Java RMI and Enterprise JavaBeans

Vito Baggioolini SL/CO
What is RMI

- **Java Remote Method Invocation**
  - between objects in different Java Virtual Machines

- Mono-language
- Multi-platform
- 100% Java
RMI Features

- **OO remote method call**
  - Pass Java objects as parameters

- **Name lookup & binding (location service)**
  - Find and bind to remote objects

- **Easy multi-threaded servers**

- **Remote object activation**
  - Server objects occupy memory only when active

- **Automatic distributed garbage collection**

- **Dynamic class loading over the net**
Java Features used by RMI

- **Communication-level security**
  - Encryption and authentication (SSL)

- **Introspection**
  - Explore interfaces at run-time
    => support for generic clients and servers

- **Locking**
  - Enforces exclusive access to server methods

- **Integration of legacy (JNI)**
  - Relatively easy for server-side C/C++ code
  - Bulky for client-side C/C++ code

- **Late binding (linking)**
Importance of Features

- Fully OO Remote Method Call
  - Local and distributed programming very similar
  - Allows late decisions on how to distribute application

- Passing objects as parameters
  - Not only data, but also code
  - No manual packaging/unpackaging

- Dynamic class loading + late binding
  - No need to re-compile on changes
  - Smooth and transparent upgrades
public class Magnet extends UnicastRemoteObject
    implements MagnetI
{
    public Magnet() {
        Naming.bind("//eanorth/magnet", this);
    }
    int current;
    public void setCurrent(int val) {
        current = val;
    }
}

public static void main(…) {
    MagnetI mag = (MagnetI)
        Naming.lookup("//eanorth/magnet");
    mag.setCurrent(300);
}
Java vs. CORBA

- **CORBA**: integration tool
  - CORBA’s strength: language independent

- **RMI**: distributed programming tool
  - RMI’s strength: language centric

- **RMI + CORBA → RMI over IIOP**
  - ✔ Easy programming interface (RMI)
  - ✔ Interoperability
  - ✘ Restricted functionality
Java Centric vs. Language Independent

<table>
<thead>
<tr>
<th>Java centric</th>
<th>Language independent</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔ Full Java functionality everywhere</td>
<td>❌ Lowest common denominator</td>
</tr>
<tr>
<td>❌ More difficult to integrate other languages</td>
<td>✔ Easy to integrate languages</td>
</tr>
</tbody>
</table>
Related Java Technology

- **JavaBeans**
  - Software components, enable visual programming

- **JavaDoc**
  - Easy documentation on the Web

- **Embedded Java**
  - Small memory footprint, 100% Java

- **Compiled Java (unofficial)**
  - Easy programming, good performance
Enterprise JavaBeans
What are Enterprise JavaBeans?

- Framework for Server Components...
- ...with access to Middleware services

- 1. Goal: easy server development
  - Developers concentrate on their application
- 2. Goal: portable server components
  - Components can be used in any EJB Product
EJB Component Model

- **Container** (= Run-time Environment)
- **Enterprise JavaBean** (= your application)
Middleware Access

- Container provides access to middleware services...
- ...transparently
- EJB can use middleware explicitly
- ... through standardized Java API’s
Standardized Enterprise API’s

- Naming and Directory Service
- Transactions
- Message-oriented Middleware
- Remote Method Call
- Database Access
Summary

◆ RMI
✓ Easy to use
✓ Truly OO
✓ Dynamic class loading and late binding
✓ Distributed garbage collection
✗ Integration of other languages less easy
✗ Performance, scalability (solved with Java 2.0?)
✗ Still in evolution

◆ EJB
✓ Easy to use
✓ 100% Java, portable
✗ Still in evolution
✗ Not much vendor support yet