Java RMI

Quality Aspects

Vladimir Smatanik
vladimir.smatanik@gmail.com

Richard Dérer
richard.derer@gmail.com

Jacek Marek
pl.jacekmarek@gmail.com

Alexandru Dimitriu
alexdimitriu2005@gmail.com

Utrecht University

December 17, 2010

Outline

1. Quality aspects
2. Trade-offs
3. Use Cases
   - Connection
   - Method Execution
4. Alternate systems
5. Materials & Further Reading

Quality aspects

ISO/IEC 9126 Software engineering - Product quality

ISO/IEC 9126

- Address some of the well known human biases that can adversely affect the delivery and perception of a software development project
- Tries to develop a common understanding of the project's objectives and goals
- Divided into four parts:
  - quality model
  - external metrics
  - internal metrics
  - quality in use metrics.

Quality model

Quality model classifies software quality in a structured set of characteristics:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Functionality

Functionality Quality Aspect and RMI

A set of attributes that bear on the existence of a set of functions and their specified properties. The functions are those that satisfy stated or implied needs.

- Suitability
- Accuracy
- Interoperability
- Security
- Functionality Compliance

Reliability

Realiability Quality Aspect and RMI

A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time.

- Maturity
- Fault Tolerance
- Recoverability
- Reliability Compliance

Usability

Usability Quality Aspect and RMI

A set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

- Understandability
- Learnability
- Operability
- Attractiveness
- Usability Compliance

Efficiency

Efficiency Quality Aspect and RMI

A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

- Time Behaviour
- Resource Utilisation
- Efficiency Compliance

Maintainability

Maintainability Quality Aspect and RMI

A set of attributes that bear on the effort needed to make specified modifications.

- Analyzability
- Changeability
- Stability
- Testability
- Maintainability Compliance

Portability

Portability Quality Aspect and RMI

A set of attributes that bear on the ability of software to be transferred from one environment to another.

- Adaptability
- Installability
- Co-Existence
- Replaceability
- Portability Compliance

Trade-offs

Using Java as a programming language

- memory management
  - GC and automatic memory allocation
  - overal control over memory
- portability
  - platform independent
  - slightly slower performance

Functionality

Security

- security manager
- policy configuration files

Reliability

Fault tolerance

- remote exceptions
- more source code

Usability

Attractiveness

- working in Java

Usability Compliance

- object oriented
- potentially expensive

Efficiency

Resource Utilization

- long-lived persistent objects
- adding complexity to the system

Performance

- communication via Internet
- performance loss for our comfort
- bypassing firewalls

Maintainability

Maintainability

- easy to write
- easy to maintain
- not so much functionality

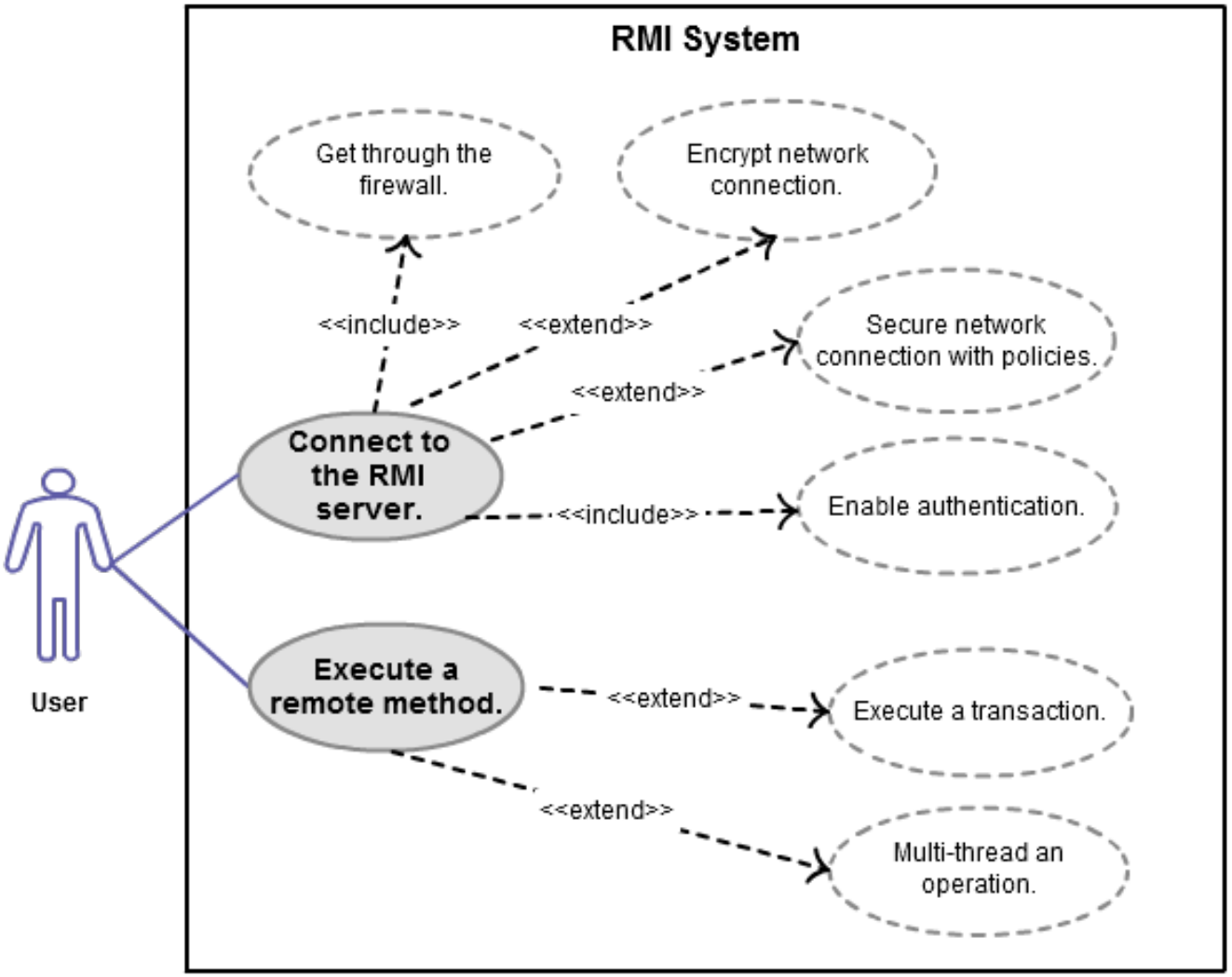
Portability

Portability

- Write Once, Run Anywhere model
- only in Java

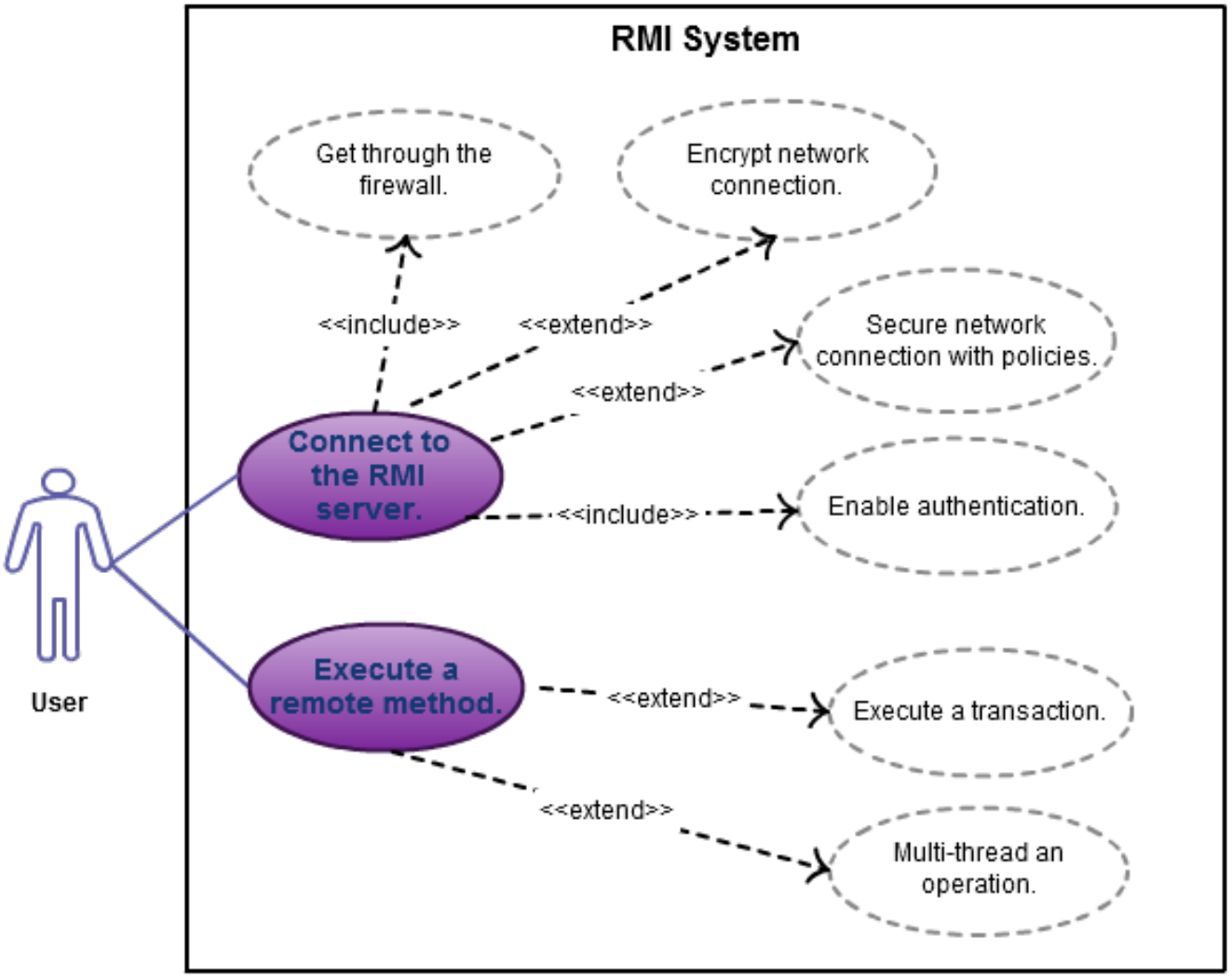
Use Cases

Use Cases



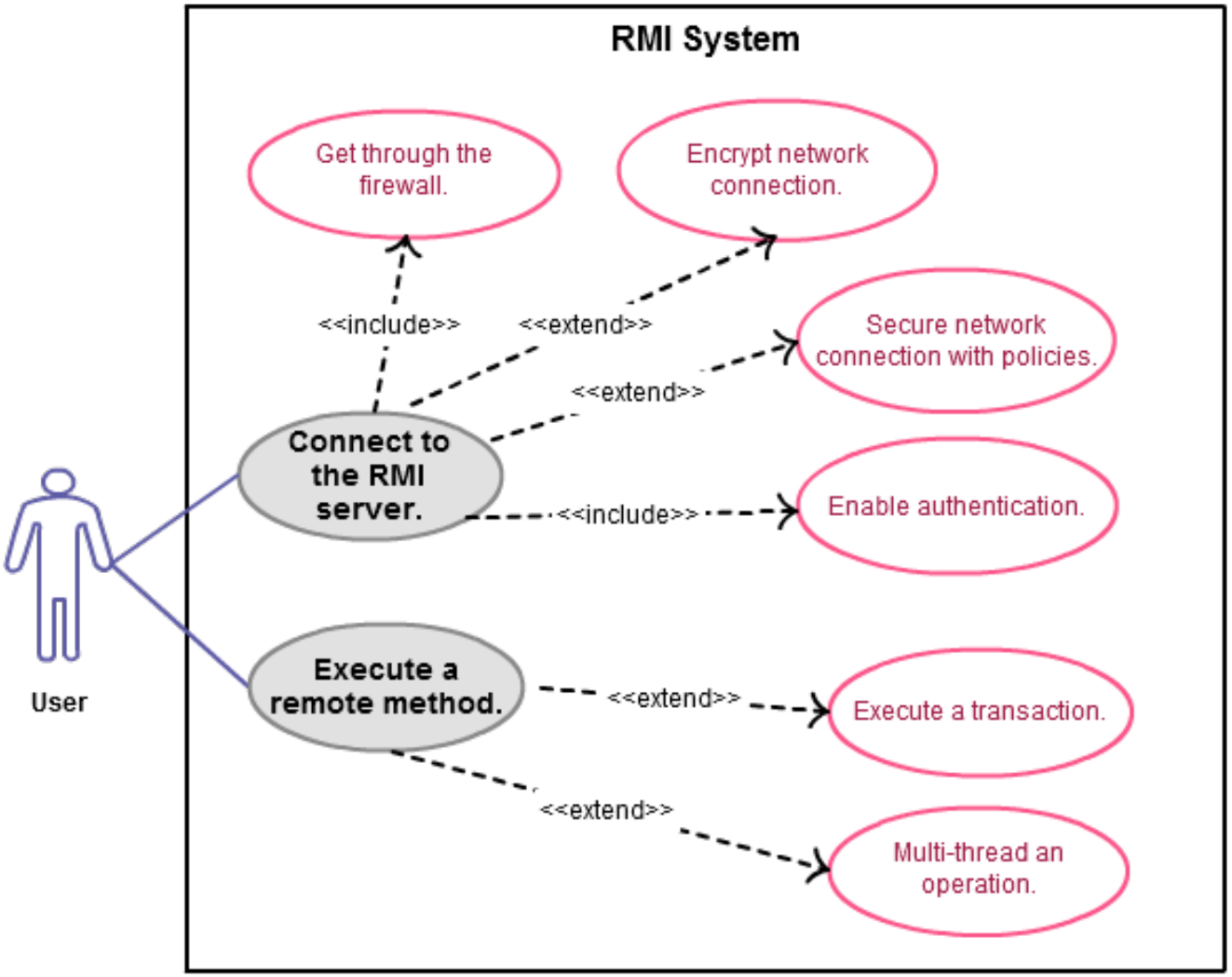
[online diagramming & design] creately.com

Use Cases



[online diagramming & design] creately.com

Use Cases



[online diagramming & design] creately.com

Secure network connection with policies

- Security Manager

Secure network connection with policies

- Security Manager

```
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new RMISecurityManager());
}
```

Secure network connection with policies

- Security Manager

```
if (System.getSecurityManager() == null) {
   System.setSecurityManager(new RMISecurityManager());
}

public class RMISecurityManager extends java.lang.SecurityManager {
    public RMISecurityManager() {}
}
```

Secure network connection with policies

- Security Manager

```
if (System.getSecurityManager() == null) {
   System.setSecurityManager(new RMISecurityManager());
}

public class RMISecurityManager extends java.lang.SecurityManager {
    public RMISecurityManager() {}
}
```

Policy configuration

java.exe -Djava.security.policy=server.policy RmiServer

Secure network connection with policies

- server.policy

```
grant {
  permission java.net.SocketPermission "127.0.0.1:*", "connect,resolve";
  permission java.net.SocketPermission "127.0.0.1:*", "accept";
};
```

Secure network connection with policies

- server.policy

```
grant {
  permission java.net.SocketPermission "127.0.0.1:*", "connect,resolve";
  permission java.net.SocketPermission "127.0.0.1:*", "accept";
};
```

- Security Manager assertions

```
SecurityManager sm = System.getSecurityManager();

sm.checkConnect("remotehost", 1099);
sm.checkPropertyAccess("java.rmi.server.codebase");
sm.checkRead("/tmp/file");
sm.checkListen(1099);
```

Encrypt network connection

- exporting RMI service without encryption

```
UnicastRemoteObject.exportObject(appImpl, port);
```

Encrypt network connection

- exporting RMI service without encryption

```
UnicastRemoteObject.exportObject(appImpl, port);
```

- encrypting RMI client-server channel

```
RMIClientSocketFactory csf = new AppClientSocketFactory(pattern);
RMIServerSocketFactory ssf = new AppServerSocketFactory(pattern);
UnicastRemoteObject.exportObject(appImpl, port, csf, ssf);
```

Encrypt network connection

- exporting RMI service without encryption

```
UnicastRemoteObject.exportObject(appImpl, port);
```

- encrypting RMI client-server channel

```
RMIClientSocketFactory csf = new AppClientSocketFactory(pattern);
RMIServerSocketFactory ssf = new AppServerSocketFactory(pattern);
UnicastRemoteObject.exportObject(appImpl, port, csf, ssf);
```

- encrypting RMI client-registry/server-registry channel

```
LocateRegistry.createRegistry(port, csf, ssf)
LocateRegistry.getRegistry(host, port, csf)
```

Enable authentication

- Using additional RMI login service

```
LoginService loginService =
   (LoginService) Naming.lookup("rmi://remotehost/LoginService");

SecuredService securedService =
   (SecuredService) loginService.login(user, pass);
```

Enable authentication

- Using additional RMI login service

```
LoginService loginService =
   (LoginService) Naming.lookup("rmi://remotehost/LoginService");

SecuredService securedService =
   (SecuredService) loginService.login(user, pass);
```

- JAAS (Java Authentication and Authorization Service)

Enable authentication

- Using additional RMI login service

```
LoginService loginService =
   (LoginService) Naming.lookup("rmi://remotehost/LoginService");

SecuredService securedService =
   (SecuredService) loginService.login(user, pass);
```

- JAAS (Java Authentication and Authorization Service)
- since JRE 1.4

Enable authentication

- Using additional RMI login service

```
LoginService loginService =
   (LoginService) Naming.lookup("rmi://remotehost/LoginService");

SecuredService securedService =
   (SecuredService) loginService.login(user, pass);
```

- JAAS (Java Authentication and Authorization Service)
- since JRE 1.4

JAAS authorization package

*javax.security.auth.**

Get through the firewall.

- Done by RMI internally.

Get through the firewall.

- Done by RMI internally.

Communication techniques priority:

1. Communicate directly to the server's port using sockets.

Get through the firewall.

- Done by RMI internally.

Communication techniques priority:

1. Communicate directly to the server's port using sockets.
2. Build a URL to the server's host and port and use an HTTP POST request to send the information to the skeleton as the body of the POST.

Get through the firewall.

- Done by RMI internally.

Communication techniques priority:

1. Communicate directly to the server's port using sockets.
2. Build a URL to the server's host and port and use an HTTP POST request to send the information to the skeleton as the body of the POST.
3. Build a URL to the server's host using port 80 using a CGI script that will forward the posted RMI request to the server.

Get through the firewall.

- Done by RMI internally.

Communication techniques priority:

1. Communicate directly to the server's port using sockets.
2. Build a URL to the server's host and port and use an HTTP POST request to send the information to the skeleton as the body of the POST.
3. Build a URL to the server's host using port 80 using a CGI script that will forward the posted RMI request to the server.

Communication techniques

Techniques 1 and 2 are significantly slower but work across the Internet & Web.

Execute a transaction.

Passing control

When a remote method is invoked, the control is passed to the server.

Execute a transaction.

Passing control

When a remote method is invoked, the control is passed to the server.

- Server can use a transaction mechanism of choice

Execute a transaction.

Passing control

When a remote method is invoked, the control is passed to the server.

- Server can use a transaction mechanism of choice
- JTA (Java Transaction API)
- Java EE API (*javax.transaction.**)

Execute a transaction.

Passing control

When a remote method is invoked, the control is passed to the server.

- Server can use a transaction mechanism of choice
- JTA (Java Transaction API)
- Java EE API (*javax.transaction.**)

```
@Stateless
@TransactionManagement(BEAN)
public class ExampleBean {

   @Resource private UserTransaction utx;

    public void foo() {
        utx.begin(); // start a transaction
        // Do work
        utx.commit(); // Commit it
    }
}
```

Multi-thread an operation.

Server threading possibilities:

- Thread-per-request
- Thread-per-connection
- Thread-per-object

Multi-thread an operation.

Server threading possibilities:

- Thread-per-request
- Thread-per-connection
- Thread-per-object

RMI specification - automatic server-side threading

"A method dispatched by the RMI runtime to a remote object implementation (a server) **may or may not execute in a separate thread**. Calls originating from different clients Virtual Machines **will execute in different threads**. From the same client machine **it is not guaranteed that each method will run in a separate thread**."

Multi-thread an operation.

Server threading possibilities:

- Thread-per-request
- Thread-per-connection
- Thread-per-object

RMI specification - automatic server-side threading

"A method dispatched by the RMI runtime to a remote object implementation (a server) **may or may not execute in a separate thread**. Calls originating from different clients Virtual Machines **will execute in different threads**. From the same client machine **it is not guaranteed that each method will run in a separate thread**."

- It is not thread safe - concurrency mechanisms required

Multi-thread an operation.

Server threading possibilities:

- Thread-per-request
- Thread-per-connection
- Thread-per-object

RMI specification - automatic server-side threading

"A method dispatched by the RMI runtime to a remote object implementation (a server) **may or may not execute in a separate thread**. Calls originating from different clients Virtual Machines **will execute in different threads**. From the same client machine **it is not guaranteed that each method will run in a separate thread**."

- It is not thread safe - concurrency mechanisms required
- RMI call blocks client thread waiting for results

Multi-thread an operation.

Server threading possibilities:

- Thread-per-request
- Thread-per-connection
- Thread-per-object

RMI specification - automatic server-side threading

"A method dispatched by the RMI runtime to a remote object implementation (a server) **may or may not execute in a separate thread**. Calls originating from different clients Virtual Machines **will execute in different threads**. From the same client machine **it is not guaranteed that each method will run in a separate thread**."

- It is not thread safe - concurrency mechanisms required
- RMI call blocks client thread waiting for results
- Separate thread for RMI calls

Alternate systems

Distributed computing infrastructure

- Why do we need it?
  - Software gets larger, slower, more expensive
  - Hardware gets smaller, faster, cheaper
- Solution
  - Split up the software on multiple computers
  - Use RPC Models to communicate

Which one is better?

- Impossible to compare them straight forward (apple-orange idiom)
- Depends on many factors
  - Usability
  - Portability
  - Security
  - Performance

Usability

- Web Services, Java RMI and CORBA all have a quite similar architecture, the costs of development are very similar.
- WS & CORBA require the knowledge of an additional language to describe the interface:
  - WSDL - Web Service Definition Language (for WS)
  - IDL - Interface Description Language (for CORBA)
- Java RMI uses TCP/IP by default, ports need to be opened if data passes through a firewall. WS rely on HTTP, which is more convenient because ports 80/443 are not normally blocked by firewalls.

Portability

- WS - platform-independent and language-independent (XML data)
- CORBA - platform-independent, language-independent
- Java RMI - platform-independent, but supports only Java

Security

- Security is not a default feature for WS, Java RMI or CORBA
- It can be achieved:
  - Web Services - using WS-Security
  - Java RMI & CORBA - using SSL (Secure Socket Layer) or TLS (Transport Layer Security)

Performance - resource utilisation

- Java RMI - uses a distributed garbage collector
- CORBA objects are not garbaged collected (as it is language-independent and some languages don't support garbage collector, e.g. C++)

Performance - time behavior

- Java RMI has the best overall performance, slightly better than CORBA. However, its client-side performance is lower than CORBA's when handling large amount of data.
- WS performance in terms of time behavior is by far the worse.
- Some benchmarks:
  - WS - $\approx$ 9 times slower than RMI
  - RMI-SSL - $\approx$ 40 times slower than RMI
  - WS-Security - $\approx$ 100 times slower than WS
  - Windows - improved performance for RMI (5%) & RMI-SSL (15%)
  - Linux - improved performance for WS (8%) & WS-Security (30%)

Materials & Further Reading

Materials & Further Reading

- http://en.wikipedia.org/wiki/ISO/IEC_9126
- http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html
- http://www.comp.lancs.ac.uk/ weerasin/csc253/tutorials/week7.html