Andre Bottaro, PhD | Orange Labs
Fred Rivard, PhD | IS2T

OSGi ME
An OSGi Profile for Embedded Devices
Executive Summary

• OSGi is a reference for flexible component based platforms and standardization on the Home Market.

• So, why OSGi does not take off in the Home Market? Because OSGi has drawbacks for embedded devices.

• OSGi ME
  - keeps the core features of OSGi technology
  - is compliant with Java ME CLDC
  - simplifies OSGi technology for simpler needs
  - strengthens robustness
  - and requires much less resources than OSGi to target massive deployment
OSGi is a reference for flexible component based software architecture
OSGi is used for flexible software

- OSGi, the module layer at the basis of
  - success stories (see OSGi web site)
  - numerous European projects
  - most of Enterprise application servers
  - some Vehicle gateways
  - some Mobile applications
  - some e–Health projects
  - numerous Home prototypes
  - …
OSGi model is the primary reference in Home standardization bodies

- **Broadband Forum**: Software lifecycle management is lead by OSGi concepts
- **HGI**: OSGi is the main software platform pushed in HGI specifications
- **UPnP Forum**: OSGi is the primary reference for Device Management
- **OSGi concepts and specifications**
  - JSR232: Mobile Operational management
  - JSR 248–249, JSR 277, JSR 291…
A Business breakthrough is expected on the Home Market

- Home boxes have now complex software
- Linux is the most spread Operating System
- Home industrial status: Software upgrades is hard to do and done only on a yearly basis

- Home market asks for a breakthrough to deliver an exploding world of applications
  - Digital Home: Multimedia sharing
  - Smart Energy: Optimization of the Energy consumption at Home
  - E–Health: Elderly care at Home

- Could OSGi technology accelerate service delivery?
- Could OSGi technology build a Home Application Store?
- Could OSGi technology open a box to third-party applications?

« For embedded devices, the main consideration is not longer which OS to employ but instead, which application platform to use»

F.A.S.T, European Commission, Nov. 2005
So why OSGi does not take off on the Home Market?
OSGi deployment is still not big in the Home

- What is said about OSGi
  - The reference to deploy a future Application Store for Home boxes
  - A standard modular platform implemented by many providers
  - Numerous standardized software bundles available
  - Strong open source communities and commercial providers
  - A pledge for patent abandon by the most active actors

- So is OSGi the right candidate?
  - Few (but successful) deployments outside the Enterprise domain
    - Cars, Trucks, Mobile, ...
    - Home: hard to say since operators do not talk much ...

- What is the problem?
  - OSGi technical requirements are too high for the Home Market
    - OSGi yet requires a Java environment of some MBs, with redundant functionalities with the underlying OS
  - OSGi modularity has to be improved for Embedded devices:
    - More reliability and stronger isolation
    - Missing resource management for openness to third parties
The Home ecosystem requires

- a **standard** software environment
- **open** to third party applications
- with **agile** development techniques
- while remaining compatible with **embedded** constraints
  - Keep device cost low
  - Make robust applications

To create a dynamic market of applications with Home players:

- Applications designers (bundles & services)
- Platform administrators & application stores
- OSGi ME providers (SW part of the OSGi ME platform)
- Hardware providers (HW part of the OSGi ME platform)
Why OSGi ME?

- All OSGi fundamentals apply for the Home Market
- But OSGi current version is too heavy, not robust enough

⇒ Need for a new profile to target embedded constraints

- OSGi ME: a proposition for a Profile for Embedded Devices
  - OSGi RFP 126 states the requirements
    - OSGi software flexibility and openness to 3rd parties
    - Java ME CLDC compliance: the most spread edition
    - Aligned with the embedded nature of Home devices
  - Orange Labs and IS2T to deliver the specification
  - IS2T to deliver the first reference implementation
  - Orange and IS2T to compare OSGi and OSGi ME on an ARM9 based hardware
OSGi ME in a nutshell
OSGi ME in a nutshell

• Keeping the core features of the OSGi technology...
  – fine-grained code sharing and isolation model
  – dynamic software management
  – requiring a full upward compatibility from OSGi ME to OSGi

• and being compliant with Java ME CLDC...
  – no user-defined class loaders on Java ME CLDC
  – Java CLDC APIs compliance

• while simplifying OSGi technology for simpler needs...
  – remove unnecessary and semantically complex features

• and strengthen robustness.
  – no stale reference
  – fully ordered initialization sequence
  – transactions
Keeping the core features of the OSGi technology…

• Fine-grained code sharing and isolation between bundles
  – The foundation of the openness to third party applications

• Dynamic software management
  – Install, update, uninstall, start and stop individual bundles
  – Perform operations at runtime

• OSGi standard APIs
  – Enable a full binary upward compatibility from OSGi ME to OSGi.
Fine-grained code sharing and isolation

- Modularity demands a strict model of what is guaranteed at runtime
  => A code change in a bundle has only an impact on bundles that declare code imports.

- Openness to third party applications demands an even stricter model
  => Distinct actors will only share object access through declared APIs

- OSGi foundation is a unique basis (vs Linux, Windows, .NET, Android,...)
  - A bundle declares code imports / code exports / code that remains private
  - OSGi framework guarantees bundle declarations
    - **Sharing**: a bundle will have visibility on APIs that are imported
    - **Isolation**: a bundle will never access private code and code that is not imported
  - Sharing is efficient: direct method call between bundles (no IPC)
Part of OSGi R4 APIs do not totally enforce sharing & isolation

- Any bundle can load any private class and instantiate it
  - BundleContext.getBundles() + Bundle.loadClass()
  - Class.forName()

- Any bundle has visibility on any implemented class signature
  - `<service object>.getClass()` gives the private name of classes implementing shared APIs

- Any bundle has visibility on any file or resource
  - BundleContext.getBundles + Bundle.getBundleContext().getDataFile()
  - BundleContext.getBundles + Bundle.getBundleContext().getResource()
  - BundleContext.getBundles + Bundle.getBundleContext().findEntries()

- Any bundle can register services for other bundles
  - BundleContext.getBundles()
  - + Bundle.getBundleContext().registerService()

⇒ OSGi precludes Java 2 security to ensure isolation
OSGi ME ensures a stricter model for sharing & isolation

- Class lookup is refined
  - `Class.forName()` behavior depends on the context
    (the bundle containing the class defining the method calling `Class.forName`)
  - `Bundle.loadClass` is banished since any `Bundle` object is accessible

- Private class signature are never visible to other bundles
  - `<service object>.getClass()` depends on the context

- The access to the bundle context of other bundles is banished
  - `Bundle.getBundleContext` is removed in OSGi ME
  - `BundleContext.getResource()` and `BundleContext.registerService()` thus really depend on a specific bundle context.

⇒ OSGi ME ensures a strict model for sharing & isolation and does not preclude Java 2 security to do so
OSGi ME adapts
Software Dynamics to business needs

- **Different market requirements** on binary code downloads
  - 1st case: No download due to certification, B.O.M., threats
  - 2nd case: Controlled downloads for only well-known (approved) services, with threat control, proprietary protocols via proprietary media, e.g., uart, spi, i2c, CAN, Ethernet, GSM, Zigbee, ...
  - 3rd case: Authorization of any download from any kind of sources

- **Device Software Dynamics levels characterize how binary code is loaded into devices**
  - DSD 0 = no download
  - DSD 1 = download through a controlled media
  - DSD 2 = no restriction
Device Software Dynamics

- The code is loaded using some probe, e.g., JTAG, or bootloaders
  - The device needs to be “switched off” to download the whole application
  - No runtime download

- A closed system
  - OSGi ME to design a component based application
  - The whole application code is known at link-time
  - Full off-board linking is feasible
Device Software Dynamics 1

- **Code loading under strict control**
  - The media by which the code gets downloaded to the device is controlled by some technical means, most probably protected by some proprietary protocol.

- **A controlled system**
  - OSGi ME to design a component based application
  - The application can download bundles at runtime
  - Pre-linking and off-board pre-analysis is feasible
  - Adding and/or Updating are feasible
Device Software Dynamics 2

- **Free downloads**
  - The device has to embed all the necessary protections.

- **A fully open system**
  - OSGi ME to design a component based application
  - The application can download any bundle at runtime
  - All linking is done on-board
  - Adding, updating, uninstalling feasible without constraints
and being compliant with Java ME CLDC...

- Java ME CLDC is Java most spread edition on embedded devices
  - Spread on mobile phones and part of M2M modules
  - OSGi specification is not compliant because it specifies how to implement sharing & isolation (class loaders)

- No (heavy and too flexible) user-defined class loaders in Java ME CLDC
  - Although at the basis of OSGi code sharing and isolation model
  - And at the basis of OSGi dynamic software management features
  ⇒ Because they are not the only technical solution, OSGi ME specification does not mention class loaders

- Interfaces used at runtime must be declared at development time
  - Dynamic or optional imports are not compliant and not needed
  ⇒ OSGi ME reinforces OSGi class import declarative model

- Restricted Java APIs
  ⇒ OSGi ME is compliant with embedded Java environment
simplifying OSGi for simpler needs...

• **Remove unnecessary features** that are against a strict sharing and isolation declarative model
  - dynamic imports
  - optional imports
  - bundle requirements
  - bundle fragments
  - bundle extensions

• **Remove complex features**
  - Loading of several versions of a same exported class
and strengthen robustness.

- OSGi ME enables a more robust dynamic service cooperation
  - Avoid *stale references*
  - Improve garbage collection
  ⇒ Propose an API to *semantically build a service out of several cooperative objects*
    ⇒ Explicitly attach objects to services
    ⇒ Throw a *DeadServiceException* on access to unregistered services or their objects

- OSGi ME makes the program initialization deterministic
  ⇒ *With a fully ordered class initialization only depending on application code*

- OSGi ME provides transactional guaranties
  ⇒ *An API must allow transactional guaranties*
Focus on initialization for robustness

• JVM specification
  “The intent here is that a type have a set of initializers that put it in a consistent state and that this state be the first state that is observed by other classes.” (JVM specification section 2.17.4)

• OSGi ME reinforces this intent:
  • unambiguous start-up sequence
  • No possible deadlock while the (Java) device starts.
  • Uniqueness of the “first-state” (same for each device's start).
Transactional life cycle for robustness
First results on an ARM926 based hardware
Reference implementation validation device

- Tested on an Orange prototype
  - ARM9 399MHz
  - 256MB Flash
  - 128MB RAM (execution in RAM)

- Compared software architectures

```
<table>
<thead>
<tr>
<th>Application components into OSGi™ bundles</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSGi™ R4 Platform</td>
</tr>
<tr>
<td>Java VM (&gt;Java ME CDC)</td>
</tr>
<tr>
<td>Linux</td>
</tr>
<tr>
<td>Home Automation Box prototype</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application components into OSGi™ bundles</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSGi ME Platform</td>
</tr>
<tr>
<td>JHAB (Java ME CLDC)</td>
</tr>
<tr>
<td>bare metal</td>
</tr>
<tr>
<td>Home Automation Box prototype</td>
</tr>
</tbody>
</table>
```
The demo available at OSGi Community Event 2010

Remote access to a Home device power management application
The user remotely manages the power states of devices
## Resources comparison

<table>
<thead>
<tr>
<th></th>
<th>OSGi</th>
<th>OSGi ME</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware</strong></td>
<td>ARM926</td>
<td>399Mhz</td>
</tr>
<tr>
<td><strong>Framework footprint</strong></td>
<td>~400KB</td>
<td>~40KB</td>
</tr>
<tr>
<td><strong>JVM footprint</strong></td>
<td>~2,000KB</td>
<td>~200KB</td>
</tr>
<tr>
<td><strong>OS footprint</strong></td>
<td>~11,000KB</td>
<td></td>
</tr>
<tr>
<td><strong>Application footprint</strong></td>
<td>~800KB</td>
<td>~350KB</td>
</tr>
<tr>
<td><strong>Java RAM</strong></td>
<td>~700KB</td>
<td>~550KB</td>
</tr>
<tr>
<td><strong>Startup time (HW+SW)</strong></td>
<td>33.0s</td>
<td>5.5s</td>
</tr>
</tbody>
</table>

⇒ Footprint compatible for flash MCU mass market devices.

⇒ OSGi ready for the embedded Home Market, thanks to OSGi ME profile.
The Embedded OSGi application

- Embedded UPnP & Web Services libraries
- 17 OSGi ME bundles running on the Home Automation Box
- 350kB application over OSGi ME & VM
Lessons learned

• OSGi concepts are the right basis for flexible software

• OSGi ME concepts could make OSGi a real breakthrough

• A breakthrough never comes without moves
  - Applications and OSGi service compendium have now to be ported on Java ME CLDC APIs
  - Java ME CLDC VMs have to implement fine-grained sharing and isolation mechanisms
What it means for developers

• C developers can add Java/OSGi modules on the same hardware as before
  ⇒ The migration plan is easier

• OSGi developers are guided on best practices
  - Before
    - developers needed lessons to know best practices
    - impossible profiling tasks were needed
  - With OSGi ME, developers can use
    - only strict sharing and isolation model
    - embedded virtual machines and APIs
Roadmap and next steps

• Orange & IS2T specification to be public soon. This will be available to other actors and the Alliance without constraints.

• IS2T about to sell the first OSGi ME implementation

• Java VM & software editors will be able to make their own

• Next steps
  - Build tools to automatically convert OSGi bundles to OSGi ME framework and benefit from the current OSGi community
  - Specify and insert resource management into OSGi ME framework specification for services: CPU, memory, IO
Thanks!

Andre Bottaro
andre.bottaro@orange-ftgroup.com

Fred Rivard
fred.rivard@is2t.com

OSGi ME
An OSGi Profile for Embedded Devices