Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sviluppo di un servizio di sintesi vocale per Smart Home in ambiente OSGi

A. Coronato – G. Sannino

RT-ICAR-NA-08-02 **marzo 2008**

Consiglio Nazionale delle Ricerche, Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR) – Sede di Napoli, Via P. Castellino 111, 80131 Napoli, URL: *www.na.icar.cnr.it*

Consiglio Nazionale delle Ricerche
Istituto di Calcolo e Reti ad Alte Prestazioni

Sviluppo di un servizio di sintesi vocale per Smart Home in ambiente OSGi

A. Coronato[1] – G. Sannino[1]

Rapporto Tecnico N.:
RT-ICAR-NA-08-02

Data:
marzo 2008

---

[1] Istituto di Calcolo e Reti ad Alte Prestazioni, ICAR-CNR, Sede di Napoli, Via P. Castellino 111, 80131 Napoli

Sviluppo di un servizio di sintesi vocale per Smart Home in ambiente OSGi

Antonio Coronato[1], Giovanna Sannino[1]

[1]ICAR-CNR, Via P. Castellino 111, 80131 Napoli, Italia
{ coronato.a, giovanna.sannino}@na.icar.cnr.it

Abstract

In questo lavoro descriveremo come è stato realizzato un servizio di sintesi vocale e come esso possa integrarsi ed interagire con altri servizi realizzati in un ambiente OSGi.

Negli ultimi decenni è cambiato, anche talvolta sostanzialmente, il nostro modo di vivere e di pensare all'ambiente che ci circonda. Un cambiamento dovuto alla necessità comune di migliorare l'accessibilità dell'ambiente, l'abitabilità, il confort, la sicurezza e la qualità della nostra vita.

Il grande passo che si cerca di compiere è quello di rendere quanto più trasparente possibile l'utilizzo dei dispositivi che interagiscono all'interno della smart home, far sì che il paradigma di comunicazione con l'utente sia più naturale e veloce e riuscire ad interconnettere tutti questi dispositivi insieme a formare un'unica rete.

1. Introduzione

La diffusione di computer sempre più piccoli e poco costosi a cui si è assistito negli ultimi anni, accoppiata con i progressi delle tecnologie di comunicazione, ha portato ad una nuova forma di Computing: il Pervasive Computing, che non è solo la naturale estensione dei paradigmi computazionali esistenti, ma rappresenta una e vera e propria visione della nostra vita futura.

Non a caso, più di dieci anni fa, Mark Weiser iniziò il suo articolo "The Computer for the 21st Century", in cui compariva, per la prima volta, il termine Pervasive

Computing, con la celebre frase: "*Le tecnologie più profonde sono quelle che svaniscono. Esse si fondono nella vita di tutti i giorni fino a divenire indistinguibile da essa*."

Il lavoro presentato in questo articolo si inserisce in un ambiente di Ambient Intelligence (AmI), cioè in un ambiente sensibile e reattivo alla presenza di persone.

AmI è una visione sul futuro dell'elettronica di consumo, telecomunicazioni e informatica, sviluppata alla fine degli anni '90 e basata sul Pervasive Computing e sullo Human- Centric Computer.

Occorre evidenziare che è sempre più forte l'impegno di rendere i servizi inseriti in questi ambienti quanto più trasparenti all'utente, dotandoli di interfacce di tipo naturale, ossia capaci di offrire risposte sensate a utenti che differiscono per esperienza, cultura, provenienza e intelligenza.

Il servizio OSGi descritto mira infatti alla realizzazione di un componente che permetta un'interazione naturale ed immediata verso l'utente attraverso l'uso della voce.

L'utilizzo del linguaggio naturale abbatte in parte gli standard di comunicazione tra utente e macchina ottenendo un significativo miglioramento della qualità di interazione.

Il sistema software di supporto per la realizzazione del servizio OSGi è un sintetizzatore vocale scritto interamente in Java: FreeTTS.

2. OSGi: Framework per le Smart Home

La Open Service Gateway iniziative nasce ad opera di un gruppo di compagnie tra le quali Nokia, Cisco, Sun Microsystems, IBM, Ericsson. L'idea di base del progetto era quella di standardizzare un residential gateway capace di connettersi con un vasto spettro di applicazioni per lo scambio di informazioni, in modo da fornire una interfaccia sia tra gli apparati domestici che tra questi e le reti pubbliche. Oggi la tecnologia OSGi ha riscosso molti consensi nelle community Open Source, come dimostrato dai progetti Apache Felix e Derby, Eclipse Callisto, Equinox e Corona, OSCAR, Knopflerfish, e altri. OSGi si distingue per le molteplici caratteristiche che presenta e definisce una API (Application Program Interface) interamente scritta in linguaggio Java, liberamente accessibile e portabile su diverse piattaforme.

2.1 Caratteristiche e dettagli

Le caratteristiche di OSGi sono:

• *Platform Independence*

• *Application Independence:* OSGi fornisce una piattaforma "orizzontale" su cui sviluppare applicazioni che implementano servizi

• *Multiple Service Support*: permette di registrare servizi di provider differenti

• *Service Collaboration Support:* interoperabilità fra servizi differenti

• *Security:* installazione e gestione remota

• *Multiple Network Technology Support: il framework*OSGi non è legato a nessun mezzo trasmissivo

• *Simplicity:* la complessità per la gestione dei servizi non deve ricadere sull'utente finale

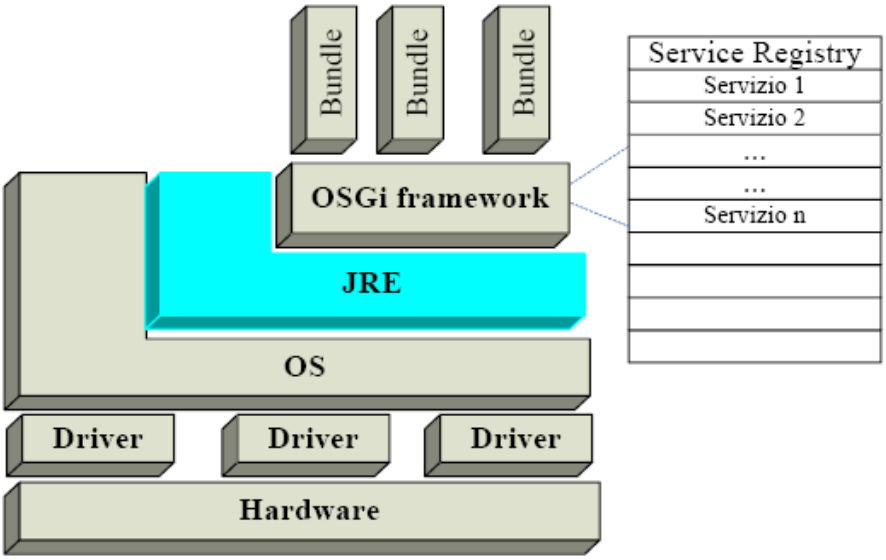

Figura 2.1 : OSGi framework

Componenti fondamentali, riportati nella figura, sono:

• *OSGiframework*: ambiente di esecuzione dei bundle

• *Servizi*:interfacce e relative proprietà

• *Bundle*:archivi contenenti l'implementazione dei servizi e le direttive di deployment

Un *bundle* è un file Java Archive (*jar*) nel quale sono contenute le risorse per implementare uno o più *servizi.* Queste risorse possono essere sia classi java sia altri tipi di file (HTML, gif, etc). C'è un file *manifest.mf* nel quale sono specificati i

parametri di installazione ed attivazione del bundle e le classi esportate e/o importate, le dipendenze dai package, gestite dal framework automaticamente e le dipendenze dei servizi, gestite da programma.

All'inteno del *bundle* è, inoltre, implementatala classe *Activator* che esegue l'inizializzazione dei parametri del bundle in fase di attivazione (start) e il clean up in fase di disattivazione (stop) del bundle. Infine può esserci la directory opzionale OSGI-OPT contenente ad esempio la documentazione o il codice sorgente.

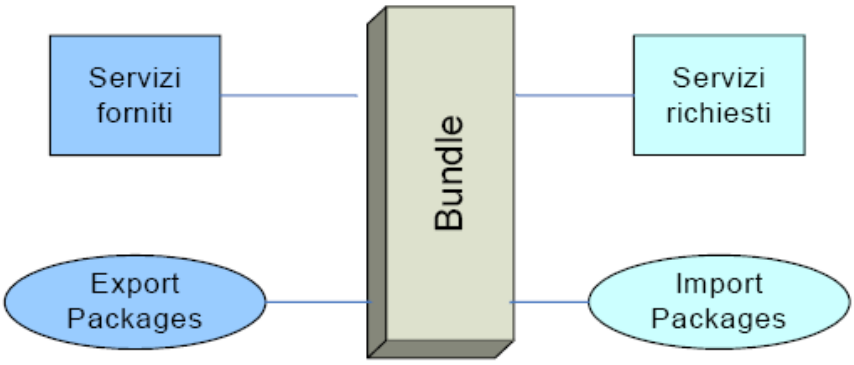

Figura 2.2 : I Bundle OSGi

Più precisamente la classe Activator implementa i metodi dell'interfaccia *BundleActivator* e permette al framework di gestire il *lifecycle* (deployment).

L'interfaccia BundleContext rappresenta l'ambiente di esecuzione del bundle all'internodel framework OSGi, viene creata dal framework quando un bundle viene attivato ed è usato dall'Activator di un bundle (BundleActivator) per interagire con il framework.

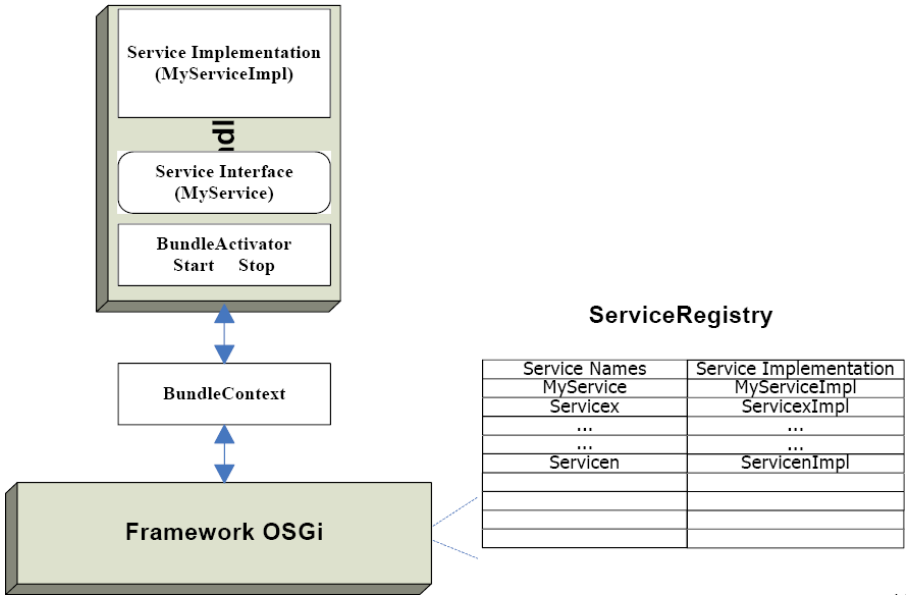

Figura 2.3 : Il BundleContext

In dettaglio il ciclo di vita di un bundle si articola in più fasi.

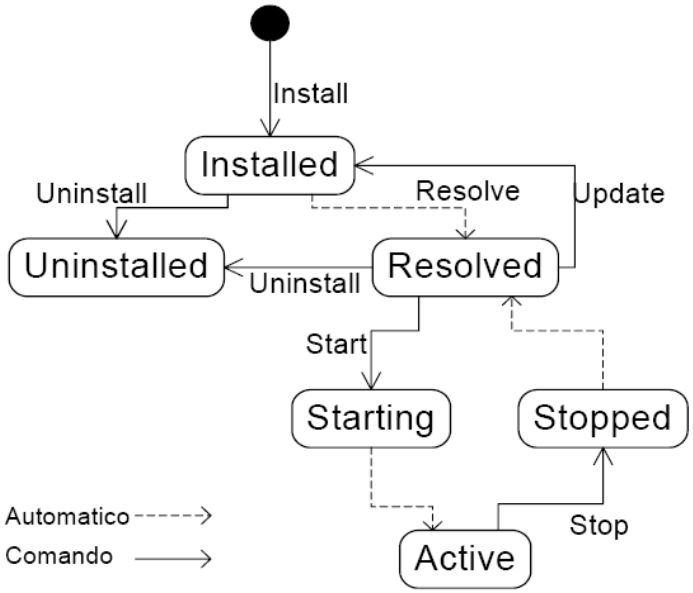

Figura 2.4 : Ciclo di vita di un Bundle

• Installazione:

Il framework OSGi legge il contenuto del bundle, assegna un ID, memorizza la locazione e lo stato. Un class loader specifico viene creato per accedere le risorse del bundle.

• Registrazione

Il framework controlla che le classi usate dal bundle siano disponibili (esportate da altri bundle), risolve anche le dipendenze dei bundle assicurandosi che il bundle A sia presente prima dell'attivazione del bundle B.

• Attivazione

Se la fase di registrazione è andata a buon fine il metodo start del bundle viene chiamato ed i service vengono registrati

• Disattivazione

Viene chiamato il metodo stop del bundle

• Disinstallazione

Il bundle viene rimosso

Una volta che il servizio è stato registrato attraverso il BundleContex, la loro ricerca viene fatta fornendo al framework: il nome dell'interfaccia e un filtro LDAP con delle condizioni di ricerca.

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.

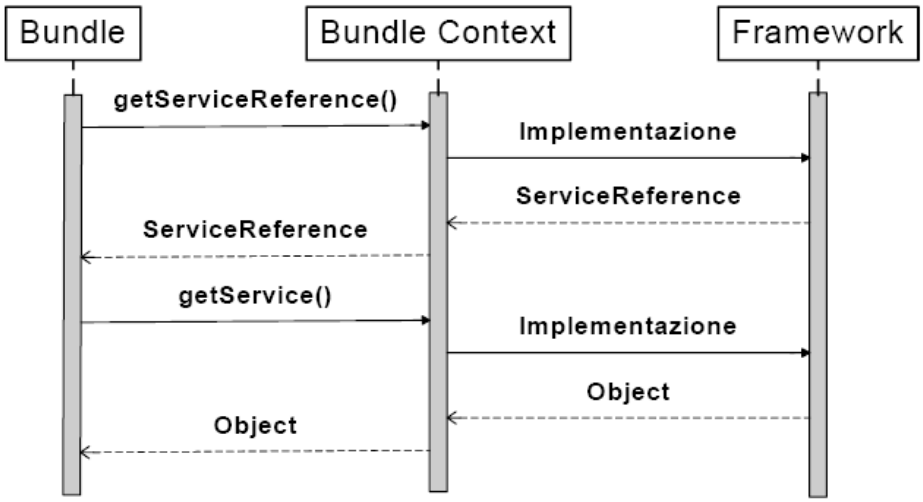

Figura 2.5 : Servizi OSGi

2.2 Knopflerfish OSGi

Knopflerfish è una distribuzione open source di OSGi. Scopo del progetto knopflerfish è sviluppare e distribuire un codice open source facile da usare, creare tools e applicazioni per tecnologie OSGi. La distribuzione del framework knopflerfish supporta la R4 completamente e solo due parti risultano in fase di perfezionamento: Security Layer e Framework Services. Nella prima è assente l'implementazione della verifica dei bundle (vedi paragrafo 4.1) mentre nella seconda il Conditional Permission Admin non è ancora stato incluso. Il K2 fornisce una serie di utili componenti:

- *Desktop*, è un tool grafico per gestire l'esecuzione del framework;
- *Log utilities*, classi utili per facilitare il login;
- *Misc utilities*, insieme di classi per la gestione dell'I/O e dei job;
- *Console*, fornisce tool basati su comandi necessari a rendere utilizzabili i propri bundle, ed include:
    - TTY Console, console con uso di stdin/stdout;
    - TCP Console, console che ascolta su una porta TCP;
    - Log commands, comandi per gestione del log di OSGi;
    - Framework commands, comandi per gestire il ciclo di vita di OSGi;

- CM commands, comandi per la gestione dei dati di con_gurazione;

- *OSGi Metatype XML*, l'implementazione del Metatype XML di OSGi definisce una piattaforma per informazioni metatype che possono essere usate dal CM commands e altri servizi che hanno bisogno di tali informazioni.

3. Architettura del servizio di sintesi vocale

La realizzazione di questo servizio ha portato dapprima alla realizzazione di un server RMI, basato su tecnologia JAVA, che esporta un servizio di sintesi vocale, e successivamente un bundle OSGi che lo rende disponibile ad altri bundle.
Il servizio RMI è definito da un'interfaccia che descrive i metodi che possono essere eseguiti in remoto.
Il server RMI deve registrarsi (presso un lookup service) per permettere ai client di poterlo trovare. Ad ogni registrazione è associato un nome (rappresentato come una stringa), che viene utilizzato dai client per selezionare il servizio appropriato.
Una volta che il servizio è registrato, esso attende le richieste provenienti dai client.
Questi ultimi inviano messaggi RMI per invocare metodi di oggetti remoti, ma prima di poter effettuare l'invocazione, devono ottenere un riferimento all'oggetto remoto che può essere tipicamente ottenuto cercando il servizio nel registro o richiedendo il nome di una particolare applicazione, e ricevere una URL alla risorsa remota.

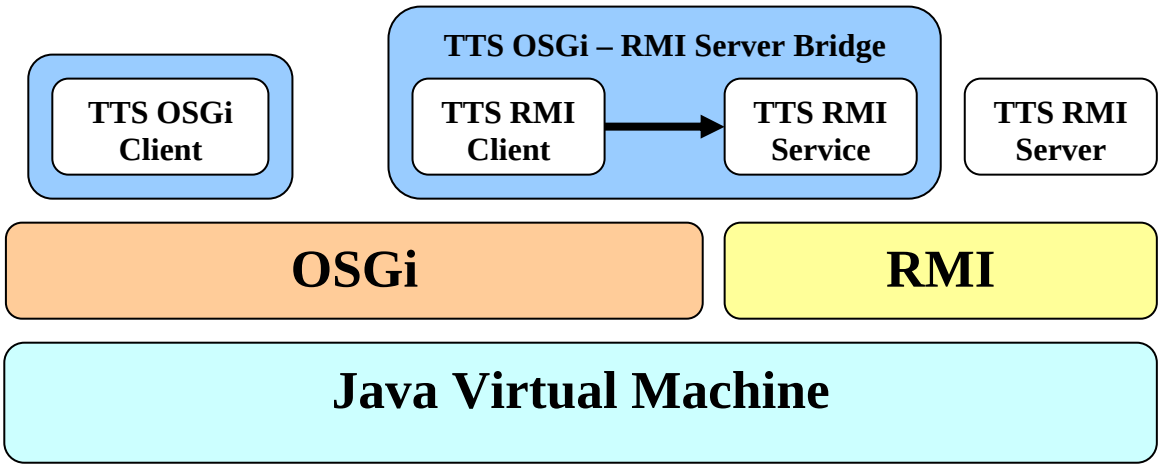

Figura 3.1 : Architettura del servizio

Alla base del servizio di sintesi vi è un sintetizzatore vocale, che può essere realizzato tramite software o via hardware. I sistemi di sintesi vocali sono spesso chiamati sistemi text-to-speech (TTS - in italiano: da testo a voce) per la loro possibilità di convertire il testo in parole.
Le qualità più importanti di una sintesi vocale sono la naturalezza, quanto la voce sintetizzata si avvicina a quella umana , e l'intelligibilità, facilità di comprensione della voce sintetizzata.
Le due tecnologie principali per la sintesi vocale sono la sintesi concatenativa e la sintesi basata sulle regole. Ciascuna tecnologia ha i suoi punti di forze e di debolezza: la scelta di quale utilizzare dipende tipicamente dal tipo di utilizzo finale della sintesi vocale.
Per la realizzazione del nostro servizio abbiamo utilizzato un motore di sintesi open-source scritto interamente nel linguaggio di programmazione Java: FreeTTS. Esso si basa su Flite: un piccolo run-time di motore di sintesi vocale sviluppato presso la Carnegie Mellon University. Flite deriva da Festival Speech Synthesis System presso l'Università di Edimburgo e dal progetto FestVox Carnegie Mellon University.

4. Dettagli implementativi

Prima di fornire una descrizione dei dettagli implementativi, presentiamo una sorta di detailed-deployment diagram che illustra le interazioni e le entità coinvolte quando si utilizza il servizio RMI.
Il **Server RMI** implementa un'interfaccia relativa ad un particolare oggetto RMI e registra tale oggetto nel Java RMI Registry. Il Java RMI Registry è, semplicemente, un processo di tipo daemon che tiene traccia di tutti gli oggetti remoti disponibili su un dato server. Il **Client RMI** effettua una serie di chiamate al registry RMI per ricercare gli oggetti remoti con cui interagire. Nel nostro caso il client RMI costituisce anche l’OSGi Server che esporta il servizio TextToSpeech utilizzato da un Client OSGi.

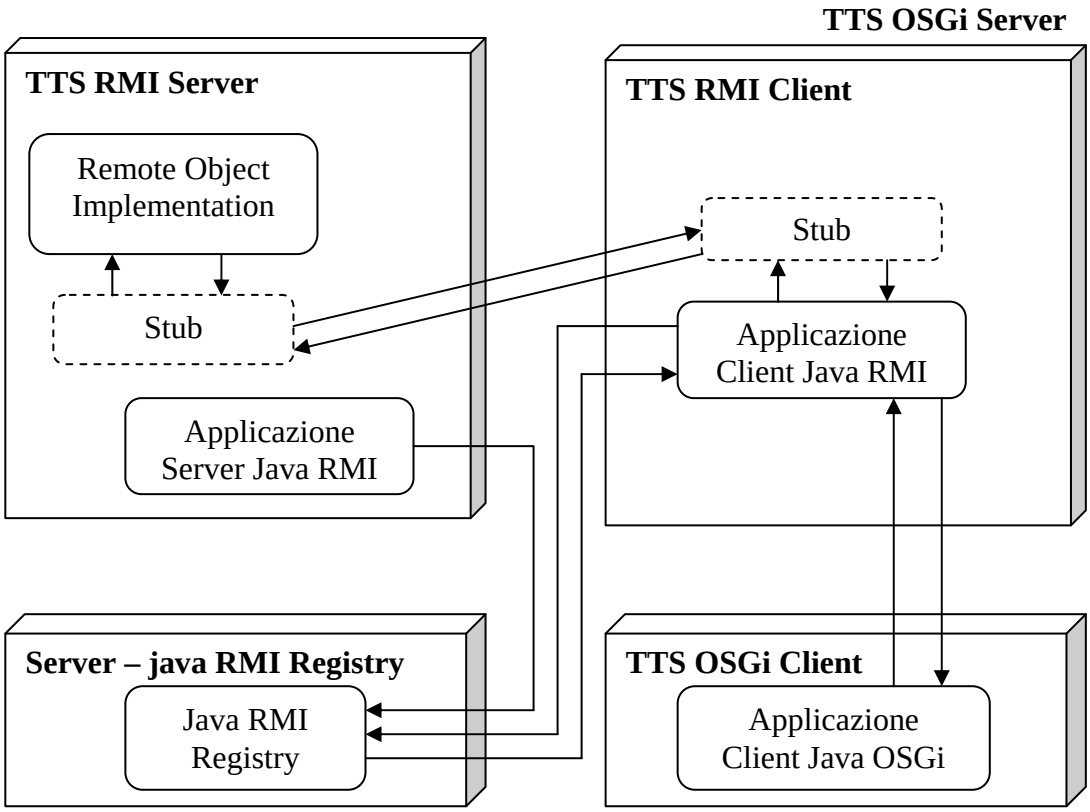

Figura 4.1 : Architettura ed interazione tra i vari componenti

Vediamo in pratica l'implementazione del nostro oggetto remoto che espone un metodo che consente di sintetizzare un messaggio vocale.

4.1 Creazione dell'interfaccia remota

Anzitutto creiamo un'interfaccia remota che contenga al suo interno la definizione del metodo che l'oggetto sul server espone ai client.

```
package TextToSpeech;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TTSinterface extends Remote
{
  void text_to_speech(String text) throws RemoteException;
}
```

4.2 Creazione dell'applicazione server

Creata l'interfaccia remota vediamo l'implementazione della classe sul server che contiene l'implementazione dell'oggetto remoto vero e proprio.

```
package TextToSpeech;

import java.util.Properties;
import java.net.MalformedURLException;
import java.rmi.server.UnicastRemoteObject;
import java.rmi.Naming;
import java.rmi.RemoteException;
import com.sun.speech.freetts.Voice;
import com.sun.speech.freetts.VoiceManager;


public class TTSserver extends UnicastRemoteObject implements TTSinterface
{
	public TTSserver()throws RemoteException
	{

	}

	public void listAllVoices() throws RemoteException
	{

		System.out.println();
		System.out.println("All voices available:");
		VoiceManager voiceManager = VoiceManager.getInstance();
		Voice[] voices = voiceManager.getVoices();
		System.out.println(voices.length);
		for (int i = 0; i < voices.length; i++)
		{
			System.out.println("    " + voices[i].getName()
					+ " (" + voices[i].getDomain()
					+ " domain)");
		}
	}

	public void text_to_speech(String text) throws RemoteException
	{	listAllVoices();
		String voiceName = "kevin";
		System.out.println();
		System.out.println("Using voice: " + voiceName);
		/* The VoiceManager manages all the voices for FreeTTS. */
		VoiceManager voiceManager = VoiceManager.getInstance();
		Voice helloVoice = voiceManager.getVoice(voiceName);
		System.out.println(helloVoice);
		if (helloVoice == null)
		{
			System.err.println(
				"Cannot find a voice named "
					+ voiceName
					+ ".  Please specify a different voice.");
		}
		/* Allocates the resources for the voice.  */
		helloVoice.allocate();
		/* Synthesize speech.   */
```

```
            helloVoice.speak(text);
            /* Clean up and leave.  */
            helloVoice.deallocate();
      }


      public static void main(String[] argv)
      {     try
            {
                  TTSinterface server = new TTSserver();
                  Naming.rebind("//localhost/Text_To_Speech_Server",
                                    server);
                  System.out.println(" >>  REGISTRAZIONE SERVIZIO...<< ");
            }
            catch (RemoteException e) { e.printStackTrace(); }
            catch (MalformedURLException e) { e.printStackTrace(); }
            System.out.println(" >> SERVIZIO REGISTRATO:
                              + "Text_To_Speech_Server ");
      }
}
```

Molto importante è l'istruzione *Naming.rebind("//localhost/Text_To_Speech_Server", server)* con la quale, infatti, viene effettuato il bind dell'oggetto server (di tipo TTSinterface) con il nome " *Text_To_Speech_Server* ".
Si noti, altresì, che è stato utilizzato l'indirizzo localhost che, nel caso in cui l'applicazione Server si fosse trovata su una workstation separata, avrebbe dovuto essere sostituito dall'indirizzo IP di tale macchina.

4.3 Creazione del ServerOSGi

Compito del componente è ricercare l'applicazione server RMI che espone gli oggetti remotizzati messi a disposizione ed invocarne il metodo.
In realtà tale bundle è sia un client RMI che un server per gli altri bundle OSGi.
Nell'Activator viene registrato il servizio che sarà poi utilizzato dagli altri client OSGi.

```
package TextToSpeechClient;

import java.util.Properties;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;


public class Activator implements BundleActivator {

public void start(BundleContext context) throws Exception {
     Properties props = new Properties();
     props.put("TextToSpeech", "String");
     context.registerService( TextToSpeech.class.getName(),
                       new TextToSpeechImpl(), props);   }
```

```
public void stop(BundleContext context) throws Exception { }
}
```

Nel bundle è poi presente l'interfaccia del servizio

```
package TextToSpeechClient;

public interface TextToSpeech {
      public void text_to_speech(String text);
}
```

E l'implementazione dello stesso

```
package TextToSpeechClient;

import java.util.Properties;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.net.MalformedURLException;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import TextToSpeech.TTSinterface;

public class TextToSpeechImpl extends Thread implements TextToSpeech {

      public void text_to_speech(String text) {
            try
            { TTSinterface ServizioRemoto =
                  (TTSinterface)Naming.
                        lookup("rmi://localhost/Text_To_Speech_Server");
                  ServizioRemoto.text_to_speech(text);
            }
            catch (NotBoundException e)
            {     e.printStackTrace();          }
            catch (RemoteException e)
            {     e.printStackTrace();          }
            catch (MalformedURLException e)
            {     e.printStackTrace();          }
      }

      public TextToSpeechImpl(){    }

}
```

È importante notare l'istruzione *TTSinterface ServizioRemoto = (TTSinterface) Naming.lookup ("rmi://localhost/Text_To_Speech_Server")* che si occupa proprio di effettuare il lookup, ossia la ricerca, dell'oggetto remoto denominato "*Text_To_Speech_Server* " sulla macchina il cui indirizzo è localhost.

4.4 Creazione del ClientOSGi

Infine presentiamo la creazione di un componente OSGi che utilizza il servizio TextToSpeech messo a disposizione dal Server OSGi realizzato in precedenza.

```
package TTSClientOSGi;


import java.util.Properties;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
import org.osgi.framework.ServiceReference;
import TextToSpeechClient.TextToSpeech;

public class Activator implements BundleActivator {

public void start(BundleContext context) throws Exception {

      ServiceReference TextToSpeechRef =
context.getServiceReference(TextToSpeech.class.getName());
      if (TextToSpeechRef != null)
      {     TextToSpeech TextToSpeechService =
                  (TextToSpeech)context.getService(TextToSpeechRef);
            TextToSpeechService.text_to_speech("Hello World");
      }
 }

public void stop(BundleContext context) throws Exception { }
}
```

5. Conclusioni

Il risultato finale consiste nell'aver sviluppato un componente RMI server, un bundle OSGi che importa il servizio remoto e lo rende disponibile nell'ambiente OSGi e un bundle client che ne fa uso. È interessante notare come vari componenti interagiscono all'interno di una Smart Home per produrre la sintesi vocale di un testo.

Si ricorda che per utilizzare i bundle, si necessita di una JavaVirtualMachine e di un'implementazione di framework OSGi, come Knopflerfish.

Una volta predisposto l'ambiente, il servizio realizzato può entrare a fare parte del sistema domotico ed interagire con altri bundles.

Bibliografia

1. M. Weiser, *"The Computer for the 21st Century"*, Scientific Am., Sept., 1991, pp. 94-104.
2. D. Salber, A. K. Dey, G. D. Abowd, *"Defining an HCI Research Agenda for an Emerging Interaction Paradigm"* Technical report, GVU Center and College of Computin, Georgia Tech, 1999.
3. M. Satyanarayanan, Carnegie Mellon, *"Pervasive Computing: Vision and Challenger"*.
4. *"Pervasive Computing" http://en.wikipedia.org/wiki/Ubiquitous_computing*
5. *"Pervasive Computing"* http://laspinanelfianco.wordpress.com/2006/03/08/pervasivecomputingubiquitus-computing/
6. *"Ambient Intelligence"* http://en.wikipedia.org/wiki/Ambient_intelligence
7. F.Sorrentino, F.Paganelli, *"L'intelligenza distribuita. Ambient Intelligence: il futuro delle tecnologie invisibili*", Erickson, Trento 2006.
8. *"Smart Home"* http://it.wikipedia.org/wiki/Domotica
9. *"OSGi"* http://www.osgi.org/About/Technology
10. *"Human Computer Interaction"* http://it.wikipedia.org/wiki/Interazione_umano-computer
11. *"Sintesi Vocale"* http://it.wikipedia.org/wiki/Sintesi_vocale
12. *"FreeTTS"* http://freetts.sourceforge.net/docs/index.php