

# Il framework Open Services Gateway initiative (OSGi)

*Domotics Lab – ISTI (CNR)*  
*cesare.concordia@isti.cnr.it*

# OSGi: gli eventi

- Il framework OSGi espone diversi eventi attraverso l'interfaccia `BundleContext`:
  - *ServiceEvent* segnala la registrazione e la cancellazione dei servizi, nonché la variazione delle proprietà degli stessi
    - Sincroni
  - *BundleEvent* segnala la variazione nel lyfe cycle dei bundle presenti nel framework
  - *FrameworkEvent* segnala se il framework è attivo e gli errori che si verificano

# OSGi: i listener

- A ciascun tipo di evento è associato un listener
  - **ServiceListener**
    - intercetta gli eventi relativi alla registrazione/deregistrazione dei service object
  - **BundleListener**
    - intercetta le variazioni di stato dei bundle
  - **FrameworkListener**
- Un bundle che usa un servizio dovrebbe registrare un **ServiceListener** per poterlo “monitorare” prevedendo delle procedure di gestione degli eventi

# OSGi: ServiceFactory

- In alcuni casi la configurazione di un servizio dipende dal bundle che lo usa
- In questi casi il service object viene implementato come *Service Factory*
- Il framework OSGi crea una istanza del service object per ciascuna invocazione
- Una Service Factory è registrata nella stessa maniera di un service object: `BundleContext.registerService(...)`
- Il fatto che un servizio sia implementato da un service object o da una service factory è trasparente al client.

# OSGi: ServiceFactory Object

- ServiceFactory Object (oggetti ServiceFactory)
- Gli oggetti ServiceFactory sono utili per gestire le dipendenze non esplicitamente gestite dal framework
- Il framework OSGi notifica all'oggetto ServiceFactory l'evento disattivazione (stop) del bundle che invoca il servizio
- In caso di disattivazione del chiamante l'oggetto ServiceFactory può eseguire delle operazioni di gestione

# OSGi: ServiceFactory Object

- Un oggetto ServiceFactory deve implementare l'interfaccia `org.osgi.framework.ServiceFactory`
- `getService(Bundle, ServiceRegistration)`
  - Il framework passa all'oggetto ServiceFactory il bundle del chiamante
  - Il framework controlla che il service object restituito sia del tipo corretto
- `ungetService(Bundle, ServiceRegistration, Object)`
  - Il framework notifica l'evento all'oggetto Servicefactory
  - *garbage collection*

# OSGi : Esempio ServiceFactory

```
public class Activator implements BundleActivator {
...
private static class DictionaryImpl implements
ServiceFactory{
public Object getService(Bundle bnd, ServiceRegistration reg) {

    Dictionary myDict=bundle.getHeaders();
    String lang=(String) myDict.get("Bundle-Language");
    if (lang !=null){

        if (lang.equalsIgnoreCase("IT"))
            return new ItalianDictionary();
        if (lang.equalsIgnoreCase("EN"))
            return new EnglishDictionary();
    }
    // default
    return new EnglishDictionary();
}

public void ungetService(Bundle bundle, ServiceRegistration
reg, Object service) {
    // gestione dell'evento
}
}
}
...
```

# OSGi : Esempio ServiceFactory

**Bundle-Activator:** tutorial.example3IT.Activator

**Import-Package:** tutorial.example2c.service

**Import-Service:**

tutorial.example2c.service.DictionaryService

**Bundle-Name:** Client per il dizionario di italiano

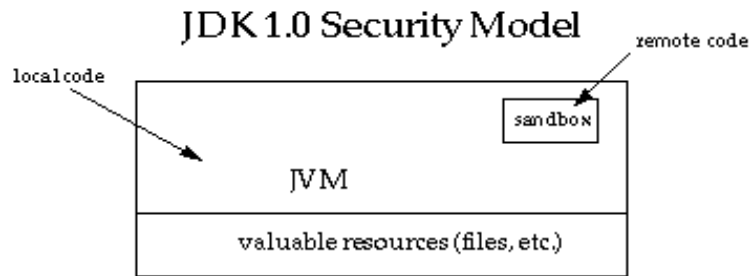
**Bundle-Description:** A bundle that uses the Italian dictionary service if it finds it at startup

**Bundle-Vendor:** Cesare Concordia

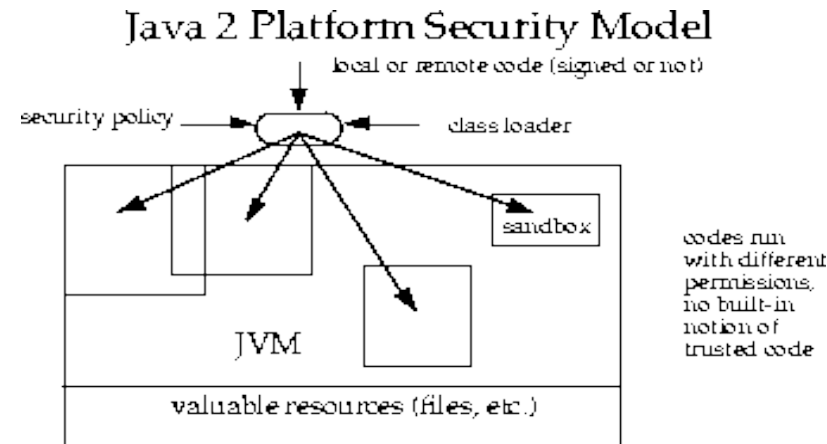
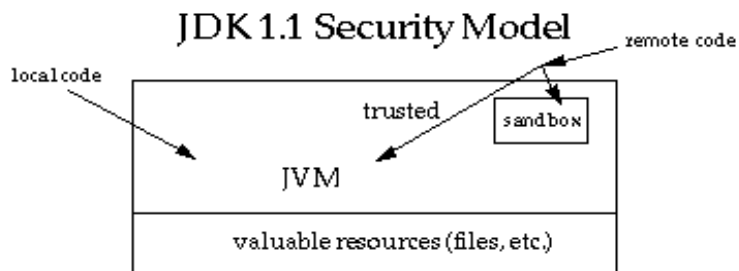
**Bundle-Version:** 1.0.0

**Bundle-Language:** IT

# La sicurezza in Java



Signed Applets



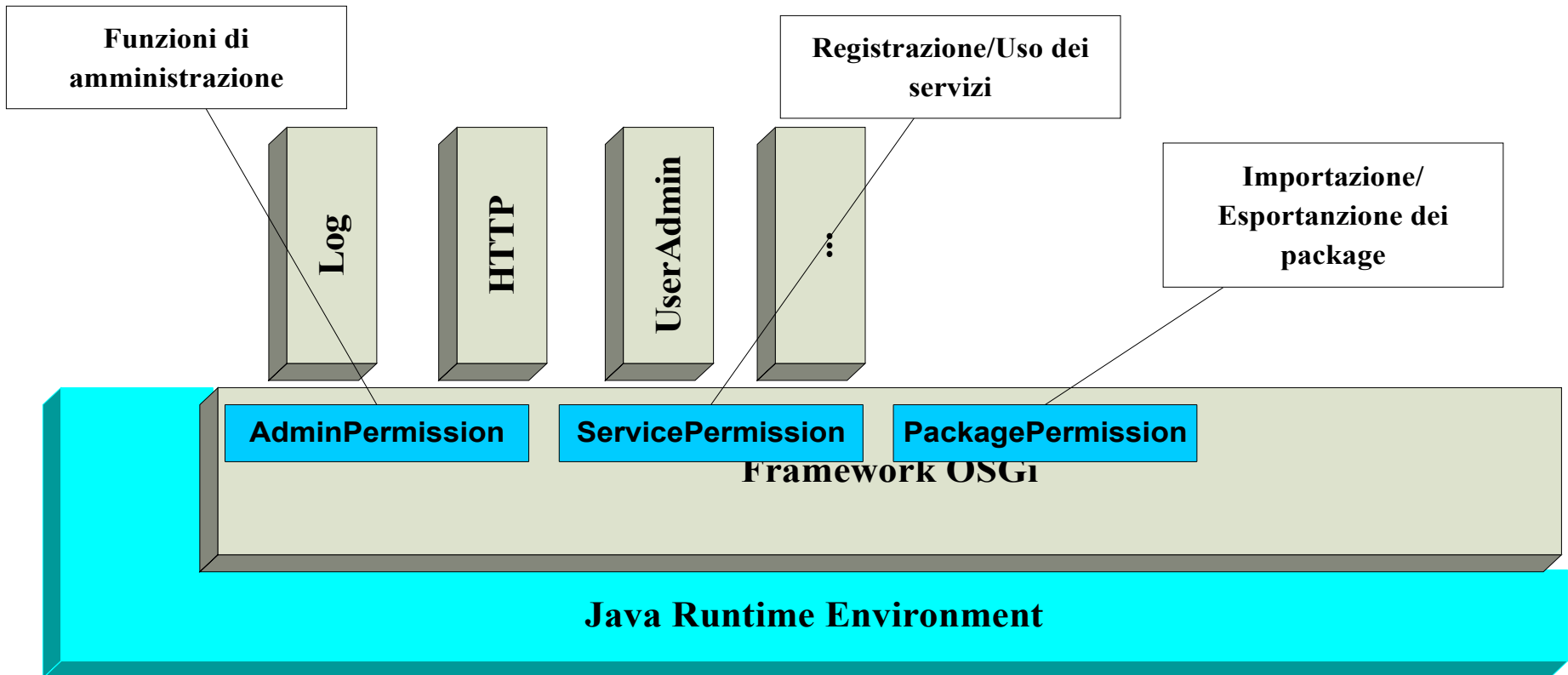
# La sicurezza in Java

- Non esiste più la differenza tra codice locale e codice remoto
- *Typed permission*, es:
  - `perm=new java.io.FilePermission("/tmp/abc","read");`
- Si possono definire “estendendo” la classe `java.io.Permission` o una delle sue sottoclassi
  - `java.io.BasicPermission`
- Occorre definire accuratamente la *security policy* delle applicazioni distribuite realizzate in Java
  - `{java.home}/lib/security/java.policy, java.security`

# OSGi: la sicurezza

- Basata sulle specifiche di Java 2
- Definizione standard dei permessi (permissions)
  - `AdminPermission` controlla l'accesso alle funzioni amministrative del framework
  - `ServicePermission` controlla la registrazione e l'accesso ai servizi
  - `PackagePermission` per il controllo dell'importazione/esportazione dei package
- I servizi possono definire dei propri permessi

# OSGi: la sicurezza



# OSGi: la sicurezza

- **ServicePermission**
  - **Interface Name** il nome dell'interfaccia a cui è assegnato il permission
  - **Action**
    - **GET** permesso di ottenere il servizio
    - **REGISTER** permesso di registrare il servizio
- Valido sia per i **ServiceReference Object** sia per i **Service Object**
- Usato per filtrare gli eventi notificati al **Service Listener** ed i metodi che permettono di accedere ai servizi dei bundle quali:  
**Bundle.getRegisteredServices** o  
**Bundle.getServicesInUse**
- Il framework deve garantire che un bundle non possa individuare un servizio se non ha il permesso di farlo

# OSGi: la sicurezza

- **PackagePermission**
  - **Package Name** il nome del package su cui è assegnato il permission
  - **Action**
    - **IMPORT**
    - **EXPORT**
- Se un bundle ha il permesso di esportare un package ha automaticamente anche il permesso di importarlo
- Un PackagePermission con parametri (**\***, **EXPORT**) garantisce al titolare il diritto di esportare ed importare tutti i package

# OSGi : example.policy

```
grant codeBase "file:bundle/shellgui.jar" {  
  permission org.osgi.framework.ServicePermission  
  "org.ungoverned.osgi.bundle.shellgui.Plugin", "get";  
};  
grant codeBase "file:bundle/permissionmanager.jar" {  
  permission java.security.AllPermission;  
};  
grant codeBase "file:${lib.dir}/osgi.jar" {  
  permission java.security.AllPermission;  
};  
grant {  
  permission java.io.FilePermission  
  "${user.home}${file.separator}.oscar${file.separator}-", "read,  
write, delete";  
  permission org.osgi.framework.PackagePermission "*",  
  "EXPORT";  
};
```

# OSGi: declarative service model

- La natura dinamica dei servizi deve essere gestita con attenzione dal programmatore
- I servizi dichiarativi semplificano in parte la gestione dei servizi
- Le dipendenze sono dichiarate in file xml nei quali è possibile specificare:
  - Le dipendenze opzionali dai servizi
  - La registrazione opzionale dei servizi da parte di un bundle
  - L'inizializzazione *lazy* di un bundle
  - ...

# OSGi: declarative service model

- Un componente può essere una classe qualsiasi.
- I metodi *activate* e *deactivate* sono chiamati per attivare/disattivare il componente

```
<?xml version="1.0" encoding="utf-8" ?>
<component
  name="aQute.tutorial.component.World">
  <implementation
    class="aQute.tutorial.component.World"/>
  <reference name="log"
    interface="org.osgi.service.log.LogService"
    bind="setLog" />
</component>
```

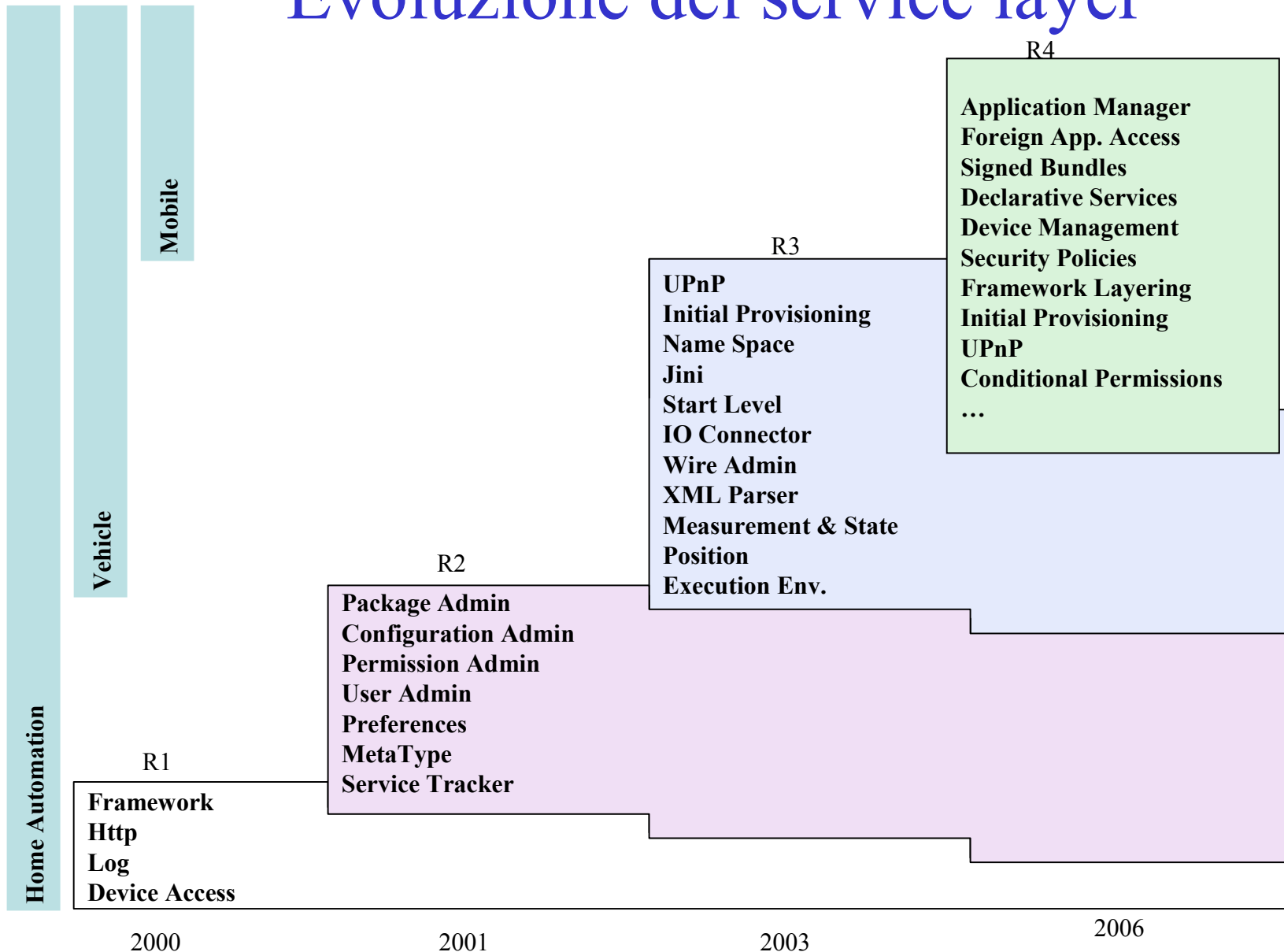
```
package aQute.tutorial.component;
import org.osgi.service.component.*;
import org.osgi.service.log.LogService;

public class World {
    LogService      log;
    protected void activate(
        ComponentContext context) {
        log.log(LogService.LOG_INFO,
            "Hello World"); }
    protected void deactivate(
        ComponentContext context) {
        log.log(LogService.LOG_INFO,
            "Goodbye World"); }
    public void setLog(LogService log) {
        this.log = log; }
}
```

# Open Services Gateway initiative (OSGi): Servizi nativi

Domotics Lab – ISTI (CNR)

# Evoluzione del service layer



# OSGi: servizi nativi

- **Permission Administration Service**
  - Usato per gestire l'accesso dei bundle alle risorse del sistema
- **Configuration Administration Service**
  - Permette ad un operatore di definire la configurazione dei deployed bundle
- **Package Administration Service**
  - Gestisce la condivisione di package tra i bundle
- **User Administration Service**
  - Gestione delle autenticazioni e dei profili utente

# OSGi: servizi nativi

- Log Service
  - Composto da due servizi, uno per il logging delle informazioni, l'altro per la ricerca di messaggi registrati in precedenza
- HTTP Service
  - consente l'accesso alle risorse di Internet
- Device Access specification Service
  - La gestione dei dispositivi (discovering, drivers etc)
- XML Parser Service
  - Parser XML implementato usando le API JAXP

# OSGi: servizi nativi

- IO Connector Service
  - Infrastruttura di comunicazione basata su J2ME
- Wire Admin Service
  - Usato per gestire la “wiring topology” della piattaforma

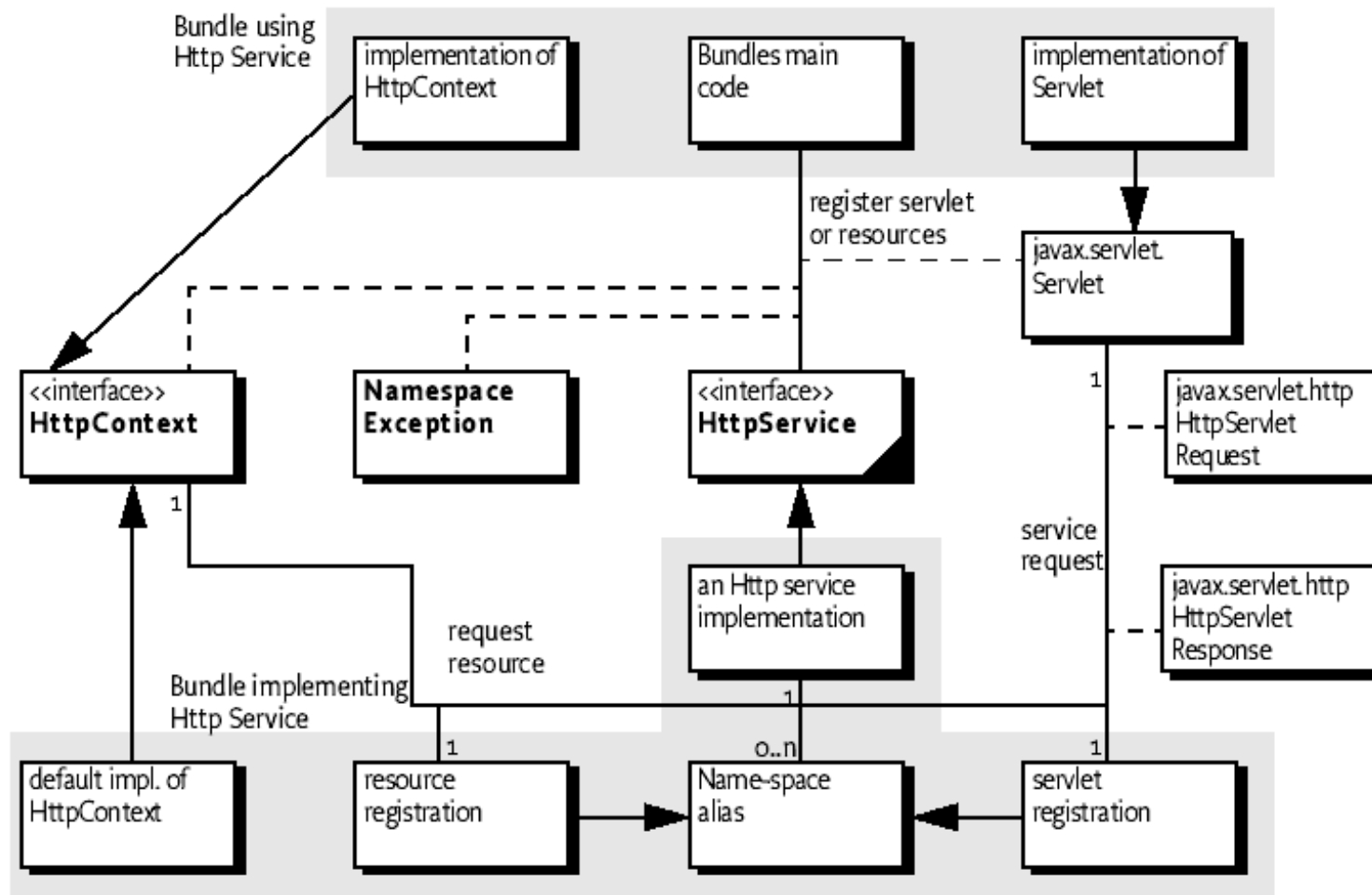
# OSGi: HTTP service

- *“An OSGi Service Platform normally provides users with access to services on the Internet and other networks. This access allows users to remotely retrieve information from, and send control to, services in an OSGi Service Platform using a standard web browser.”* (specifiche OSGi release 3.0)
- HTTP service di OSGi supporta due tecniche standard:
  - *Registering Servlets* cioè Java Objects che implementano le Servlet API
  - *Registering resources* per rendere disponibile ai bundle l’accesso alle risorse Web (file HTML, immagini, etc.)

# OSGi: HTTP service

- Le specifiche OSGi release 3.0 stabiliscono che il framework deve implementare le seguenti interfacce:
  - `HttpContext` consente ai bundle di fornire informazioni per la registrazione di servlet o di risorse
  - `HttpService` – consente ai bundle di registrare o deregistrare dinamicamente risorse o servlet
  - `NamespaceException` - generata in caso di errore in fase di registrazione di una servlet o di una risorsa

# OSGi: HTTP Service



# OSGi: registrazione di un servlet

- Un oggetto di tipo servlet può essere registrato tramite l'interfaccia **HttpService**, usando il metodo:  
`registerServlet(String , javax.servlet.Servlet, Dictionary, HttpContext)`
- I servlet e le risorse registrate condividono lo stesso *namespace*
- L'oggetto **HttpContext** consente al servlet di accedere al framework, un context di default può essere ottenuto con:  
`createDefaultHttpContext()`
  - Il context di default si ottiene anche con il parametro **null**

# OSGi : Es. registrazione di un servlet

```
Hashtable initparams = new Hashtable();  
initparams.put( "name", "testServlet" );  
  
Servlet myServlet = new HttpServlet() {  
    String name = "<not set>";  
    public void init(ServletConfig config) {  
        this.name = (String)config.getInitParameter( "name" );  
    }  
    public void doGet(HttpServletRequest req,  
                        HttpServletResponse rsp) throws IOException{  
        rsp.setContentType( "text/plain" );  
        req.getWriter().println( this.name );  
    }  
};  
  
getHttpService().registerServlet("/myServletAlias", myServlet,  
initparams, null);  
//myServletAlias è stato registrato  
//il suo metodo init(ServletConfig config) viene chiamato.  
//Le invocazioni remote all'URL "http://web.site.tld/myServletAlias"  
//vengono passate al servlet  
...  
getHttpService().unregister("/myServletAlias");  
// myServletAlias viene de-registrato ed il metodo  
//destroy() viene chiamato
```

# OSGi: registrazione di una risorsa

- Una risorsa è un file contenente immagini, formattazione HTML, suoni, applet etc
- Una risorsa può essere registrata tramite l'interfaccia **HttpService**, usando il metodo:  
`registerResources(String, String, HttpContext)`
- L'oggetto `HttpContext` consente di restituire il tipo Multipurpose Internet Mail Extension (MIME) della risorsa