Evolving Communication Mechanisms
of the OSGi Framework

Xuejun Chen
BMW Group
Forschung und Technik
Munich, Germany
Overview

• Motivation
  – Dynamic distribution of applications in vehicle
  – OSGi is being used in vehicles.

• Current solution to location transparence
  – Java RMI is used.
  – Performance problem

• New solution to location transparence
  – Automatically switching invocations from remote to
    local and vice versa

• Conclusion
Motivation (1)

- Vehicle software systems
  - Controllers and sensors
  - Telematics systems
  - Management systems
- Distributed systems in vehicles
  - Today, approx. 70 ECUs in a BMW vehicle
  - In the future, maybe only several high performance ECUs in vehicles
Motivation (2)

• Requirements of future vehicle software
  – Optimized system performance
  – Dynamic distribution of applications
  – Location transparency

• OSGi is being used in vehicle software
  – However, OSGi does not provide support for location transparent communication.
Current solution to location transparence

- Java RMI is used for location transparent communication.

- System performance is degraded because RMI is used for local communication
  - Java RMI’s serialization takes at least 25% of the costs of a remote method invocation.
  - The costs of serialization rise with growing object structures.
New solution to location transparence

• Virtual stub based solution to location transparency
• A virtual stub is a local object for a client component, and forwards the invocations from the client to the server.
• A virtual stub is dynamically loaded and updated by a system service.
• The virtual stub can automatically switch invocations from remote to local and vice versa.
• The virtual stub is used like a normal Java RMI stub.
Components communicate with each other by using virtual stubs.

```java
TestInterface v_Stub;
v_Stub = (TestInterface) componentContext.lookup("TestServer_B");
try {
    info = v_Stub.testMethod("a test call");
}
```
Integrating virtual stub into OSGi

- Implementing a bundle manager that registers the references of the local bundles, and loads, updates a required virtual stub for a client bundle.
Automatically switching invocations from remote to local and vice versa

if (local) /* A and B are local.*/
str = (String)ref.testMethod(clone(info));
else /* A and B are remote.*/
str = (String)ref.testMethod(info);
Measurement results of local/remote invocations

Test case: component A calls a method of component B, which returns a string.

<table>
<thead>
<tr>
<th>Runtime environment</th>
<th>Middleware</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two components run on a desktop PC. i.e. A and B are local.</td>
<td>Java RMI</td>
<td>1.0114</td>
</tr>
<tr>
<td></td>
<td>Virtual Stub</td>
<td>0.0623</td>
</tr>
<tr>
<td>Component A runs on a desktop, component B runs on a laptop, i.e. A and B are remote.</td>
<td>Java RMI</td>
<td>2.7913</td>
</tr>
<tr>
<td></td>
<td>Virtual Stub</td>
<td>2.8356</td>
</tr>
</tbody>
</table>

- When both components are local, the system performance of using virtual stub is significantly higher than using Java RMI.
- Overhead of virtual stub is only 0.0443 ms.

The measurement results are cited from [Chen2002].
Conclusion

• By using virtual stub, communication between components is location transparent, and at the same time, the system performance is not degraded.

• For more detailed information, please see the following article:

Thank you for your attention!

eMail: xuejun.chen@bmw.de