

# Il framework Open Services Gateway initiative (OSGi)

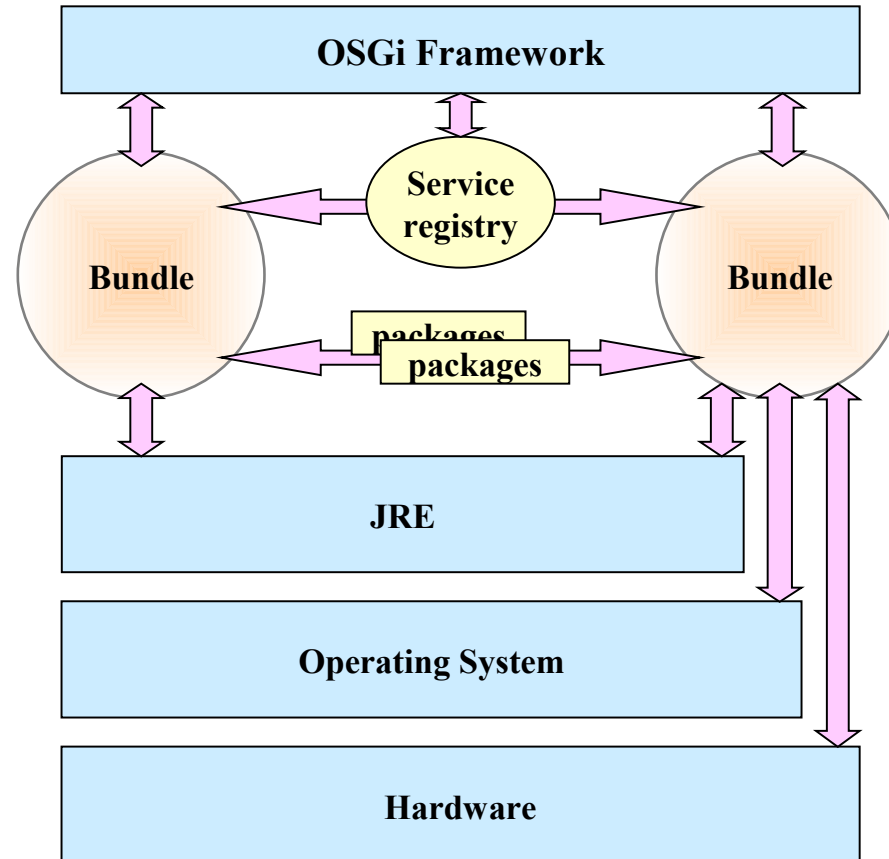
*Domotics Lab – ISTI (CNR)*

*cesare.concordia@isti.cnr.it*

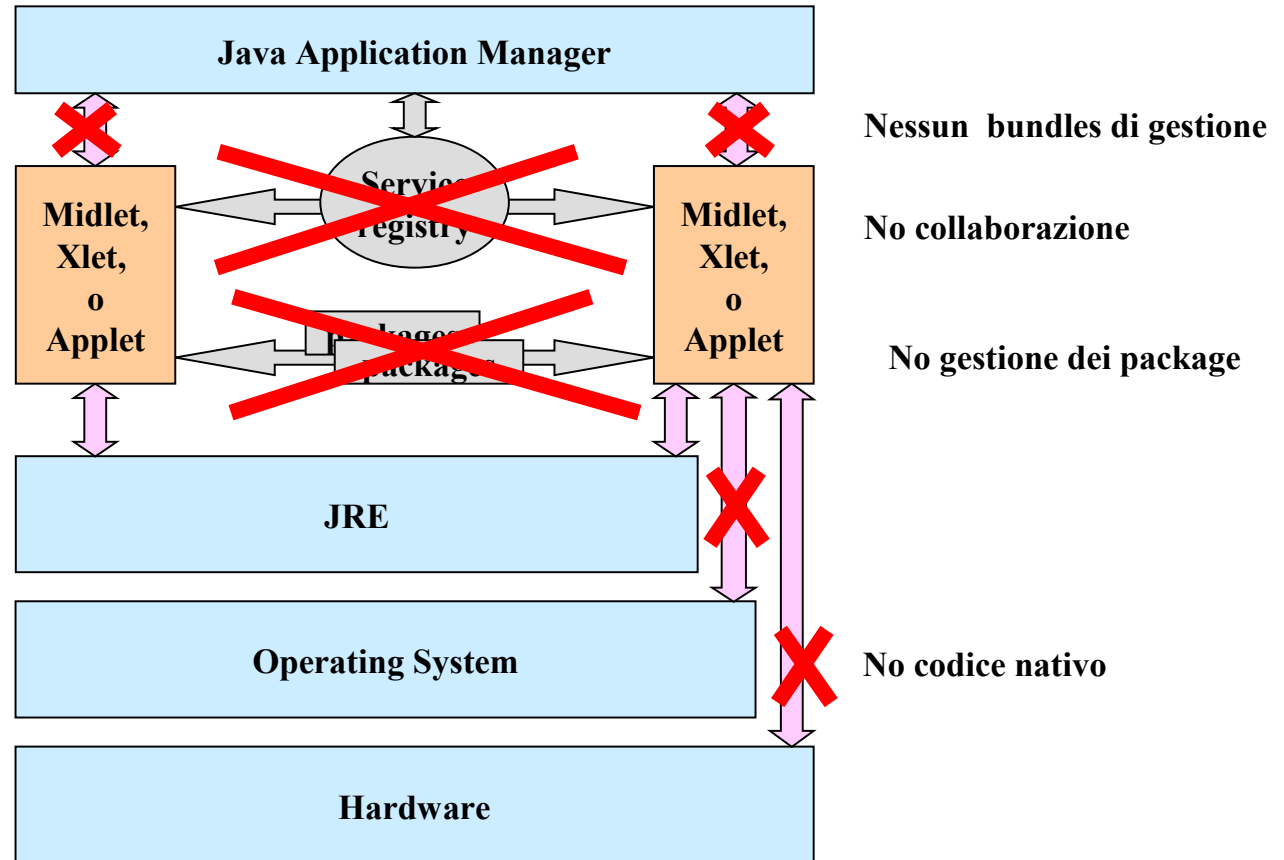
# OSGi: modello collaborativo

- I bundle possono interagire e collaborare attraverso:
  - I servizi
  - La condivisione di package
- Un registro permette ai bundle di trovare e tracciare i servizi
- La collaborazione è gestita dal framework

# OSGi: modello collaborativo



# OSGi: modello collaborativo



# OSGi: gestione del classpath

- Le applicazioni java sono composte da classi organizzate in packages
- Un programma java cerca le classi di cui ha bisogno in un insieme di directory o file jar, questo insieme è definito dal *classpath*
- Nel framework OSGi ogni bundle viene gestito da un diverso class loader e può definire il suo classpath nel proprio file manifest

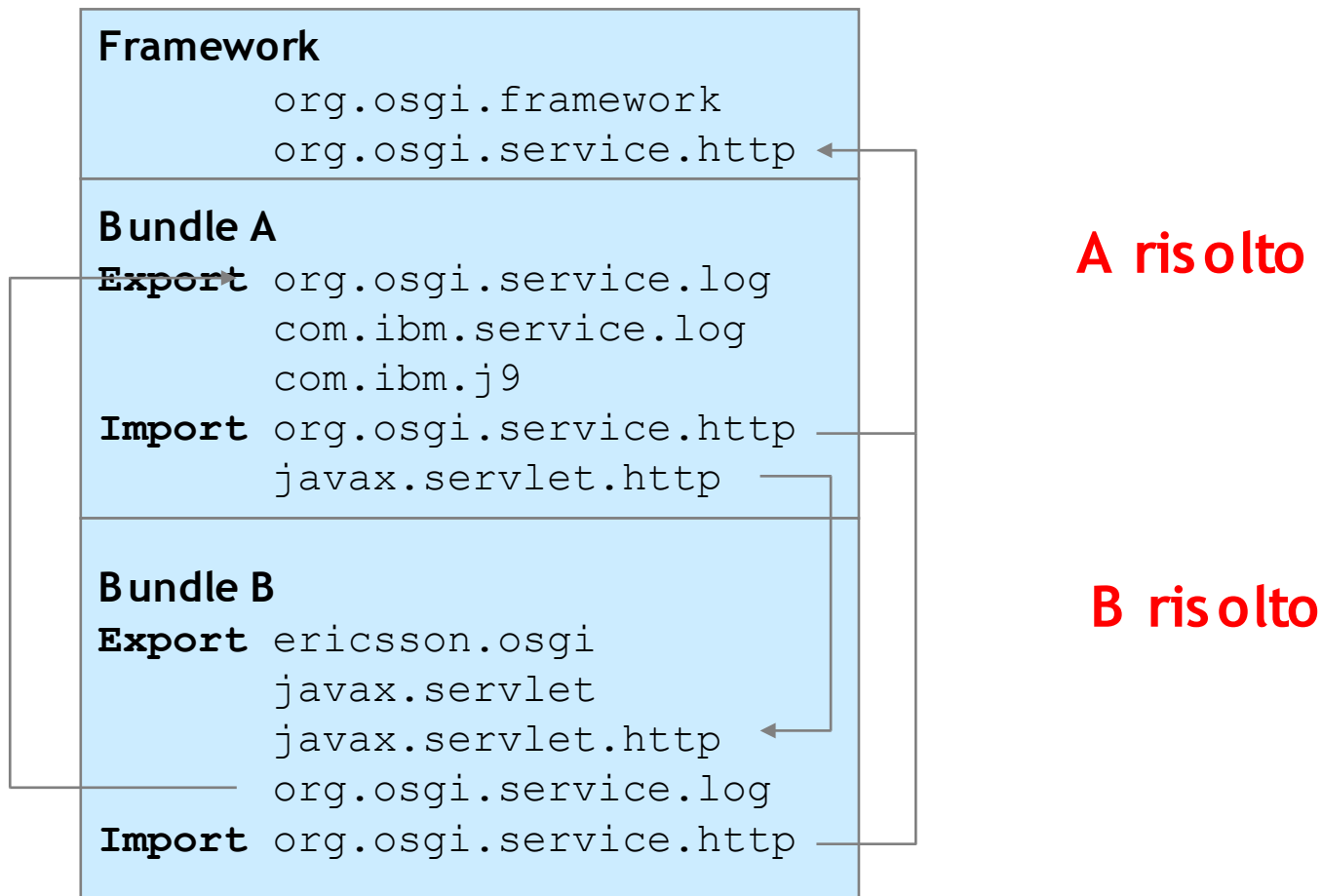
# I bundle OSGi: le dipendenze

- La gestione delle dipendenze avviene a due livelli:
  - Dipendenze dei bundle/packages (Deployment dependency)
    - Require-Bundle
    - Import-Package
  - Dipendenze dei servizi (Service dependency)

# I bundle OSGi: deployment dependency

- *Bundle-to-package*: un bundle può richiedere codice non contenuto nell'archivio jar, in questo caso importa il codice sotto forma di package
  - il codice richiesto deve essere dichiarato esplicitamente nel file *manifest* del richiedente
  - il codice richiesto deve essere esplicitamente esportato da un altro bundle
  - i bundle richiesti devono essere presenti nel framework per permettere al richiedente di essere attivato (active)
  - Questo tipo di dipendenza è gestito dal framework

# I bundle OSGi: dipendenze dai packages



# I bundle OSGi: dipendenze dai packages o dai bundles?

- Require-Bundle crea dipendenza con un intero bundle,
  - è semplice da usare ma puo' importare packages che non vengono usati
- Import-Package crea dipendenze con i soli packages specificati
  - preferibile perchè rende piu' semplice il versioning dei bundle

# I bundle OSGi: service dependency

- Gestione delle attività di pubblicazione, scoperta, binding dei servizi in un ambiente dinamico (servizi che vengono attivati o fermati dinamicamente)
- *Bundle-to-service*: un oggetto contenuto in un bundle può usare servizi esterni
  - Dichiarate nel file manifest per informazione, **non** gestite dal framework
- *Service-to-service*: l'implementazione di un servizio può richiedere l'uso di un servizio registrato
  - Non definite esplicitamente

# I bundle OSGi: le applicazioni

- Le applicazioni sono il risultato della *composizione* di più servizi
- Ciascun servizio deve scoprire se i servizi da lui richiesti sono disponibili
- I bundle devono pubblicare i servizi che espongono
- Ciascun servizio deve gestire il binding ed il rilascio automatico dei servizi richiesti
  - Dynamic assembly
  - Dynamic adaption

# I bundle OSGi: dynamic assembly & dynamic adaptation

- Dynamic assembly
  - Una applicazione è assemblata dinamicamente man mano che i servizi vengono pubblicati
  - Se i servizi non sono immediatamente disponibili il bundle va messo in attesa (idle)
- Dynamic adaptation
  - Capacità di adattarsi alle modifiche del service registry
    - *Monitoring*: ricezione delle notifiche del service registry
    - *Reconfiguration*: gestione delle modifiche

# I bundle OSGi: le dipendenze

- Le dipendenze sono caratterizzate da due fattori:
  - Cardinalità: 1, 1..n, etc.
  - Binding policy
    - Statica: la dipendenza non può cambiare a runtime
    - Dinamica
- Le dipendenze *service-to-service* e *bundle-to-service* si stabiliscono tra *istanze* e servizi.
- Una *istanza* è un oggetto creato da una classe contenuta in un bundle

# I servizi OSGi

- Il framework OSGi offre un servizio *registry* che permette la collaborazione tra i bundle
- La semantica di un servizio è definita dalla sua *interfaccia* (service interface),
  - Bundle forniti da provider differenti possono implementare la stessa interfaccia
- Un service object può implementare più interfacce
- Una implementazione di un servizio è riferita come un *oggetto servizio* (service object).

# I servizi OSGi

- Un servizio ha un identificatore unico
- Se la security è abilitata può essere necessario assegnare ai servizi i permission opportuni
- Un service object è legato ad un bundle e deve essere registrato da questi nel registry tramite il *BundleContext*
- A ciascun servizio possono essere associate delle proprietà
  - Le proprietà possono essere modificate a runtime
- Per la ricerca dei servizi viene usata la sintassi *LDAP*

# OSGi: registrazione/ricerca servizi

- I servizi sono registrati passando al framework attraverso il BundleContext i seguenti dati:
  - il nome dell'interfaccia
  - l'implementazione del servizio
  - le proprietà
- La ricerca dei servizi viene fatta fornendo al framework
  - il nome dell'interfaccia
  - un filtro LDAP con delle condizioni di ricerca

# OSGi: servizi

- Il framework consente di riferire servizi attraverso la classe `ServiceReference`
- Per ogni servizio registrato nel framework viene creato un oggetto `ServiceRegistration`
  - unico viene utilizzato per de registrare il servizio
- È possibile registrare un service factory invece che un oggetto
  - Permette di creare un servizio unico per ogni cliente

# OSGi : registrazione di un servizio

```
public class NotifyActivator implements
BundleActivator {
    private BundleContext context = null;

    private ServiceRegistration reg = null;
    public void start(BundleContext context) {
        this.context = context;

        //Notifica del nuovo servizio

        Properties props=new Properties();
        props.put("version", "1.0");
        reg = context.registerService(
            "org.ungoverned.MyService",
            new MyServiceImpl(), props);
        ....
    }
    public void stop(BundleContext context) {
        //gestione della disattivazione del bundle
    }
}
```

# OSGi : ricerca di un servizio

```
public class MessengerActivator implements BundleActivator
{
    private BundleContext context = null;
    public void start(BundleContext context) {
        this.context = context;

        // Cerca il servizio

        ServiceReference refs[]=context.getServiceReferences (
            "org.ungoverned.MyService", "(version=1.0)");

        if(refs!=null)
        {
            NotificationService ns =
                (MyService) context.getService (refs[0]);
        }

        //Uso del servizio
    }
    public void stop(BundleContext context) {
        //gestione della disattivazione del bundle
    }
}
```

# OSGi: demo

# OSGi demo: set up

- Software necessario:
  - Eclipse SDK (3.2 or 3.3) ([www.eclipse.org](http://www.eclipse.org))
  - aQute.tutorial.runtime.zip (<http://www.aqute.biz/OSGi/Tutorial>)
  - il plugin “bnd” installato (nella jar directory del tutorial runtime)
    - Facilita la creazione dei bundle generando automaticamente il manifestdi file manifest

# OSGi demo: installazione del plug in “bnd”

- Copiare il file bnd.jar dalla directory jar del aQute.tutorial.runtime alla directory dei plug in di eclipse
- Restart di eclipse :(
- Verificare: Help > About > Plugin Details
- Controllare che sia presente la descrizione “aQute Bundle Tool”

# OSGi tutorial: Hello World bundle

```
package aQute.tutorial.world;

import org.osgi.framework.*;

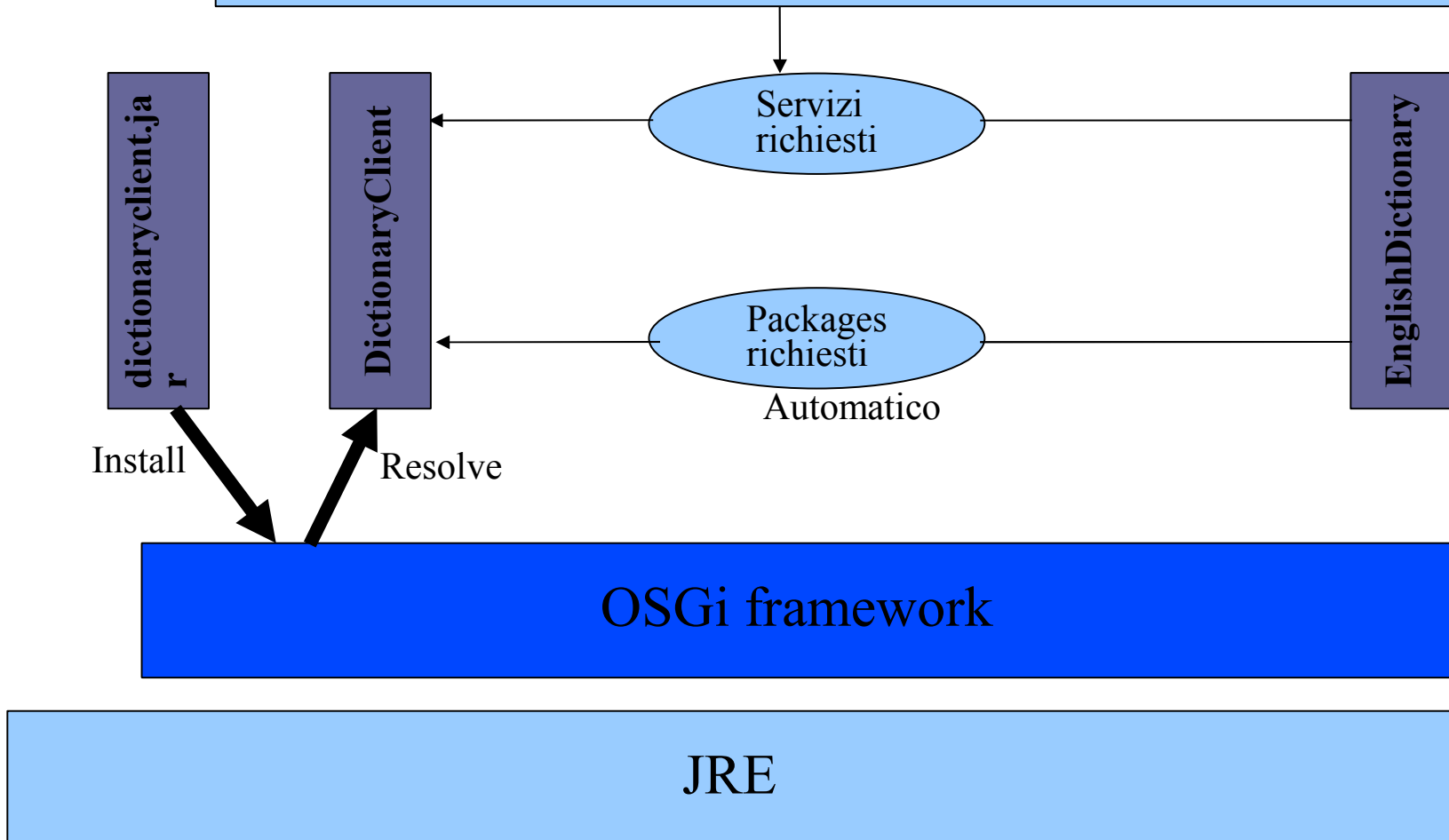
public class Activator implements BundleActivator {

    public void start(BundleContext context) throws Exception {
        System.out.println("Hello World");
    }

    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye World");
    }
}
```

# OSGi demo: DictionaryClient bundle

```
ServiceReference[] refs = ctxt.getServiceReferences(  
    DictionaryService.class.getName(), "(Language=*)");
```



# 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

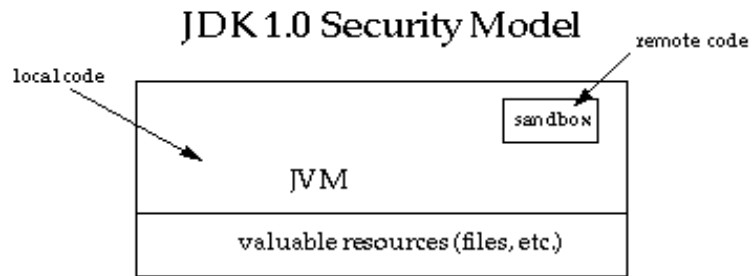
**Bundle-Description:** A bundle that uses the Italian dictionary service if it finds it at startup

**Bundle-Vendor:** Cesare Concordia

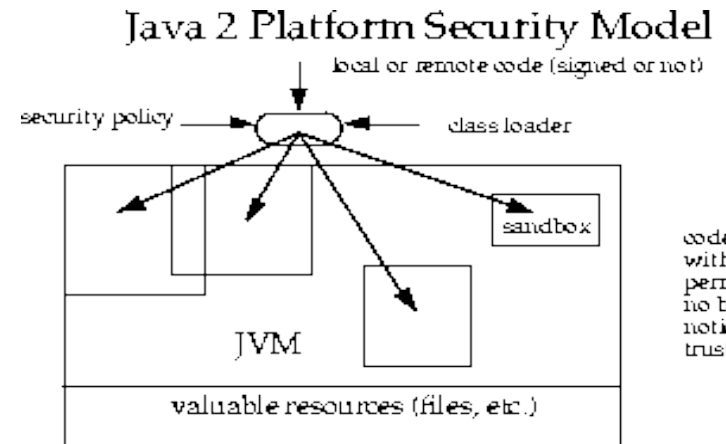
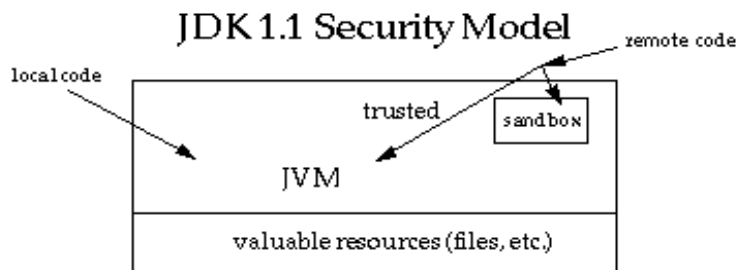
**Bundle-Version:** 1.0.0

**Bundle-Language:** IT

# La sicurezza in Java



Signed Applets



codes run with different permissions, no built-in notion of trusted code

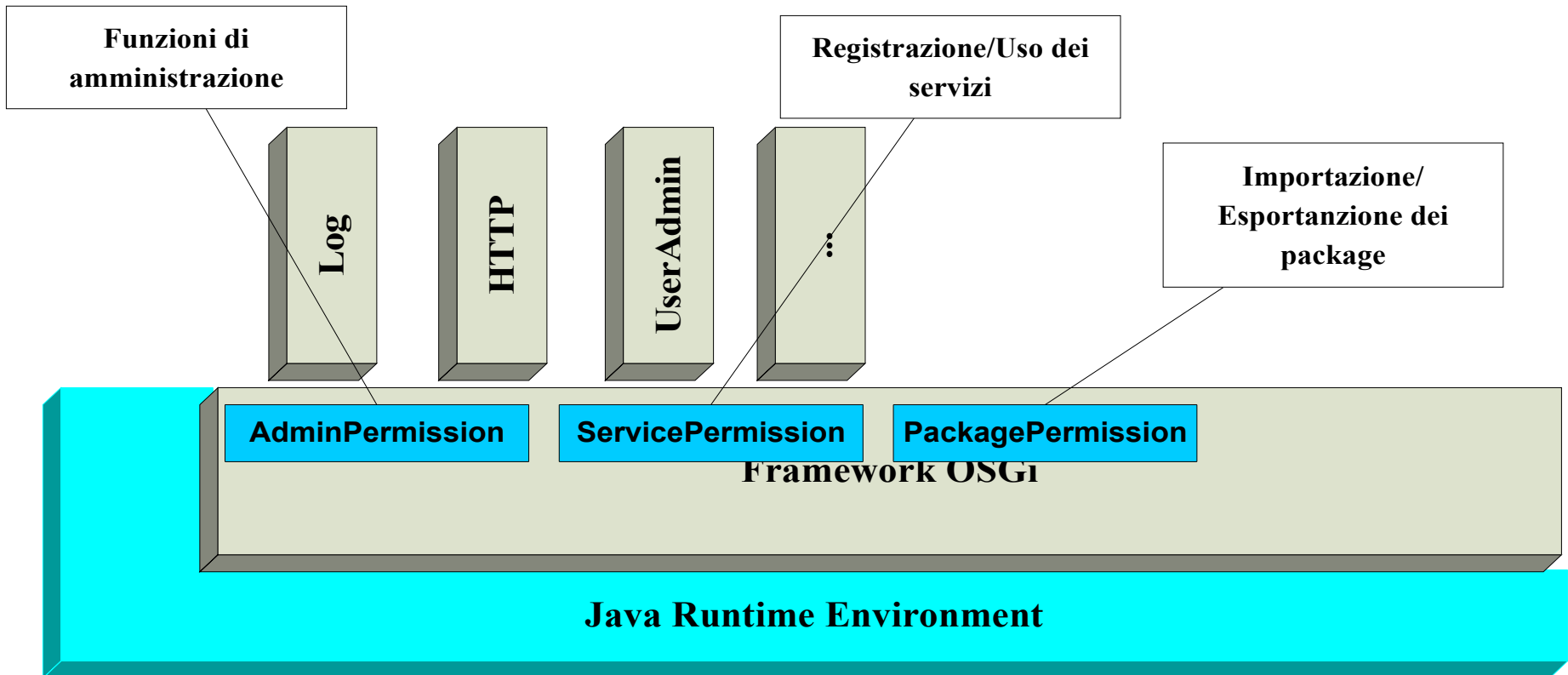
# La sicurezza in Java

- 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

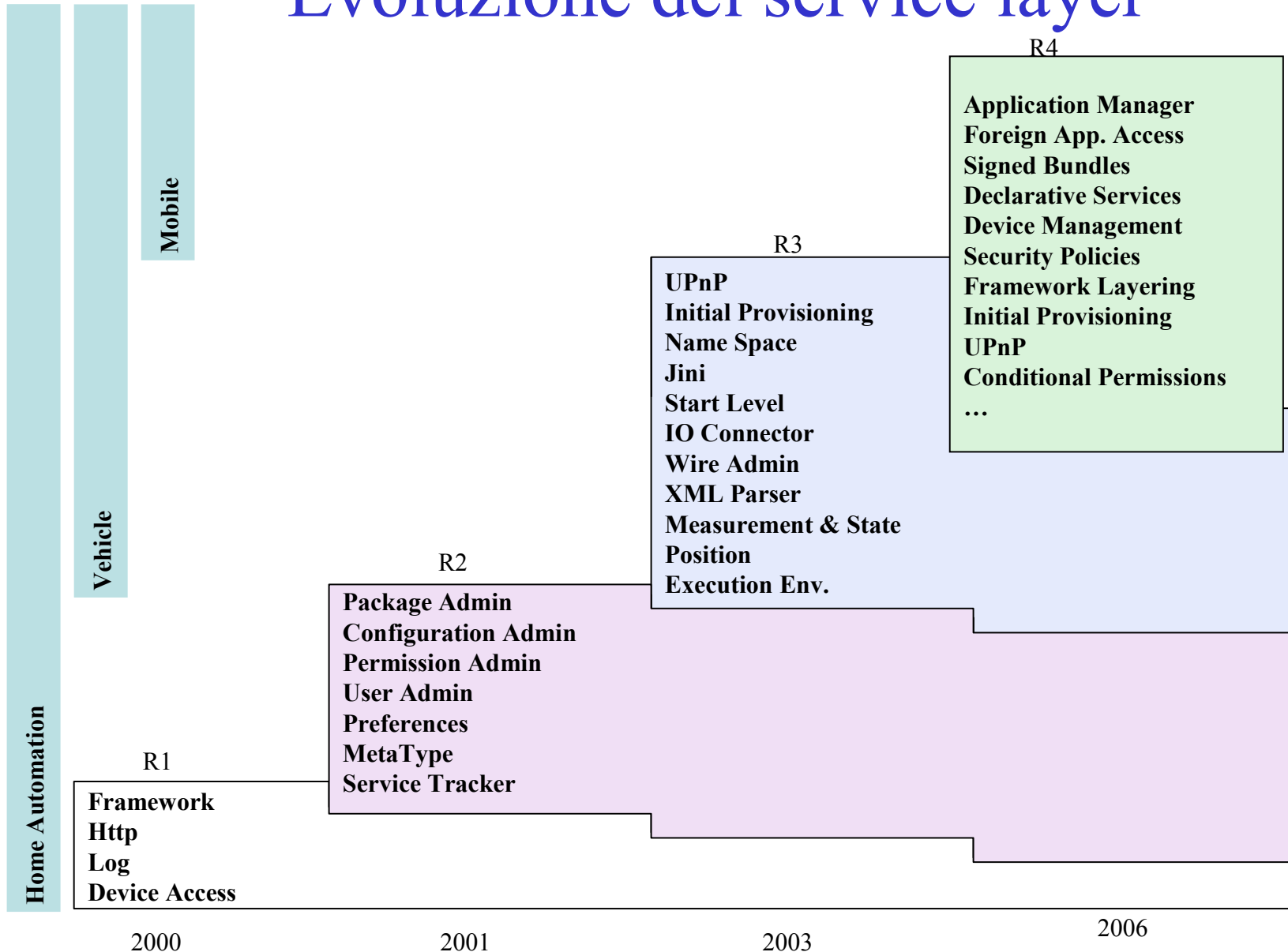
- **PackagePermission**
  - **Package Name** il nome del package su cui è assegnato il permission
  - **Action**
    - **IMPORT**
    - **EXPORT**
- Se un bundle ha il permesso di esportare un package ha automaticamente anche il permesso di importarlo
- Un PackagePermission con parametri (**\***, **EXPORT**) 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: Servizi nativi

# 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

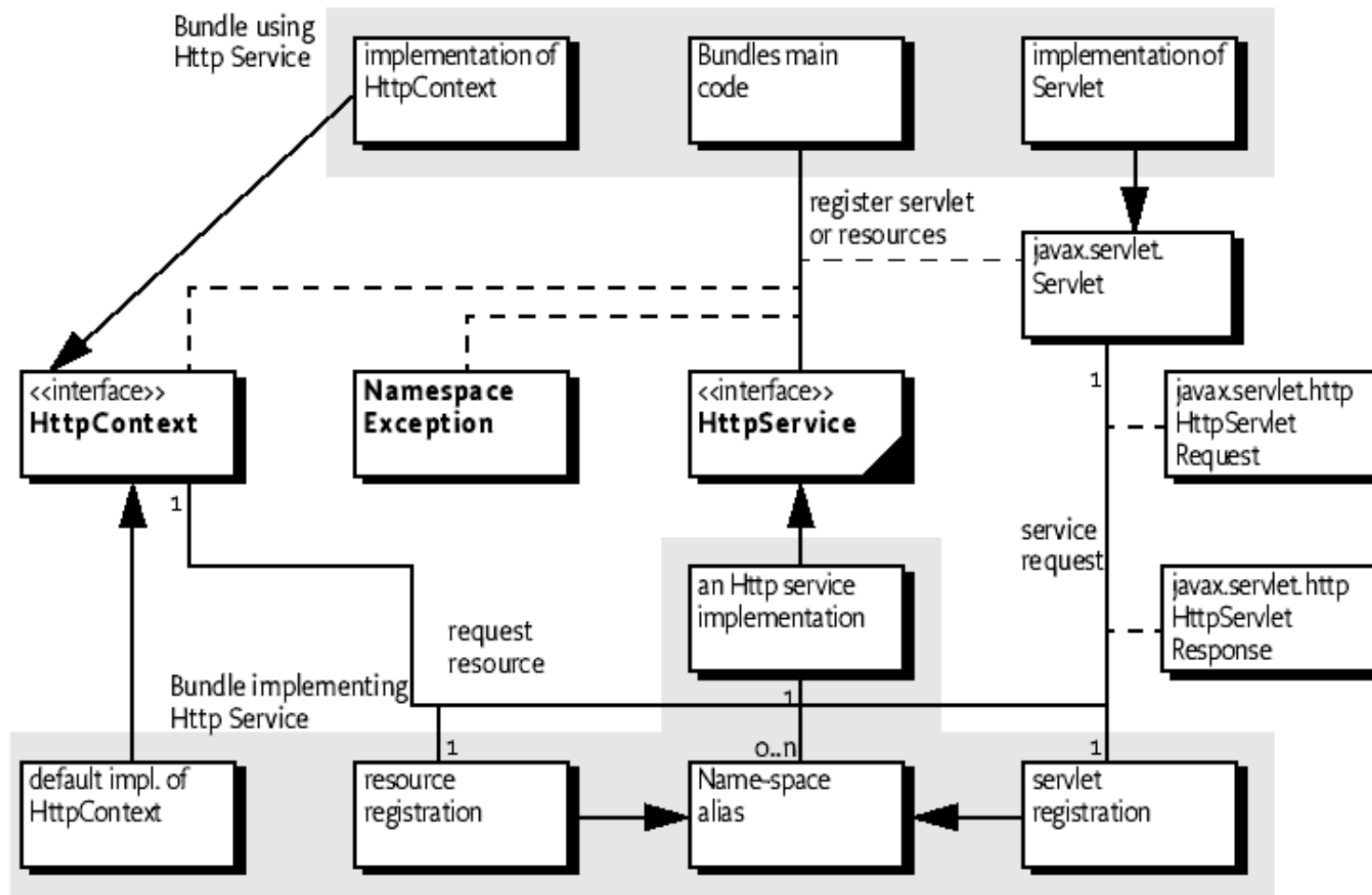
# OSGi: HTTP service

- *“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 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