OSGI – Open Services Gateway Initiative

Fritz Solms

August 21, 2015
OSGI – Open Services Gateway Initiative

Old standard
- developed in 1999 by OSGI Alliance
- used widely for plugin frameworks
  - e.g. Eclipse

Massive renewed interest
- Wide applications within ubiquitous computing
  - Mobile phones
  - Internet of Things
  - Automobiles
  - Entertainment, . . .

Reasons:
- Services are
  - discoverable, self-healable, secure, ..
- remote management
  - starting, stopping, updating, . . .
What is OSGi?

Definition

OSGi provides a dynamic modular Java-based component/application platform facilitating interoperability and remote management of software components (including services) across broad variety of devices.

- developed for resource-constrained devices
  - $\approx 300\text{kB} - 1\text{MB}$ memory footprint
- What is a Component (versus a class)?
  - Concrete realizations:
    - object, service, function
- OSGI component/service platform
  - deploy and manage components within it
  - Can have application create its own
Features of OSGi

- Contracts based services model (with versioning)
- Each plugin
  - versioned artifact
    - different versions of same artifact may co-exist.
    - even in same container.
  - own class-loader
- Declared plugin dependencies
  - Either as
    - embedded jars
    - other versioned (possible remote) plugins
    - loaded on demand
- Dynamic service wiring through an active registry.
  - service consumers automatically rebound if service
    - stopped and restarted
    - rebound to new concrete service provider
- Open standard with many O/S & vendor implementations
  - interoperability across implementations
- Flexible and dynamic security.
Applications

- Long-running services
- Interacting IoT components
- Modular cloud applications
- Application server components
- Vehicle software components
- Plugin frameworks
OSGi Applications (Bundles)

- **OSGi component**
  - set of Java classes plus
  - component descriptor: `Manifest.MF`
  - installed as unit
  - defined life-cycle

- **Meta-data in `Manifest.MF`**
  - name, `version`, description
  - activator
  - packages being exported
    - Java classes to be made externally accessible
  - packages being imported
    - Dependencies sourced from the environment
OSGi – Open Services Gateway Initiative

Overview

OSGi concepts

Bundle Life-Cycle

- **Resolved**: OSGi framework has successfully resolved all code dependencies for the bundle.
- **Starting**: BundleActivator.start() has been called but has not yet returned.
- **Active**: Bundle running and ready to process requests.
- **Stopping**: BundleActivator.stop() has been called but has not yet returned.
- **UnInstalled**: OSGi framework has successfully resolved all code dependencies for the bundle.
Bundle Activator (1/2)

- Specified in Manifest.MF:
  - Bundle-Activator: com.example.MyActivator

- Perform activation/de-activation activities:
  - provides start & stop
Bundle Activator (2/2)

- Must have default constructor
- Often
  - empty
  - only way to get hold of bundle context
    - e.g. to get handle to framework services
OSGi – Open Services Gateway Initiative

Overview

OSGi concepts

OSGi Services Registry

- Service
  - represented by interface
- Provides
  - naming service
    - like white pages
  - interface repository
  - directory service
    - like yellow pages
Dependency Management

- Why?
  - Improved decoupling
  - Dependency Injection $\rightarrow$ improved testability
  - Improved reliability
    - NULL-Object pattern $\rightarrow$ neutral null behaviour
  - Support for interception, aspects, adapters
  - Automatic re-registration
  - Code reduction
- Not standardized
  - Felix DM, Spring DM Server, . . .
- Publish to DM
  - component/service interface/contract
  - component implementations
  - component dependencies (on contracts!!)
OSGi Frameworks

Apache Equinox = reference implementation
- OSGi platform used in Eclipse
- mainly as a plugin framework

Apache Felix

Knopflerfish OSGi R5 implementation
- O/S OSGi framework
- very mature, efficient & scalable

Kura = OSGi framework for M2M/IoT
- Industrial equipment & vehicle monitoring & control
- RaspberryPi
- Wearables, . . .

Apache Karaf = light-weight higher-level framework for OSGi container
- embedd Felix, Equinox, . . .
- Provides higher-level services (console, logging, . . .)
OSGi – Open Services Gateway Initiative

Overview

OSGi supporting frameworks

**OSGi Supporting Frameworks**

- Component distribution & management
  - *Apache Ace*
- Dependency injection of dynamic services
  - *PeaBerry*
    - extension of *Google Guice*
OSGi — Open Services Gateway Initiative

Overview

OSGi as middleware for the Internet of Things (IoT)

OSGi = Middleware for IoT?

- More and more widely adopted standard for IoT
  - OSGi standards explicitly aim to target IoT
  - basis for Eclipse IoT stack

- Why?
  - Discoverable components/services
  - Remotely manageable
  - Versioning support
    - multiple versions of same component may be simultaneously active
    - support for dependencies on particular versions
  - Interoperability across O/S & vendor implementations
  - Small footprint
  - Embeddable within application
  - Designed for environment where components come & go
  - Good support for security
Heating Controller example

- Demonstrates:
  - Components may/may not be available
  - Automatic reconnect on start
  - Decoupling (only dependency on APIs)
    - use dependency management
OSGi Components & their Dependencies