Mobile Java™ Technology JSR 232 Architecture and Benefits

Jon Bostrom

Director, Architecture

Nokia
http://www.nokia.com/

Gábor Pécsy

Software Technology Specialist

TS-3757

Goal of the Talk

Understand how a Service Oriented Architecture Creates New Opportunities for Mobile Java Technology

The New Mobile Service Economy Requires Faster Cycle and Lower Costs

The old way to create mobile services is just too slow:

- Standardize technology, specify static terminal APIs, Manufacturers implement these in products, ensure vendor compatibility, wait for terminal volumes, and then, finally announce the service
  - Compare this to the way how services spread in fixed Internet (Skype, Google search etc.)
  - Using JSR 232 operators can dynamically customize the service offering available to their development community

→Speed of innovation in internet services sets a reference for innovation in mobile services

Why Next Generation Mobile Java Technology

- Mobile Java technology is the Standard technology for connecting mobile devices into the Operator Service Environment, the new WEB 2.0 Network Service Environment, and the Enterprise environment
- Mobile Java programming environment is the best development environment to enable developers to take advantage of the key capabilities of mobile terminals while connecting to network services
- Java technology provides the best integration with Enterprise development by leveraging a component based Service Oriented Architecture and allowing mobile terminals to use the capabilities of the emerging Enterprise Service Bus (ESB)

Terminal API Evolution

1st Generation APIs… Slow but **DONE**

- Static APIs to Basic Terminal features

2nd Generation Terminal APIs… Slow but **DONE**

- Static APIs to Services running in the native OS platform

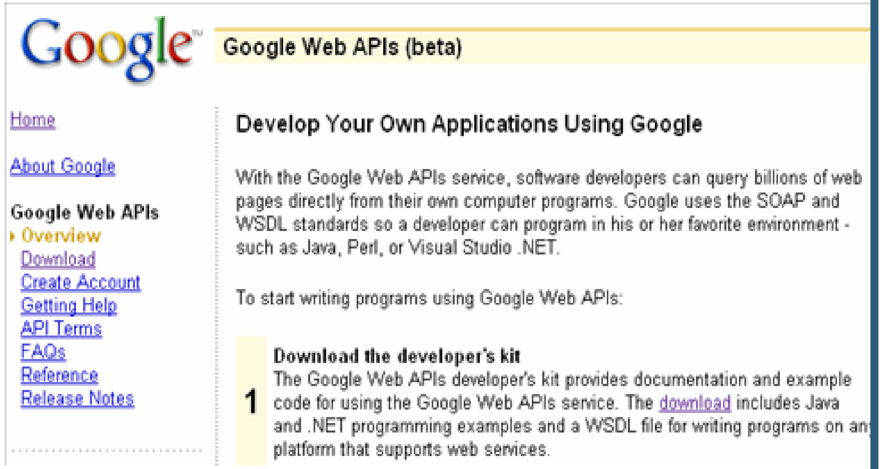
3rd Generation APIs reflecting Network Services…

- Continuously Evolving! Mobile Operational Management (JSR 232)/OSGi
- Dynamic APIs to Services running in the network… Web 2.0 and Operator Network Services
- Service APIs are dynamically available and updatable for the application developer
- Services shield the application developer from implementation, protocol, and networking details
- Services can be created and deployed with the assurance of security and authority

The Network Service Environment (Web 2.0)

- Web 2.0 is a term often applied to the ongoing transition of the World Wide Web from a collection of websites to a full-fledged computing platform serving web applications to end users
- Web 2.0 is the network… as a platform, spanning all connected devices
- Web 2.0 paradigm shifts:
  - A network platform enabling the utilization of distributed services
  - The phenomenon describing the transformation of the web from a publication medium to a platform for distributed services
- Examples are Google, Yahoo, Amazon, Network Operators, many others that are creating a Service Platform from the World Wide Web by opening their service APIs to third party developers

Mobile Java Technology Innovation and Time to Market



Google Web APIs (beta)

Home

About Google

Google Web APIs
- Overview
- Download
- Create Account
- Getting Help
- API Terms
- FAQs
- Reference
- Release Notes

Develop Your Own Applications Using Google

With the Google Web APIs service, software developers can query billions of web pages directly from their own computer programs. Google uses the SOAP and WSDL standards so a developer can program in his or her favorite environment - such as Java, Perl, or Visual Studio .NET.

To start writing programs using Google Web APIs:

1 **Download the developer's kit**
The Google Web APIs developer's kit provides documentation and example code for using the Google Web APIs service. The download includes Java and .NET programming examples and a WSDL file for writing programs on an platform that supports web services.

Developers Utilizing the Next Generation Mobile Java Platform:

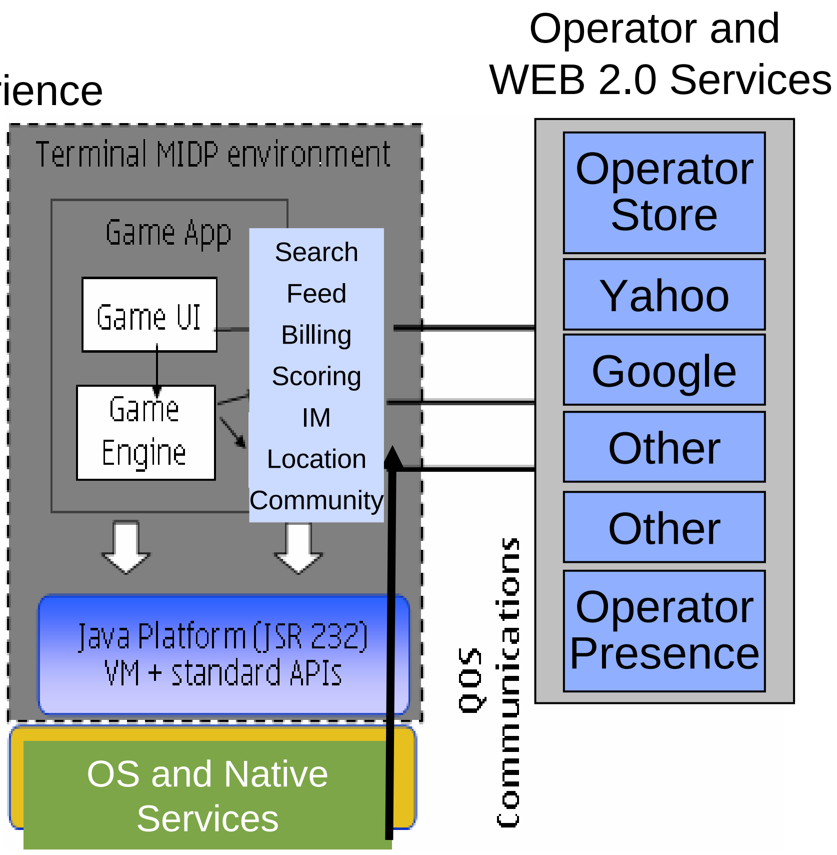
- Go to the Google website
- Take the Google Service API Adapt as a Mobile Java Service with all communication self contained including mobile specific QOS
- Publish the Service API to developers

- Result: The Google service would be instantly available to be embedded in any application running in the Java environment, with any customized UI the application designer wants
- Cost… several hours of skilled programmer time
- Benefit…Huge market advantage to evolve and react at WEB speed

Next Generation Mobile Java Technology… Innovation in Assembly… “Mobile Mash-ups”

A mobile mash-up is a value added service that seamlessly combines services from more than one source into an integrated experience for the Mobile Java platform end application developer

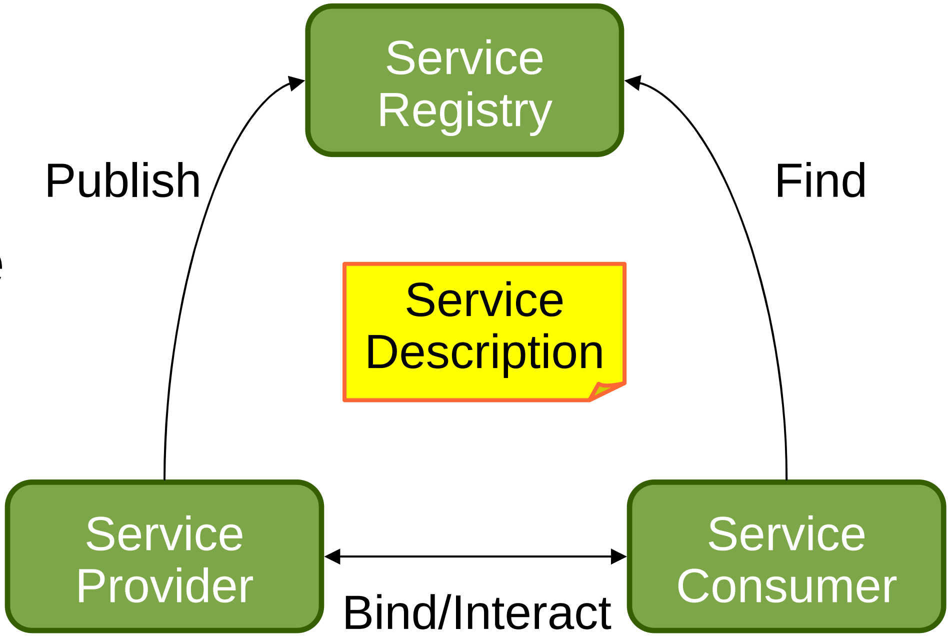
- End developer focuses on application experience
- Operator adds value by providing Mashup services which hide the complexity of mobile communications and interactions from the developer
- Services can be used without having to understand all of the various web protocols and XML constructions
- End developer can easily combine with services offered by the terminal; PIM, messaging etc to create innovative applications
- These services will be developed using existing WEB 2.0 and operator network services and no server code will need to be created



Mobile Mash-up

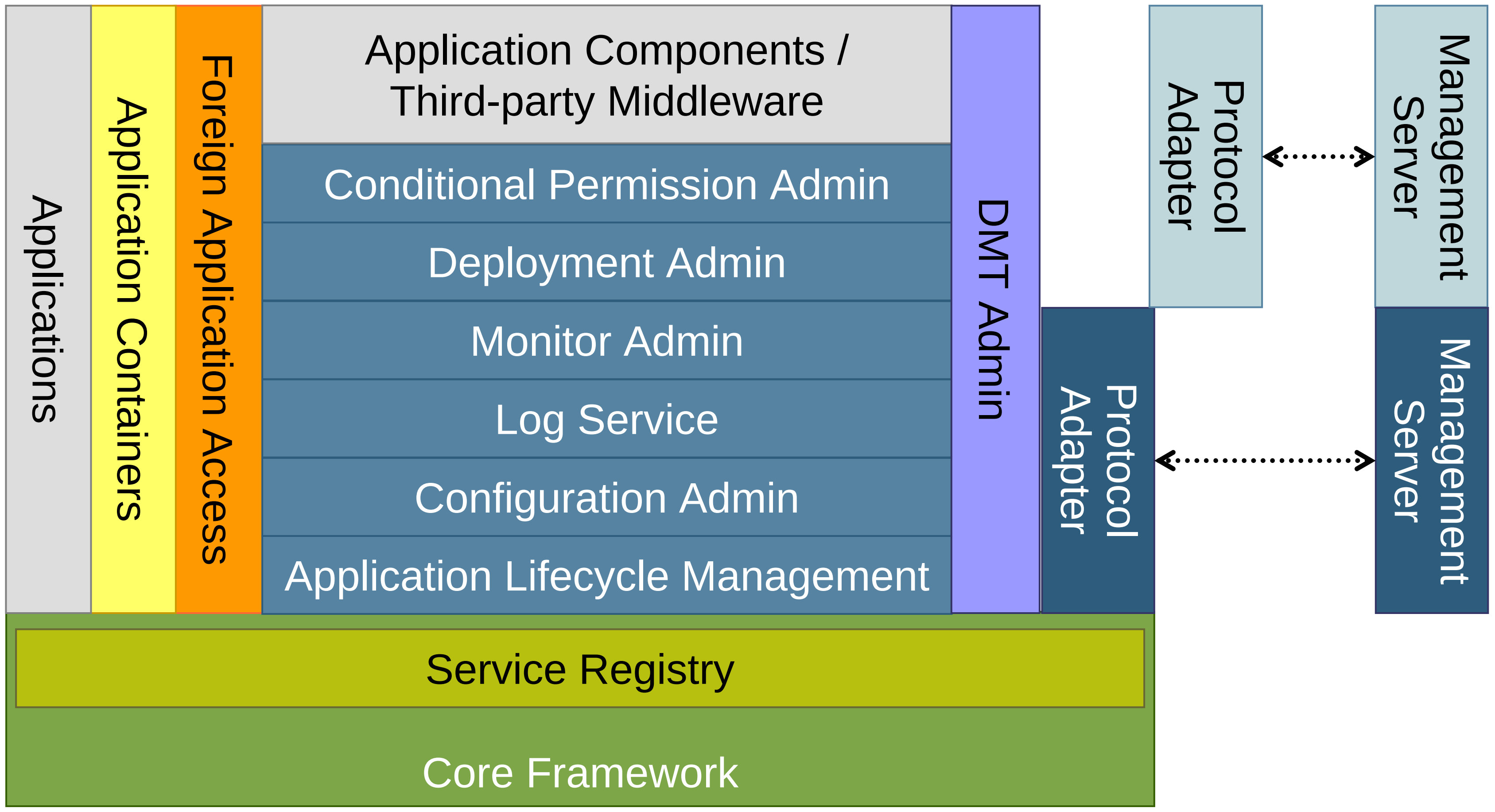
Service-Oriented Architecture

- SOA is often mistaken with web services
- Service orientation is more general and is founded on the concepts of
  - Late non-explicit bindings
  - Functionality description
  - Discovery
- OSGi architecture is an intra-VM realization of SOA concepts

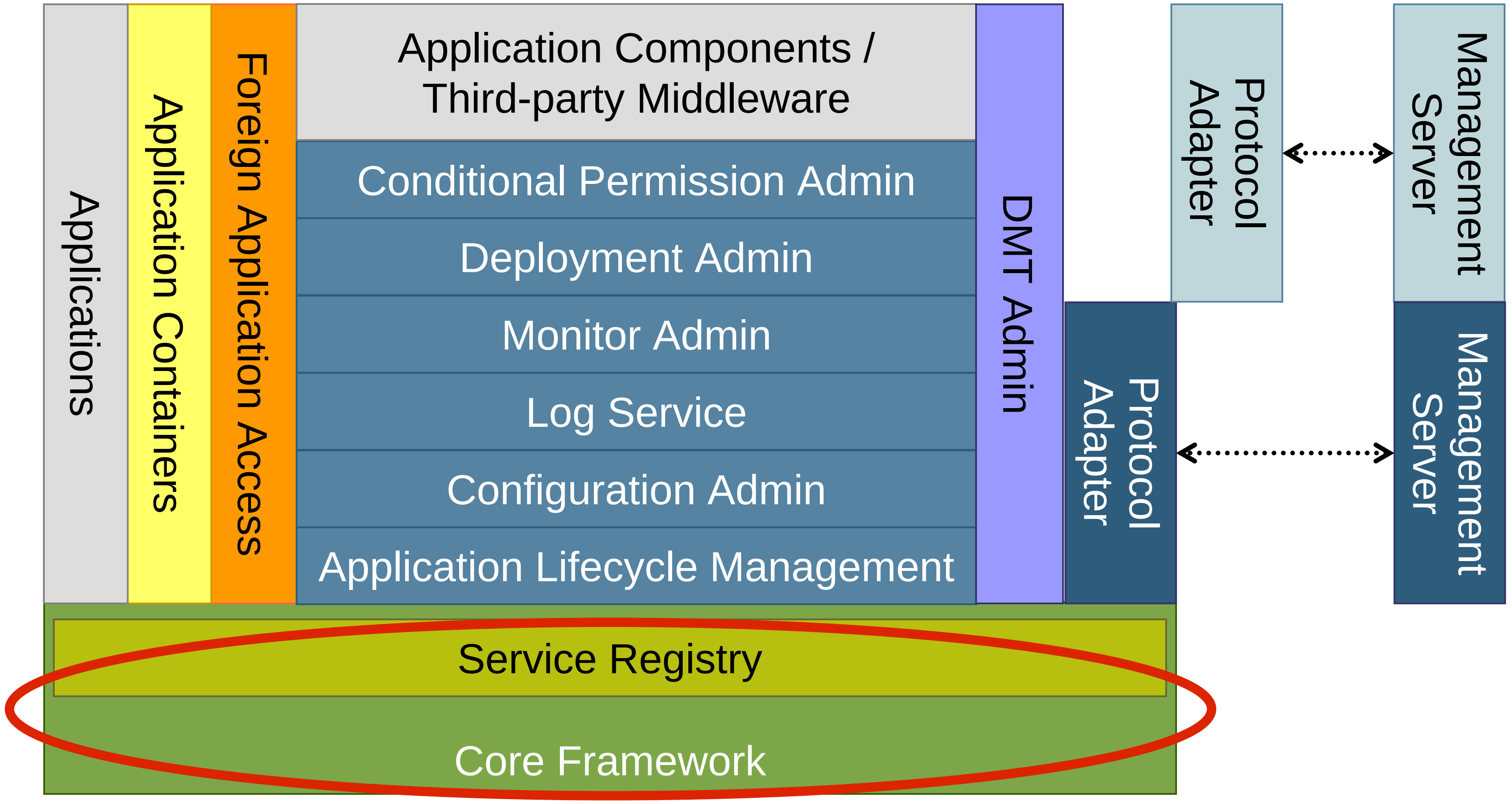


- Simple, scalable architecture
- Self-contained services
- Loose coupling

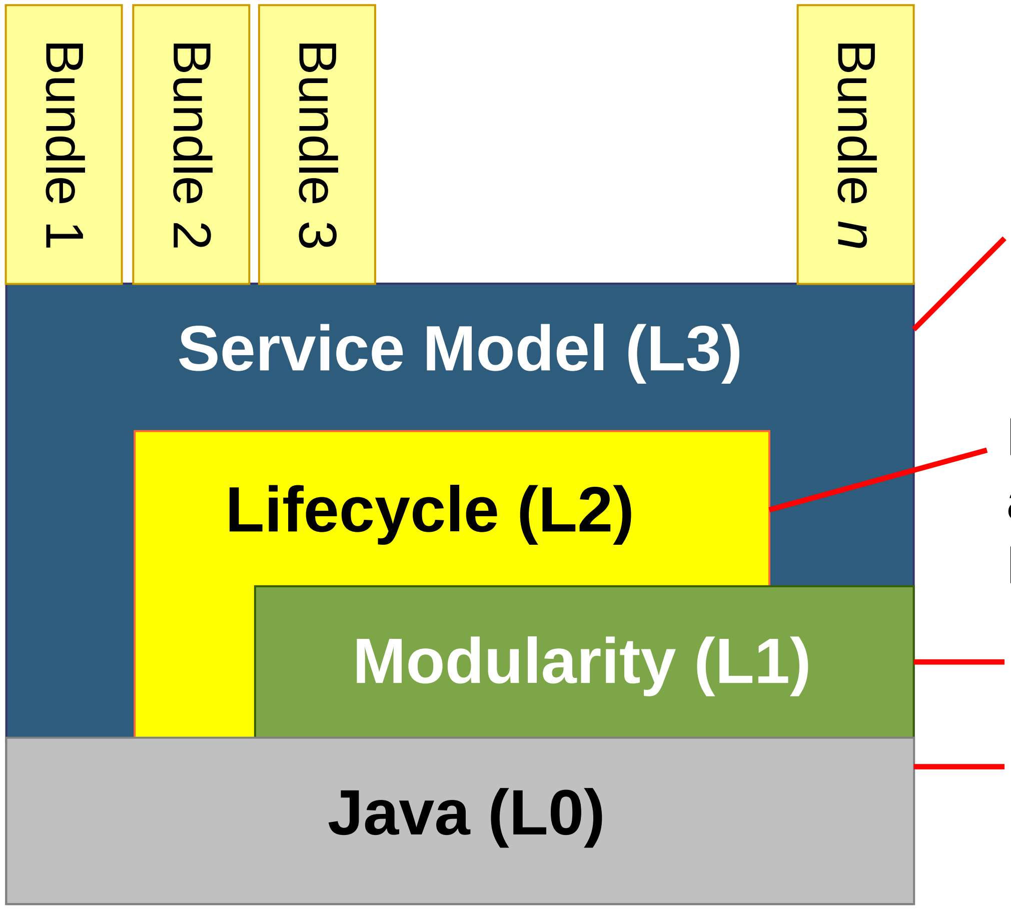
What We Have in the Box—High-level Architecture



Core Framework



JSR 232 Architecture = A Layered Core Framework + Services



Decouples Bundles and Provides a Service Registry

Manages Lifecycle of Bundles and Provides a Bundle Repository

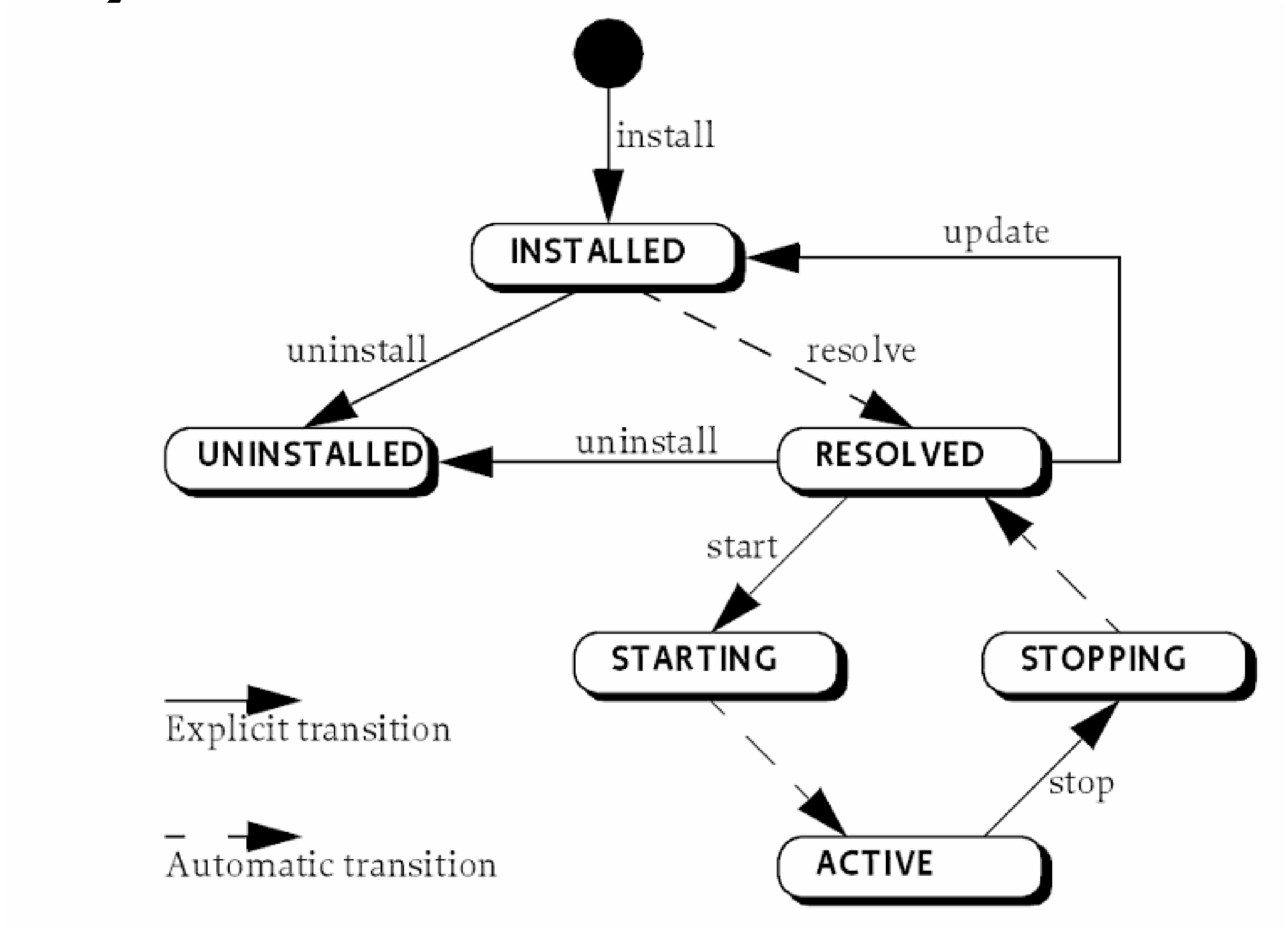
Supports Bundles as Modules

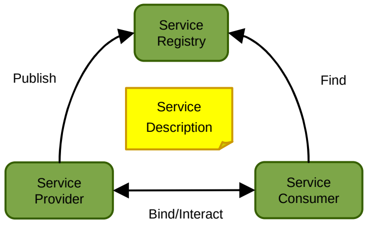
Execution environment:
- CDC/FP
- OSGi/Minimum EE
- Java SE platform

JSR 232 Component Model—Bundles

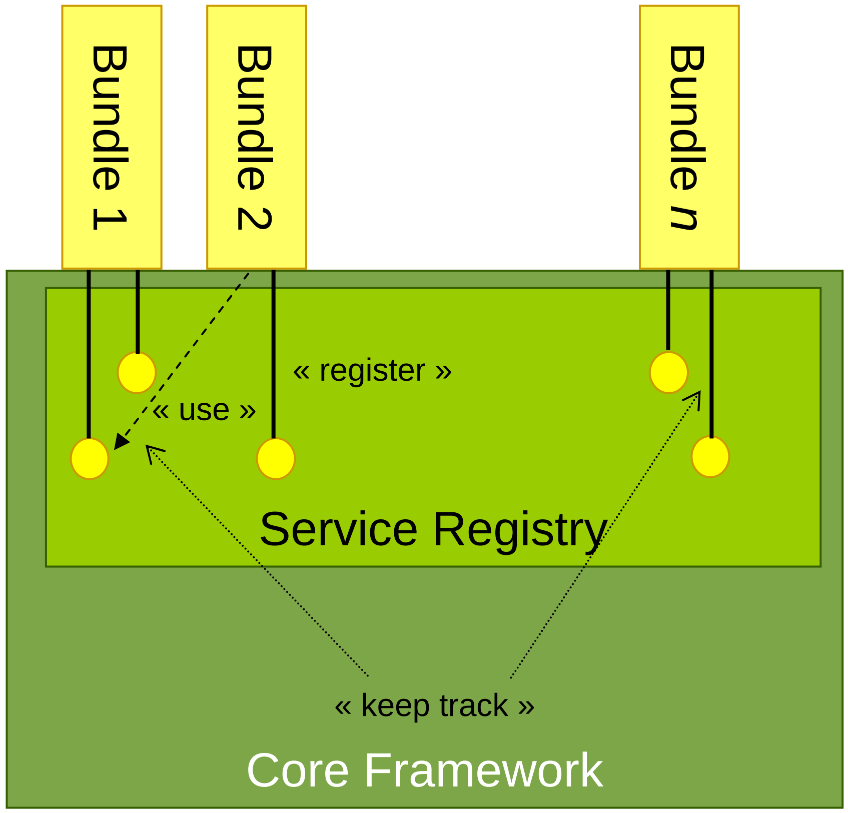
- Bundle is a unit of encapsulation in JSR 232
  - A JAR file containing different resources and metadata
    - Visibility of resources can be public or private
  - Own class loader → each bundle is a separate namespace.
- Metadata is included in the Manifest.mf
  - Bundle identification (name, version)
  - Dependencies on external packages (imports)
  - Provided packages (exports)
  - Information for bundle management (e.g. lifecycle management)
- Can be dynamically installed and removed
- Framework resolves the packages dependencies

Lifecycle of a Bundle



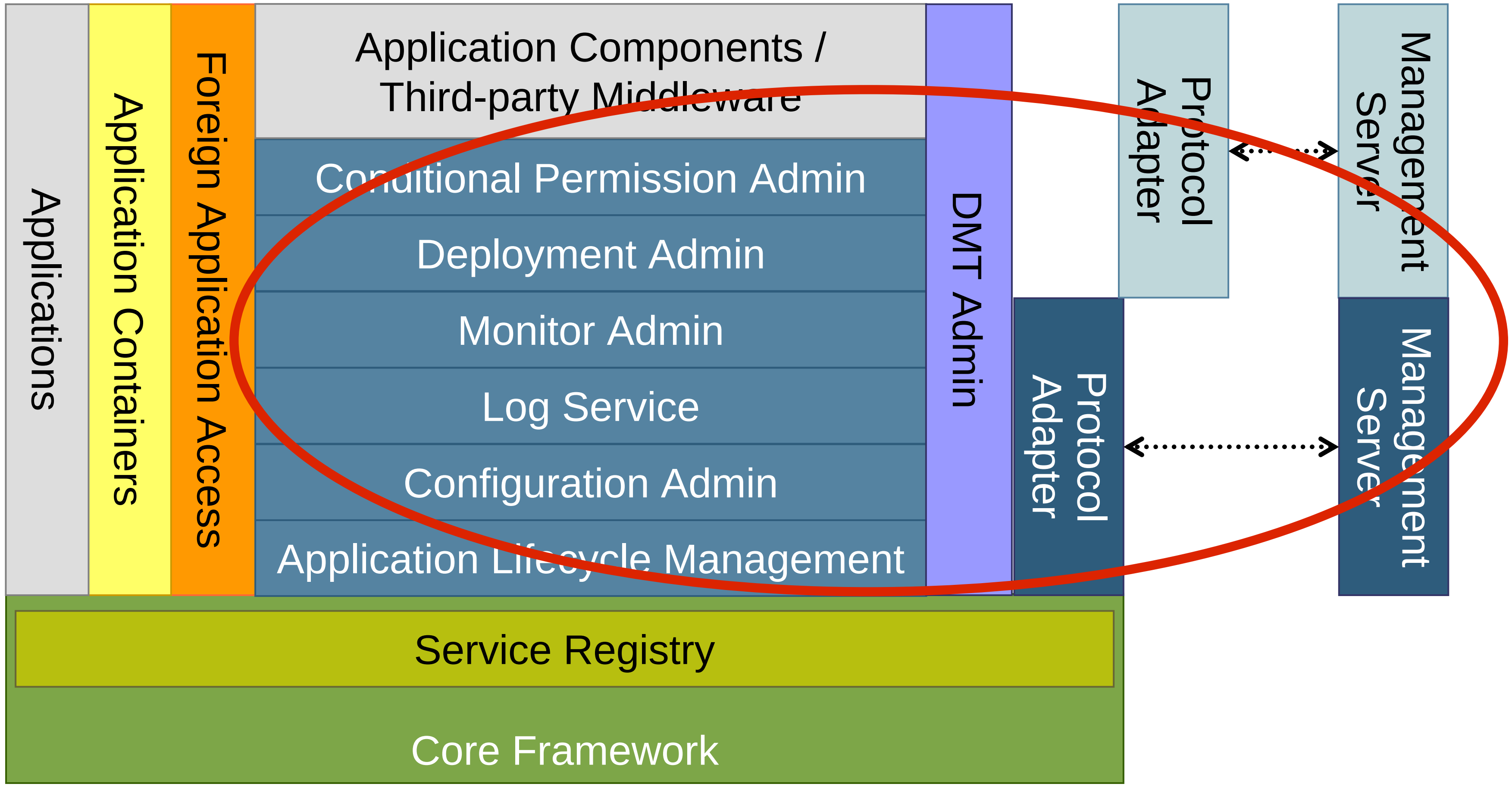


SOA in OSGi—Service Registry

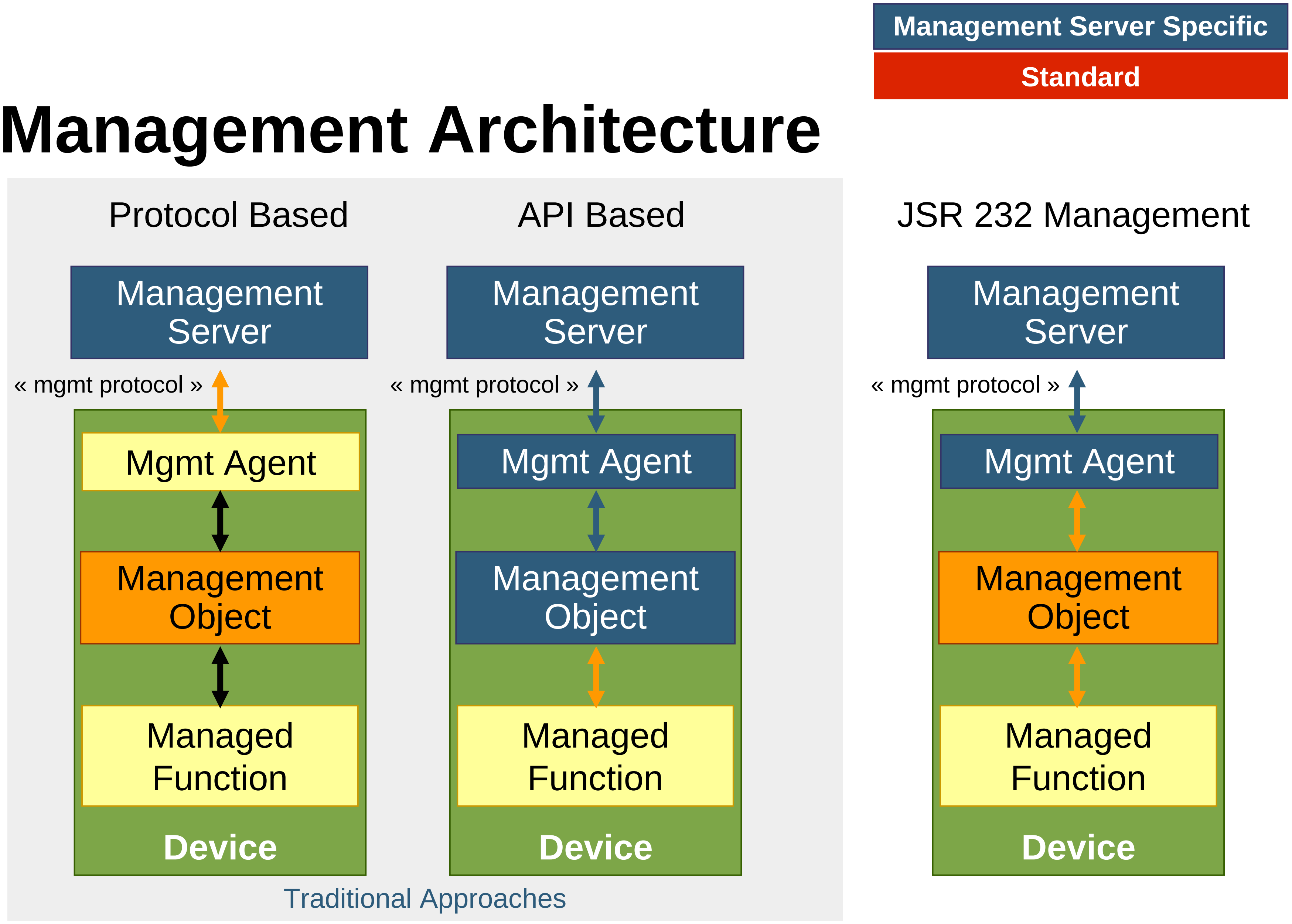


- Bundles register their services
  - Service is defined by a Java interface
  - Arbitrary service attributes
- Consumers look up services by interface
  - Filtering based on service attributes: **`(&(service.vendor=Nokia)(com.nokia.custom.attr=42))`**
- Service is accessed via direct method calls
- Notification mechanism
- Stateless and stateful services
- Framework tracks the usage of services

Manageability



Management Architecture



Management Architecture

- Combines the protocol and API based approaches
  - Protocol-based approach is simpler for management servers
  - API-based approach is easier for application developers
- Management Objects for the most common management services are defined,
  - Format: Based on Device Management Tree model defined in OMA DM
- Plug-in architecture
  - Pluggable management agents enable interoperability with different management servers
    - Support for local management applications as well
  - Pluggable Management Objects to extend DMT

Management Services

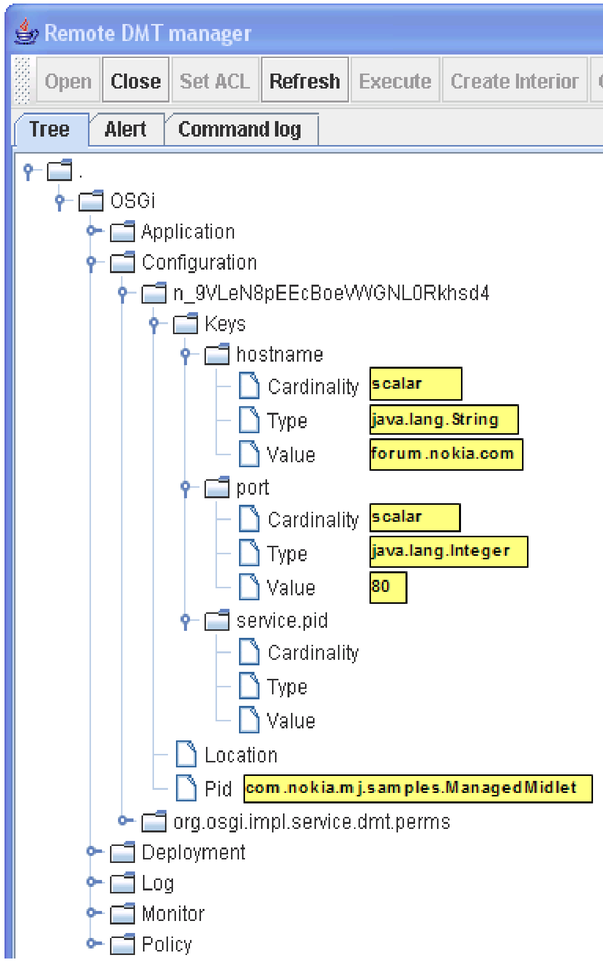
- Log service
  - Creating log entries
  - Log listeners
  - Log search
- Configuration admin
  - Create, update and delete configuration objects
  - Push and pull models
- Monitor admin
  - Publish status variables
    - Different collection methods
  - Monitoring jobs
- Application lifecycle management
  - Start and stop applications
  - Event triggered execution
  - Scheduled execution
- Deployment admin
  - Deployment package format
  - Install, update, uninstall
  - Pluggable resource processors

Configuration Management—Example

Application Developer’s View:

```
private static final String MYPID =
      ManagedMidlet.class.getName();

ApplicationContext context;
Configuration myConfig;

protected void startApp()
                throws MIDletStateChangeException {
  context = Framework.getApplicationContext(this);
  ConfigurationAdmin configAdmin =
        (ConfigurationAdmin)context.locateService(“config”);

  myConfig = this.configAdmin.getConfiguration(MYPID);

  Dictionary config = this.myConfiguration.getProperties();
  hostname = (String)config.get(“hostname”);
  port = (Integer)config.get(“port”);
}
```

Administrator’s View:



```
.
  OSGi
    Application
    Configuration
      n_9VLeN8pEEcBoeVWGNL0Rkhsd4
        Keys
          hostname
            Cardinality   scalar
            Type          java.lang.String
            Value         forum.nokia.com
          port
            Cardinality   scalar
            Type          java.lang.Integer
            Value         80
          service.pid
            Cardinality
            Type
            Value
        Location
        Pid  com.nokia.mj.samples.ManagedMidlet
      org.osgi.impl.service.dmt.perms
    Deployment
    Log
    Monitor
    Policy
```

Managed Application Deployment

- Deployment Package: A packaging format for Mobile Java based application
  - Based on JAR file format
  - Can contain bundles and other resources
  - It is a unit of deployment: Installed, updated and uninstalled atomically
  - Special variant: Fix pack—Contains the delta between versions
- Deployment Admin: Processes the deployment package
  - Plug-in architecture to customize deployment process
    - Plug-ins called Resource Processors
  - A standard Resource Processor: Autoconf

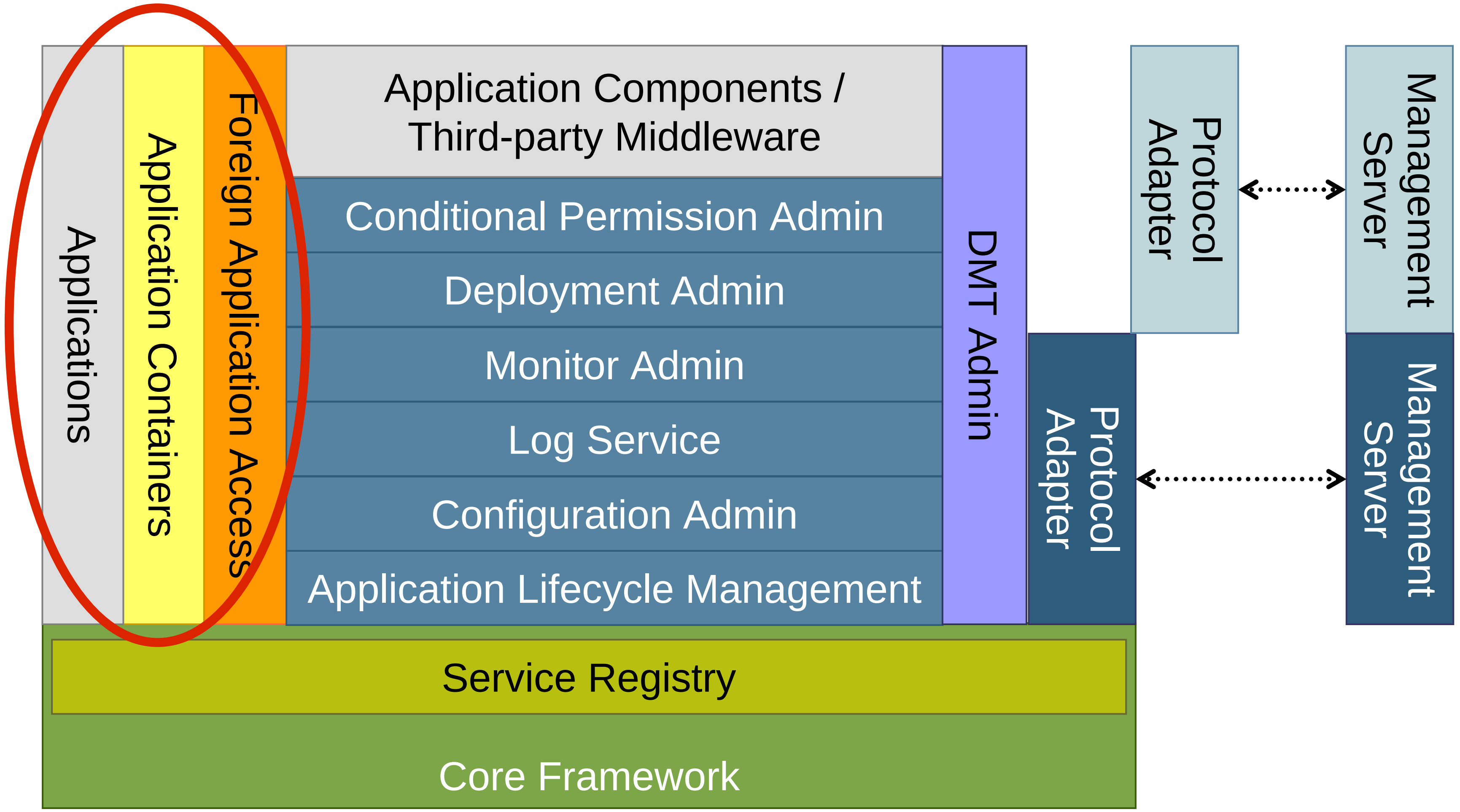
Auto Configuration

- Autoconf script is an XML resource in a DP
  - RP: `org.osgi.deployment.rp.autoconf`
- Structural description of configuration data
  - Name, type, human readable description of configuration item
- The definition of configuration objects
  - To be added to Configuration Admin
- The metadata can be used to automatically create a user-friendly configuration UI

Auto Configuration—Example

```
<?xml version="1.0" encoding="UTF-8"?>
<MetaData xmlns="http://www.osgi.org/xmlns/metatype/v1.0.0“ >
   <OCD name="ManagedMidlet Configuration" id="MidletConfiguration">
      <AD id="hostname" type=“java.lang.String" name=”Host name”/>
      <AD id="port" type=“java.lang.Integer" name=”Port Number”/>
   </OCD>

   <Designate pid="com.nokia.mj.sample.ManagedMidlet"
              bundle="osgi-dp:ManagedApplication1">
      <Object ocdref="MidletConfiguration">
         <Attribute adref="hostname"  content="forum.nokia.com"/>
         <Attribute adref="port" content=“80"/>
      </Object>
   </Designate>
</MetaData>
```

Application Access



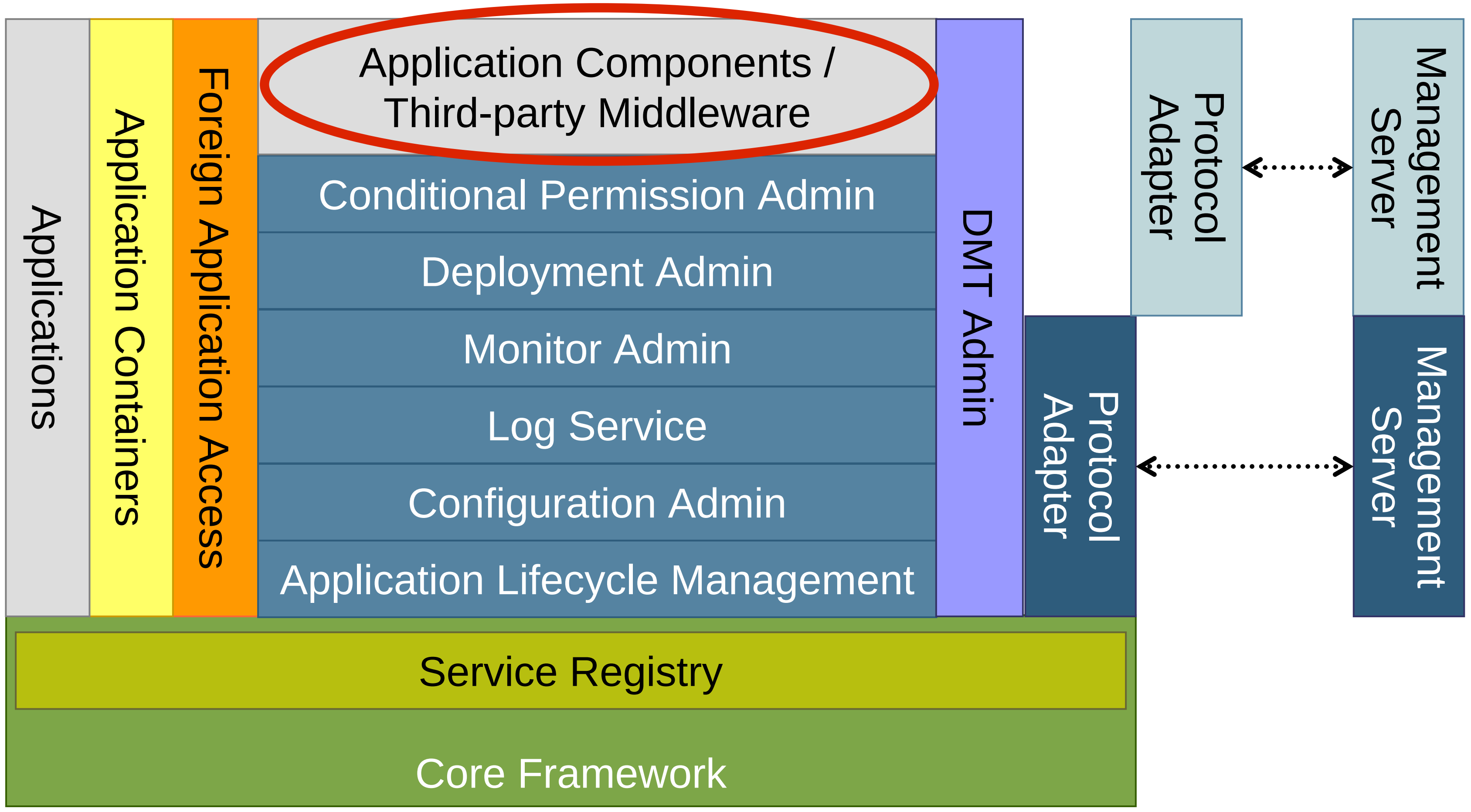
Foreign Application Access

- Enables access to the platform for applications
  - Supports existing application models like MIDP
  - Application must have managed lifecycle
  - Application must be packaged as a JAR
- Application JAR installed to as bundles
  - Application container recognizes these “bundles” and provides the necessary environment
- Applications can import shared packages
  - Declare dependencies using Import-Package
- Applications can register and use services
  - Service dependencies declared in XML descriptor

Example Application Descriptor

```
<?xml version="1.0" ?>
<descriptor xmlns="http://www.osgi.org/xmlns/app/v1.0.0">
   <application class="com.nokia.mj.sample.ManagedMidlet">
      <reference
         name="log“
         interface="org.osgi.service.log.LogService"/>

      <reference
         name="config“
         interface="org.osgi.service.cm.ConfigurationAdmin"/>
   </application>
</descriptor>
```

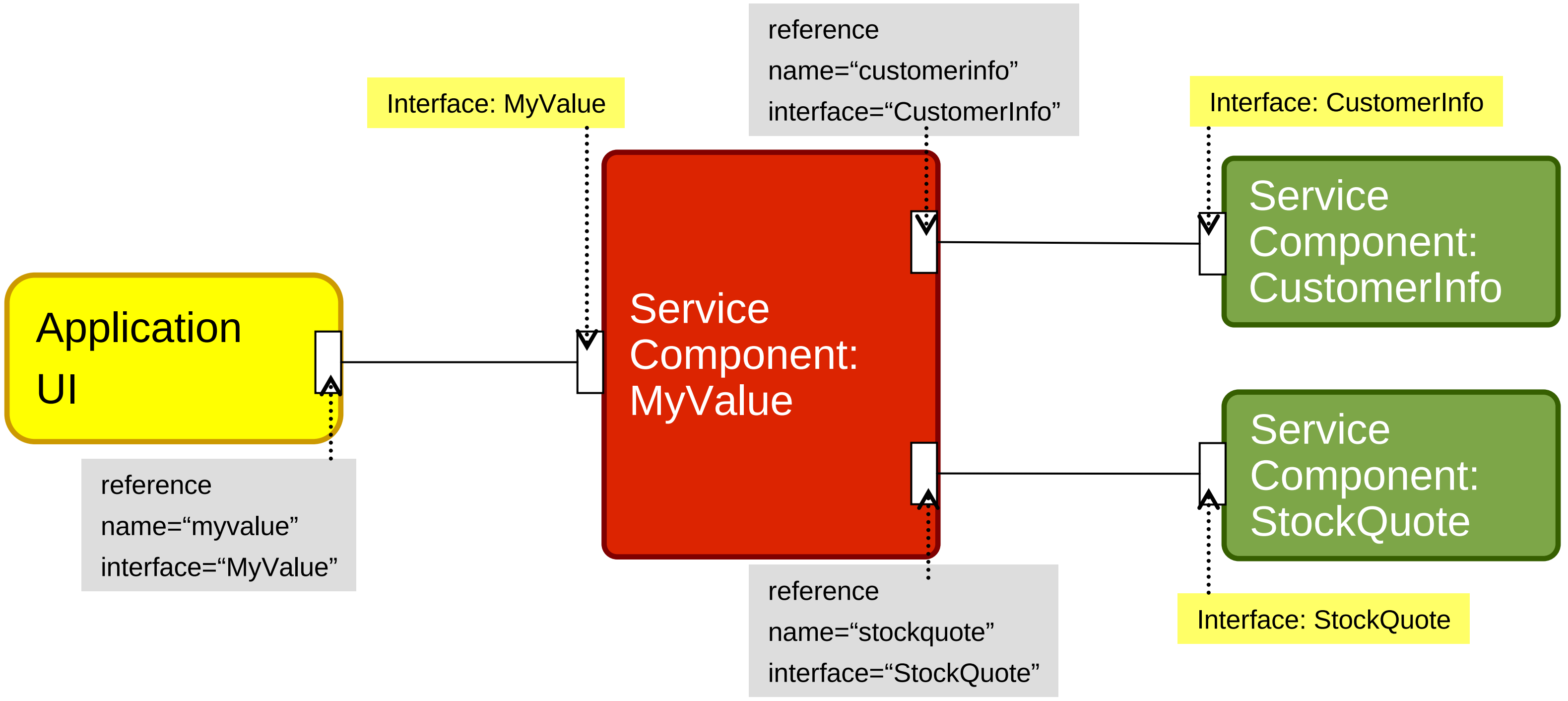
Component Development



Components

- Bundles can share packages and provide services
- Shared packages
  - Class libraries
  - Loaded only once, static elements are shared
- Services—Like daemon processes
  - Autonomous operation
  - Own execution context
  - Lifecycle independent from clients lifecycle
- Under the control of Java 2 based permissions—Fine-grained access control
  - Exporting and importing packages—On package level
  - Service lookup and registration—On service level

Creating Service Components



- A simple stock-value calculator; Service **`MyValue`** calculates the actual values of a persons stock
  - Stock price is obtained from **`StockQuote`** service
  - The number and type of owned stocks read from the **`CustomerInfo`** service—one customer owns one type of stocks

Creating Service Components

CustomerInfo.java

```
package com.isv.service.customerinfo;

public interface CustomerInfo {
  public Customer getCustomer(String customerID););
}
```

Customer.java

```
package com.isv.service.customerinfo;

public interface Customer {
   public String getCustNo();
   public String getFirstName();
   public String getLastName();
   public String getSymbol();
   public int    getNumShares();
}
```

StockQuote.java

```
package com.isv.service.stockquote;

public interface StockQuote {
  public float getQuote(String symbol);
}
```

Creating Service Components— The Service Interface

MyValue.java

```
package com.isv.process.myvalue;

public interface MyValue {
   public float getMyValue(String customerID);
}
```

Bundle manifest:

```
Manifest-Version: 1.0
Bundle-SymbolicName: com.isv.service.myvalue
Bundle-Verison: 1.0.0
Bundle-Vendor: Acme Inc.
Export-Package: com.isv.service.myvalue; specification-
 version=1.0
```

- Service defined as an interface
- Bundle exports the package of service interface definition
    - Specification version is used during dependency resolution
- Implementation will be in a different package
    - Implementation package is not exported to hide it from other bundles

Creating Service Components—Implementing Business Logic

```
package com.isv.process.myvalue.impl;

public class MyValueImpl implements MyValue {…

protected void activate(ComponentContext context) {
   this.context = context;
}

public float getMyValue(String customerID) {
   CustomerInfo cInfo =
       (CustomerInfo)context.locateService("customerInfo");

   Customer customer = cInfo.getCustomer(customerID);

   StockQuote sQuote =
       (StockQuote)context.locateService("stockQuote");

   float quote =  sQuote.getQuote(customer.getSymbol());
   return (quote * customer.getNumShares());
}
```

- The declared policy/cardinality ensures that `locateService` returns valid service objects

Creating Service Components—Component Descriptor

```
<?xml version="1.0" encoding="UTF-8"?>
<scr:component name="com.isv.process.myvalue.MyValue"
     xmlns:scr="http://www.osgi.org/xmlns/scr/v1.0.0">

  <implementation
        class="com.isv.process.myvalue.impl.MyValueImpl"/>

  <service>
    <provide interface="com.isv.process.myvalue.MyValue"/>
  </service>

  <reference name="customerInfo"
     interface="com.isv.service.customerinfo.CustomerInfo"
     cardinality="1..1"
     policy="static"/>

  <reference name="stockQuote"
     interface="com.isv.service.stockquote.StockQuote"
     cardinality="1..1"
     policy="static"/>
</scr:component>
```

- Component descriptor specifies the implementation class, provided services, service dependencies

Creating Service Components— Creating the Bundle

Additions to manifest:

```
…
Import-Package: org.osgi.framework; specification-version=1,
 org.osgi.service.component,
 com.isv.service.customerinfo,
 com.isv.service.stockquote

Service-Component: com/isv/process/myvalue/impl/myvalue.xml
```

- Import the packages for the interface of the used services
- Component descriptor file is listed in Service-Component header
- Package it in a JAR

```
META-INF\MANIFEST.MF
com/isv/process/myvalue/MyValue.class
com/isv/process/myvalue/impl/MyValueImpl.class
com/isv/process/myvalue/impl/myvalue.xml
```

Summary…

The 232 Platform

- Provides an intra-VM realization of SOA concepts, including
  - Component model
  - Service model, with efficient, direct API call based interaction model
- Enables a new, modular architecture for mobile applications
- Enables extending the platform with new functionality—Mobile middleware
- Supports the management of components through their whole lifecycle

Meet Us!

- See our JSR-232 demos at Nokia booth; Booth number 1114
- 20:30 today: BOF-0157: The OSGi Service Architecture, From Embedded to Enterprise

Mobile Java™ Technology JSR 232 Architecture and Benefits

Jon Bostrom

Director, Architecture

Nokia
http://www.nokia.com/

Gábor Pécsy

Software Technology Specialist

TS-3757