Technical Specification Overview

Introduction
What is the OSGi Service Platform?

- A Java framework for developing remotely deployed service applications, that require:
  - Reliability
  - Large scale distribution
  - Wide range of devices
  - Collaborative

- Created through collaboration of industry leaders
- Spec 2.0 publicly available at [www.osgi.org](http://www.osgi.org)
1. Framework Specification
2. Package Admin Service Specification
3. Permission Admin Service Specification
4. Service Tracker Specification
5. Log Service Specification
6. HTTP Service Specification
7. Device Access Specification
8. Configuration Admin Service Specification
9. Metatype Specification
10. Preferences Service Specification
11. User Admin Service Specification
OSGi Specifications Content

R1
- Device Manager
- Http Service
- Log Service
- Preferences Service
- Configuration Admin
- Service Tracker
- User Admin

R2
- Package Admin
- Permission Admin
- Wire Admin
- XML Parser Service
- Measurement
- Position
- Connector Service
- Jini Service
- UPnP Service

R3 Preliminary
- Start Level
- URL Handler

Framework

Execution Environment
Essentials

• Reliable
  – Large-scale deployments fail without extremely high reliability

• Portable
  – Attract third-party developers to create essential innovative services

• Dynamic
  – Allow configuration to adapt to user & operator needs over time

• Secure
  – Protect service providers from each other
  – Guarantee a prescribed quality of service

• Scalable
  – Members have very different configurations for their deployment of OSGi frameworks

©1999-2002 OSGi, All Rights Reserved
Core Functions

- Life Cycle Management
  - Install, Start, Stop, Update, Uninstall
- Registry of services with notifications
- Package and version management
- Open remote management architecture
- Strict separation of specifications and implementations
  - multiple implementations
Why Java?

• Portable byte code format
  – Deliverables independent of OS/CPU

• Security is integral to the language

• Integrity by not allowing dangerous functions

• Mature platform

• Active developer community & broad industry support
  – Availability of programmers
  – Large available code base
Java Issues

- **Integrity**
  - Cooperative, not like an OS with guarantees

- **Security**
  - Performance intensive

- **Performance cost**

- **Size**

- **Lack of Resource Management**
Technical Specification Overview

Framework Specification
Framework Entities

- OSGi environment
- Framework
- Bundles
- Services
- Filters
- Events
- Security
OSGi Environment

- Bundle (Application)
- Java VM
- Operating System
- Driver
- Hardware

= service interface exported and imported by bundles
Framework

- Allows applications to share a single Java VM
- Manages applications
  - Life cycle, Java Packages, Security, Dependencies between applications
- Service registry for collaboration
- Extensive notification mechanism
- Policy free
Framework Entities

OSGi Framework

Bundle A
{}

Bundle B
{}

Bundle C
{}

= service, java interface
A **bundle** is the deliverable application
- Like a Windows EXE file

A bundle registers zero or more services
- A service is specified in a Java interface and may be implemented by multiple bundles

Searches can be used to find services registered by other bundles
- Query language

The Framework itself is represented as the **system bundle**
• A bundle is delivered in a Java ARchive (JAR) file (zip) containing:
  – Resources to implement zero or more services
  – Manifest with information about the bundle:
    • Dependencies on other resources (Java packages)
    • Name, description, copyrights, vendor, profile, etc.
  – A class that can start/stop the bundle (Bundle Activator)

• Can also act like a library (E.g. DLL file)

• Framework can install/update/uninstall bundles dynamically
Management

- Framework provides *mechanisms*, but is *policy free*
- Management policy provided by a bundle specific to the operator:
  - Called a *Management Bundle*
- Management policies made/selected by the operator
- Enables standardized OSGi management bundles from network management vendors
Management, protocols or API?

- Local Ethernet
- Local Powernet

Management System

Access Net

Private protocol for example: SNMP, CIM, SyncML, etc.

= Bundle

OSGi Environment
Collaborative model

• More than an Applet, MIDlet, Xlet runner

• Bundles can collaborate through:
  – service objects
  – package sharing

• A dynamic registry allows a bundle to find and track service objects

• Framework fully manages this collaboration
  – Dependencies, security
Collaborative model

- OSGi Framework
- Bundle
  - Service registry
  - packages
- Bundle
  - packages
- JAVA
- Operating System
- Hardware

©1999-2002 OSGi, All Rights Reserved
How does OSGi relate to MIDP, Java TV, Browsers

Java Application Manager

Midlet, Xlet, or Applet

Midlet, Xlet, or Applet

JAVA

Operating System

Hardware

No management bundles

No collaboration

No package management (versions!)

No native code
• A *service* is an object registered with the Framework by a bundle to be used by other bundles

• The semantics and syntax of a service are specified in a Java interface
public interface Log {
    public void log(String s);
}

public class SimpleLog implements Log {
    public void log(String s) {
        System.out.println(s);
    }
}

Bundle

Gets (with query)

Log

interface

implements

Simple Log

IBM Log

Motorola Log
• Different bundles (from different vendors) can implement the same interface
  – Implementation is not visible to users
  – Allows operator to replace implementations without disrupting service

• OSGi defines a standard set of services
• Extensive notifications for services life cycles
• Services have a unique id
A Service Defined

Object that implements a Java Interface

events: REGISTERED, UNREGISTERING, MODIFIED
The Framework Service Registry is available to all bundles to collaborate with other bundles.

- Requires permission
  - Under operator control

- Services are associated with properties
  - Query language to find appropriate service

- Bundles can update the properties
• How does a bundle customize a service per bundle?

• The ServiceFactory interface allows a bundle to return a specific customized service object for each bundle

• Framework calls back the factory for service get and unget methods.
Service Factories

Framework

Bundle A

{ }

Bundle B

{ }

Bundle Standard

Bundle Factory

©1999-2002 OSGi, All Rights Reserved
• The Framework manages the dependencies between bundles
• Bundles that are installed and started will register services
• Framework will automatically unregister services when a bundle stops
• Event notifications for all important events
Dependencies

Install A

start

events: install

Bundle A

{}
Dependencies

- Uninstall
- stop
- events: uninstall

Framework

- Bundle A
  - events: unregister

- Bundle B
  - events: unregister

- Bundle C
An example with services

- Bundle A
  - register

- Bundle B
  - Events: register, unregister

- Bundle C
  - publisher

- Registry

- WAP bundle
  - get
How to find a service

- Framework supports powerful Filter object
- Syntax based on LDAP string based filters
- Searches properties registered with a service
  - Common data types, including arrays and vectors
Filter Syntax

• Expressions
  - And
  - Or
  - Not

• Comparisons
  - Less
  - Greater
  - Approximate
  - Equals
  - Substring
  - Presence

Expressions:
- And \((\&(...)(...)(...))\)
- Or \((|(...)(...)(...))\)
- Not \((!(...))\)

Comparisons:
- Less \((attr<=9)\)
- Greater \((attr>=9)\)
- Approximate \((attr\sim=osgi)\)
- Equals \((attr=9)\)
- Substring \((attr=OSG*i)\)
- Presence \((attr=*)\)
• Sub expressions
  - (&(vendor=ERICSSON)(priority>4))
  - (|(vendor~!=ERICSSON)(vendor~!=Microsoft))
  - (&(objectclass=org.osgi.service.log.Log)(vendor~!=IBM))

• Filter is useful in many contexts
  - Reduces query classes like used in Jini

• Can be used for application filtering as well
  - Applied widely in OSGi services
• Services register and unregister all the time
• Onus on the programmer to only use services that are registered
• ServiceTracker simplifies this task:
  – Maintains a list of active services
  – Events can be overridden in subclass
• Used in almost every bundle
Classpath issues

- Java applications consists of *classes* placed in *packages*

- Java searches for a package or class in different jar files and directories
  - These are usually specified in the CLASSPATH environment variable

- Applications can extend this search path dynamically with so called classloaders
Classpath issues

- Java programs are decomposed in packages
- A class name is always prefixed with the package name
  - `package ericsson.osgi.profiler;
    class X {}`
  - Is actually class called "ericsson.osgi.profiler.X"
  - Creates globally unique names
- A Java package is an indivisible unit of a set of related classes
The Classpath in OSGi

- Each bundle has its own class loader
- Bundles can only shares packages when:
  - Import and export clauses in the manifest match
  - Have permission to do so for those packages
- The framework manages the overall CLASSPATH for bundles
- Assures that all bundles use the same class
- Tracks shared usage of packages between between bundles
# OSGi Classpath

## Framework
- org.osgi.framework
- org.osgi.service.http

## Bundle A
**Export**
- org.osgi.service.log
- com.ibm.service.log
- com.ibm.j9

**Import**
- org.osgi.service.http
- javax.servlet.http

## Bundle B
**Export**
- ericsson.osgi
- javax.servlet
- javax.servlet.http
- org.osgi.service.log

**Import**
- org.osgi.service.http

A resolved

B resolved
Package Admin Service

- Policy interface to Framework package handling
- Uninstalling a bundle will not withdraw exported packages
  - anymore (this was optional in Release 1)
- Inspect current state of im- and exporting
- Cleanup stale exported packages by stopping and starting the minimum set of bundles
Native code

- JAR file contains both Java classes + native code
- Matching of correct operating system, processor and language
- Life cycle bound to bundles life cycle
- Requires Java 1.2 support
- Notice that JAR format is only delivery format
Native code

• Java Supports native code with the "native" keyword
  - public class Native {
    public native void test();
  }

• OSGi has a number of Manifest headers allowing shared libraries to be included
  - Manifest-Version: 1.0
    Bundle-Name: NativeCodeTest
    Bundle-NativeCode:
      http.dll;os=Win95;processor=x86
      http.so;os=Solaris;processor=x86;language=en
      http_deutsch.so;os=Solaris;processor=x86;language=de
      http_sparc.so;os=Solaris;processor=sparc
Dependency Management

- When a bundle stops, it is cleaned up
  - Registered services are removed
  - References to other services removed
- Bundles can be notified when a service they depend on is unregistered
- Class path dependencies are managed
- Model allows long running applications with dynamic software updates
Real code! Hello World

- HelloWorld.java:
  package org.osgi.nursery.hello;
  import org.osgi.framework.*;

  public class HelloWorld implements BundleActivator {
      public void start( BundleContext context ) throws Exception{
          System.out.println( "Hello world" );
      }
      public void stop( BundleContext context ) throws Exception {
          System.out.println( "Goodbye world" );
      }
  }

- Manifest:
  Manifest-Version: 1.0
  Bundle-Activator: org.osgi.nursery.hello.HelloWorld
To get and register a service

- public interface GPS {
    ...
}

- public void foo(BundleContext context) {
    Garmin garmin = new Garmin(...);
    Hashtable properties = new Hashtable();
    properties.put("vendor", "garmin");
    ServiceRegistration reg = context.registerService(
        GPS.class.getName(),
        garmin,
        properties);
}

- public void bar(BundleContext context) {
    ServiceReference ref = context.getService(
        GPS.class.getName());
    GPS gps = context.getService(ref);
    ...
}
OSGi (optionally) uses Java 1.2 Security based on Permission classes

A Permission class defines the semantics of what a permission means

For example: FilePermission
- Target = directory or file, "/tmp/-"
- Action = what can be done, READ, WRITE

Stack based checking

Flexible but very complex and CPU intensive
• Optional
  – Base Framework API not linked to java.security

• OSGi uses a single (1) protection domain per bundle
  – Java 2 allows multiple
  – Simpler

• Framework is, again, policy free

• Administration is done via the Permission Admin service
Java 2 Security overview

Diagram:
- Protection Domain
- Code Source
- Permission Collection
- Permissions
- File Permissions
- File Permission
- Policy
- Security Manager
- Access Controller
- Open file
- Check permission
- Get stack trace
- Use minimal permission
- Check(FilePermission)
- Implies(FilePermission)
- Implies(FilePermission)
- Implies(FilePermission)
- foo()
- bar()
- open()
Framework uses Permission Admin Service to administer permissions

Permissions associated with bundle location
  - Allows setting before bundle is downloaded

Synchronous Bundle Listener added so that a Management Bundle
  - Set the permissions Just In Time

(Simple) Serializable format for permissions
Technical Specification Overview

Component Specifications
Content R1, R2, R3

OSGi Specifications

Execution Environment

Framework

R1
- Device Manager
- Http Service
- Log Service
- Preferences Service
- Configuration Admin
- Service Tracker
- User Admin
- Package Admin
- Permission Admin

R2
- Wire Admin
- XML Parser Service
- Measurement
- Position
- Connector Service
- Jini Service
- UPnP Service
- Start Level
- URL Handler

R3 Preliminary
Log Service

- Simple and small Log service for operator
- 4 Levels
  - INFO, DEBUG, WARNING, ERROR
- Automatically logs framework events in a defined way
- Other bundles can access log history
  - Management bundle
  - Length implementation dependent
org.osgi.service.log v1.1
Log Service

- A log user
- Log entry
- LogService
- LogEntry
- LogListener
- A log user bundle
- A log user
- LogService
- LogEntry
- LogListener
- LogReader
- LogReader Service
- A log reader
- A log reader bundle
- A log reader
- LogReader Service
- Send new log entry
- Retrieve log
- Store a message for retrieval and broadcast
- Message log
- A log entry impl.
- A log entry impl.
- A log service impl.
- A log service impl.
- Log Service Impl. bundle
- ©1999-2002 OSGi, All Rights Reserved
• Provides web access to bundles
• A powerful servlet runner
  – Supports Servlets Version 2.1
• Very simple to export static pages and files (like images)
• Automatically unregisters servlets when bundle is stopped
org.osgi.service.http v1.1
Http Service

- Impl. Of Http context
- Bundles main code
- Impl. Of servlet
- javax.servlet.Servlet
- javax.servlet.Request/Response
- HttpContext
- NameSpace Exception
- HttpService
- Default impl. HttpContext
- Resource registration
- NameSpace alias
- Servlet registration

©1999-2002 OSGi, All Rights Reserved
Dynamic device driver download model

Plug & Play
- Plugged in devices identify themselves
- Device Manager will download appropriate bundle

Matching process for best driver

Extendable
- Driver Selector, Driver locator
Device Access

1. Insert camera

Camera

IEEE 1394B Interface

Network bundle

2. register

Device Manager

3. Select device

Device

4. find bundle location

Driver Locator

5. Install bundle

TV bundle

6. register

Sony CCD654

9. Select camera

Camera

8. Register camera

7. attach

Driver

10. Show camera on TV

TV
• Configures bundles
  – At startup, or any later moment

• Maintains a repository of configurations
  – Local
  – Management system

• Configurations are key/value pairs
  – Typed with Meta Types

• Can be extended with plugins
• Data Descriptors for generic editors
  – Configurations, Properties

• Supports
  – Basic types like String, Integer, Byte, Short ...
  – Arrays and Vectors

• Uses LDAP Objectclass, attribute model

• Can be localized for different languages

• General validation support
Meta Typing Class Diagram

- MetaType Provider
- ObjectClass Definition
  - Locale PID
  - Attribute Definition

1..n
1
• Simple hierarchical model like Windows Registry

• Uses simple hierarchical names
  – /bundle/121/httpport=81

• Different trees
  – Multiple named trees per bundle
  – One system tree

• Storage can be local or on management system
Preferences Class Diagram

- BackingStore Exception
- Preferences
  - Preference Node impl.
  - User name
  - Root user nodes
  - Root system node
- a bundle
  - Preferences Service
  - Bundle preferences
  - Service impl.
User Admin Service

- Repository of users
- Maintains data for authentication and other purposes
  - Private keys, passwords, bio-profile, User Preferences
- Powerful role based authorization model
  - Users, group of users, and groups of groups
- Administrative functions
User Admin Class Diagram

- **User Admin**
  - **impl**
  - **Listener**
- **UserAdmin**
  - **Permission**
  - **Event**
  - **authenticate**
  - **Request Authenticator**
  - **Action impl**
  - **Has roles**
  - **Consult for authorization**
  - **authenticate**
  - **Receive events**
  - **Send event**
- **User**
  - **impl**
  - **Group**
  - **Group impl**
  - **Role**
  - **Role impl**
  - **User impl**
- **Basic member**
- **Required member**
Wire Admin Service (R3)

- Connects Producer services to Consumer services via Wire object
- Wire objects have properties for configuration
Position, Measurement, State

- Support classes for Wire Admin
- Position
  - Supports GPS like position
- Measurement
  - SI measurement system to prevent calculation errors
  - Error calculations
  - Timestamp
- State
Start Levels (R3)

- Allows Management Agent to control startup/shutdown sequence of installed bundles
- Supports many levels
- System service, implemented by Framework
• Manages the URLStreamHandlerFactory and URLContentHandlerFactory of Java
  – These factories can only be set once so the Framework must manage them

• Bundles can register a URLStreamHandlerService or ContentHandler

• The Framework will automatically add these to the standard set
Universal Plug 'n Play (R3)

- Popular UPnP specifications adapted to OSGi Service Platform
- Makes it very easy to write a UPnP control point or device
- Involves registering a simple UPnPDevice service
• Defined how Jini devices can cooperate with an OSGi service platform

• Service Platform very good environment for Jini services, but

• Security issues
Execution Environment (R3)

- OSGi defined 2 execution environments
- Minimal
  - Intended for really small devices
- Foundation Profile
  - Adapted from JCP’s Foundation Profile
- Finally!
IO Connector Service (R3)

- Adopted J2ME javax.microedition.io package
- Process in JCP is intended to make this stand-alone
- Connection type + address selected by a single string
- Connection interfaces allows client code to adapt to different schemes
  - Datagram, Sockets, Servers
XML Parser Service

- Allows different bundles to register an XML parser
- Supports finding "best" parser for a specific application
- Based on standard Java JAR service