Opening OSGi to the world
Simple integration of services not written in Java

Tobias Wegner

11.06.2008 4:00pm SESSION Track 2
Overview

• The Problem

• Existing solutions

• Our proposal

• Conclusion
Introduction

- OSGi
  - Amazing technology
  - Fast growing

- But: Whole platform based on Java

  → Problem: Existing software, other prosperous platforms
Beside OSGi

• „Old“ software
  • Enterprise / industry specific
  • Historically grown over years
  • High migration costs

• Proprietary software
  • Only available as black box
    • Dedicated API
  • Migration might not even be possible
Example

- Contacts from Outlook
  - User manages his/her address book with Outlook
  - User has a home equipped with intelligent electronics
    ➞ E.g. inHaus in Duisburg

  ➞ When someone calls him on the phone, the caller’s name should be displayed on the TV

  ➞ The controlling OSGi service platform has to get information from the user’s address book
Requirements

- Easy to implement
- Transparent for existing bundles
  - Code reusability
- Support for various languages
Possible solutions

- Java Native Interface (JNI)
- Webservices (WS)
- Universal Plug and Play (UPnP)
- R-OSGi
JNI

- Allows to call native code
  - Using dynamic libraries (dll, so)
    - Wrapper library
  - Proxy class
  - Security issues
  - No garbage collection
    - Caution with pointers (can change in java, locking)
  - Explicit conversion of certain types
Webservices (WS)

- Based on standards like XML, HTTP
  - Platform & vendor independent

- Service description

- No standard driver for import
UPnP

- Discovery
- Description
- Control
  - Based on HTTP and SOAP
- Eventing

- Import as generic UPnPDevice
WS / UPnP Disadvantages

• XML
  • Not suitable for bigger amounts of data

• No transparency
  • Manual Service Proxy creation
R-OSGi

- Developed at the ETH Zurich
- Facilitates distributed OSGi
- Transparent usage of remote services
  - No changes to existing bundles needed
R-OSGi 2

- Efficient RMI like data transmission
  - Automatic service proxy generation
  - Remote services are entered into local service registry

- Discovery through Service Location Protocol (SLP)

- Only available for OSGi (Java)
R-OSGi architecture

- Consumer
- Proxy
- R-OSGi
- Export

- OSGi
- Provider
- R-OSGi
- Import
Idea

- Porting R-OSGi to other platforms

- Technical requirements
  - Similar data types
  - Object oriented
  - Reflection

⇒ .NET
# OSGi vs. .NET

<table>
<thead>
<tr>
<th></th>
<th>OSGi</th>
<th>.NET</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported Languages</td>
<td>Java</td>
<td>C++, C#, Visual Basic, Java, Fortran, Pascal, Perl, Python, many more</td>
</tr>
<tr>
<td>Support for various platforms</td>
<td>yes</td>
<td>yes (by Mono)</td>
</tr>
<tr>
<td>Support for software components</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Dynamic software unloading/update</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Example architecture

- MS Outlook
  - Proprietary
  - Office API
  - .NET programmable 😊
R-OSGi's needs

- Discovery
  - SLP
  - several implementations available (OpenSLP)
  - can be ported to .NET
  - can be renounced
R-OSGi‘s needs 2

• Proxy generation
  • Class description
    • Members
    • Methods
  • Can be obtained from .NET object using reflection

• Class file for Service interface
  • Cannot be obtained from .NET 😞
  • Has to be provided by the developer
R-OSGi‘s needs 3

- Method invocation
  - Java streaming I/O (JS I/O)
    - Serialization of parameters and return value

- Method and member matching
  - Name is the only mapping mechanism
  - Type for additional validity check
JS I/O

- No compatible counterpart in .NET
  - no serialization / deserialization of Java objects under .NET

- Available as open source
  - porting to .NET
    - currently no support for Arrays
    - currently suid has to be provided manually
JS I/O 2

- Similar naming scheme
  - Hierarchical package names
  - Identify corresponding classes by package & name

- Type handling
  - Simple types explicit
  - Classes recursive
R-OSGi.NET System architecture

- HDD
- Java interface
- Service
- R-OSGi.NET
- JS I/O.NET
- .NET

Socket connection
Serialized objects

OSGi

R-OSGi

Service Consumer

Service Proxy
Sample code

- **Java interface**

```java
package org.irf.interfaces;

public interface IContactInfo {
    public String getContactName(int);
}
```

- **C# class**

```csharp
public class CIImpl : IContactInfo {
    public String getContactName(int i) {
        return <NameFromAddressBook>;
    }
}
```
Sample code 2

- C# code for export

```csharp
IRemoteOSGiService ROSGiService = new R_OSGi.NET.Impl.RemoteOSGiServiceImpl();
CIlmpl contactInfoService = new CIlmpl();
ROSGiService.RegisterService( "org.irf.interfaces.IContactInfo",
    "C:\Temp\org.irf.interfaces.jar",
    contactInfoService);
```

...
Conclusion

• Simple import of services running under .NET into OSGi
  • Support for various languages

• Transparency
  • Existing bundles can be used

• Minimal effort to the developer

• Currently only simple types and classes with simple types supported
Perspective

- Update to latest R-OSGi version
- Support for
  - Arrays, hashtables, etc.
  - Eventing
- For more information
  - http://www.it.irf.uni-dortmund.de/IT/Mitarbeiter/Wegner.php
  - Mail: tobias <dot> wegner <at> tu-dortmund <dot> de