when new components are added or removed, or when some interconnections are modified.
  - Interface adaptation ⇒ Modify the list of services provided by the bundle component.
  - State adaptation ⇒ Adapt the old OSGi execution context of the bundle to obtain the new one relative to the new bundle.
- JVM State transfer perfoms the heap objects, stack frames, and others VM data Structures adaptations.

Paper Contributions

In this paper :

- we present our update process for updating classes in JavaCard based Smart Cards
  - Diff in off-card,
  - Modification of the virtual machine to interpret the Diff and patch the old version in on-card.
- we explain the OSGi's update process and present the weaknesses of the process,
- we propose a to adapt our successful solution used for JavaCard in the context of OSGi,
- and then, we present the architecture of our approach apply on OSGi which is an on-going work.

Thank You for your attention !!