Introduction to Enterprise Computing

Giovanni Chierico
CERN (IT-AIS-HR)

Inverted CERN School of Computing

Presentation "prerequisites"

The presentation doesn't go into too much details, but it might be useful to have:

- General knowledge of distributed systems
- Some experience with OO Programming
- Some Java Experience

Presentation Overview

- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

What is "Enterprise Computing"

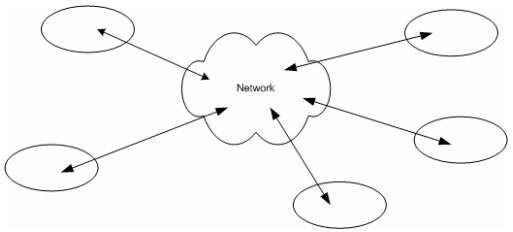
Solving computing problems in a

- Distributed
- Multi-tier
- Server-centric environment.

Common in big companies (like CERN) where users access a variety of applications that share data and resources, often integrated with legacy systems.

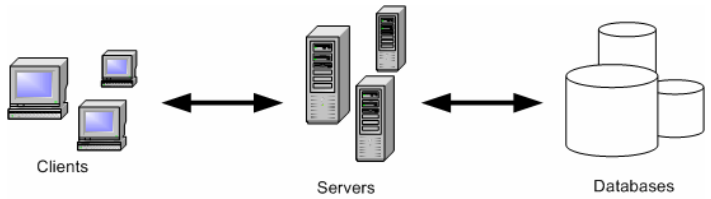
Distributed

- Means that the “components” that make up our system could be living on different machines and communicate through the network
- Components must be able to find each other and to communicate effectively



Multi-tier

- Many distributed schemas are possible (e.g. P2P)
- In an enterprise environment we can identify components having very different roles (client, server, database) and different requirements



Server centric

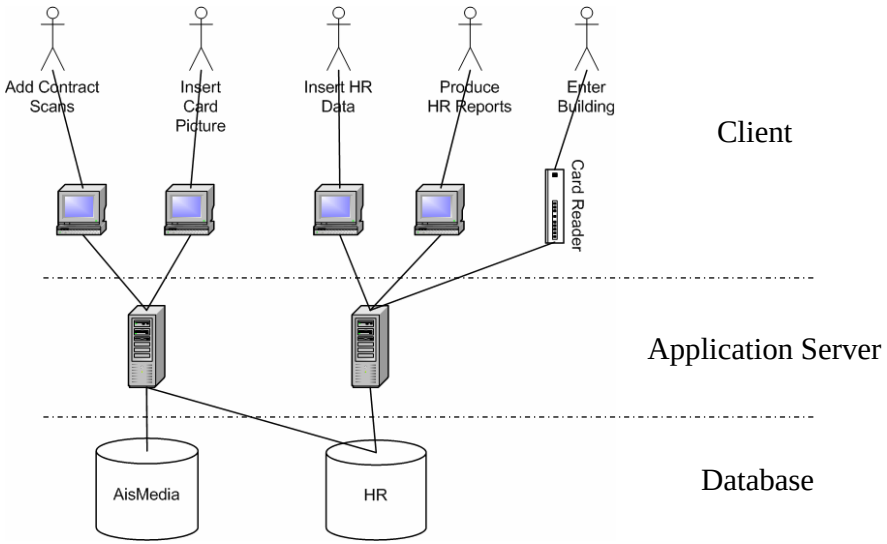
- Client “thin” and “standard” to simplify requirements and deployment
- Server implements the business logic
- Database offers standard data persistence and retrieval functionalities

… but sometimes the division is blurred

Common 3-tier architecture

1. Client
   - Interfaces with the user
2. Server
   - Implements Business logic
   - Implements Middleware logic
3. EIS (Enterprise Information System)
   - Persistently stores data
   - Retrieve stored data

Examples



Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Common Problems/Services (I)

- Remote method invocation
- Load balancing
- Transparent fail-over
- System integration
- Transactions management

Common Problems/Services (II)

- Logging
- Threading
- Messaging
- Pooling
- Security
- Caching

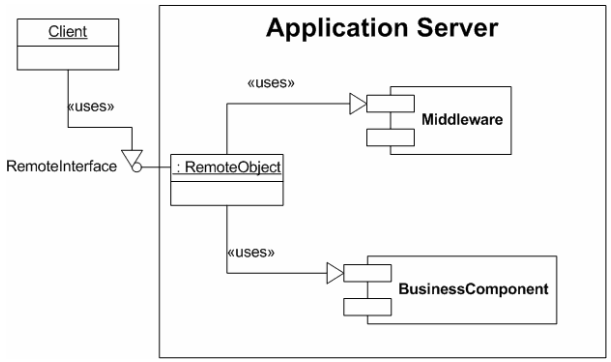
Middleware

- All these services together can be called Middleware because they don't implement our Business Logic, but yet they have to be present in our system
- Should be present in the Framework we use
- Should be more configured than implemented

Presentation Overview

- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Application Server



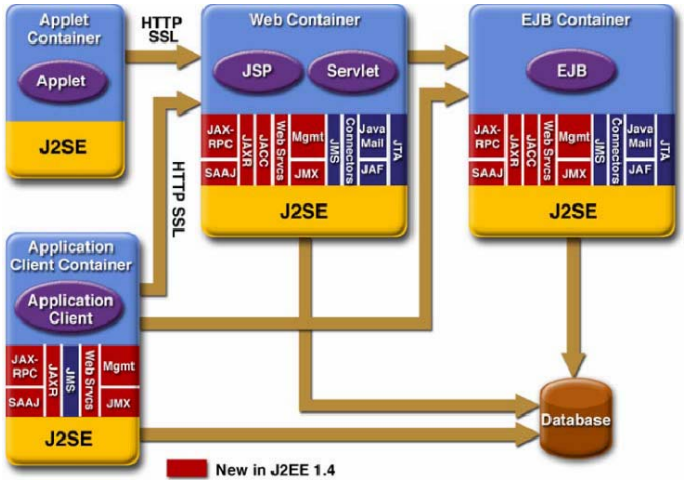
•Client uses remote interface
•Remote Object is managed by Application Server
•Transparent use of middleware
•Reduced dependencies

Java Enterprise

J2EE (Java 2 Enterprise Edition) defines various technologies specifications (JAXP, JMS, JNDI, JTA, JSP, JDBC).

Various vendors (BEA, IBM, Oracle, JBoss) implement these specifications and compete in the Application Server market.

J2EE stack



Microsoft .NET

Similar services are provided by the .NET platform.

Of course there's no one-to-one strict correspondence…

MS.NET	J2EE
ASP	JSP/JSF
DCOM	RMI
MTS/COM+	EJB
ADO	JDBC
ADSI	JNDI
MSMQ	JMS
DTC	JTA/JTS
…	…

Presentation Overview

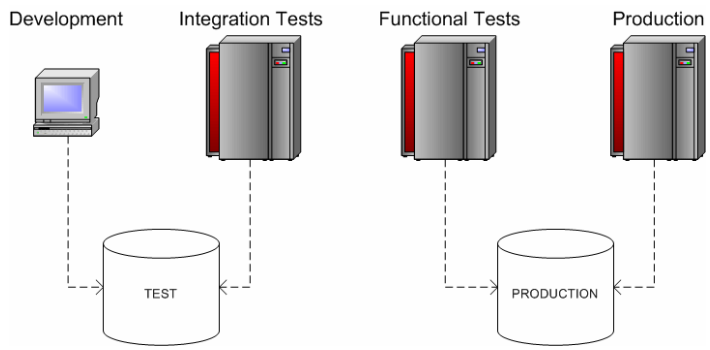
- What is "Enterprise Computing"
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Naming Services

- Map human-friendly names to objects
  - DNS
  - File System
  - LDAP

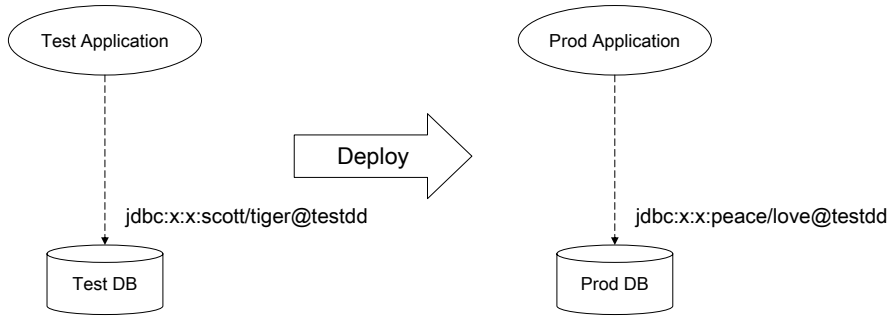
Adding this indirection layer we gain flexibility and portability.

Development and Deployment



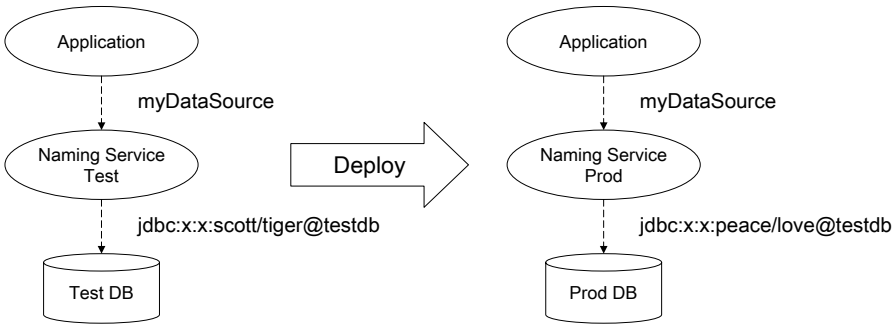
- Different Databases
- Different Hardware
- Different Operative Systems

Deployment dilemma



•There is a direct dependency between the application and the DB
•We must produce different “executables” for Test and Production environments
•Any change in the DB configuration will break our application

Enterprise Deployment



•No dependency between Application and DataBase
•No need for different Application versions
•Easier to maintain
•Separation of roles: Developer vs Application Server Administrator

Java Naming: JNDI

Java Naming and Directory Interface

Direct Connection

```
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection conn =
DriverManager.getConnection("jdbc:x:x:scott/tiger@testdd");
/* use the connection */
conn.close();
```

JNDI Connection

```
Context ctx = new InitialContext();
Object dsRef=ctx.lookup("java:comp/env/jdbc/mydatasource");
DataSource ds=(Datasource) dsRef;
Connection conn=ds.getConnection();
/* use the connection */
conn.close();
```

JNDI Configuration

using JBoss

```
<datasources>
   <local-tx-datasource>
        <jndi-name>comp/env/jdbc/mydatasource</jndi-name>
        <connection-url>jdbc:x:x:@testdd</connection-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <user-name>scott</user-name>
        <password>tiger</password>
   </local-tx-datasource>
</datasources>
```

- Application Server administrator manages this
- Application Server specific

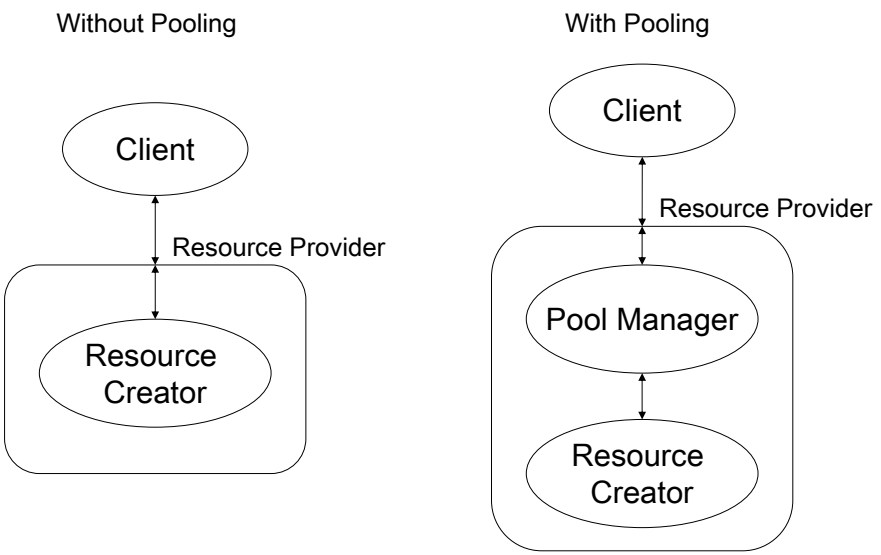
Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Pooling

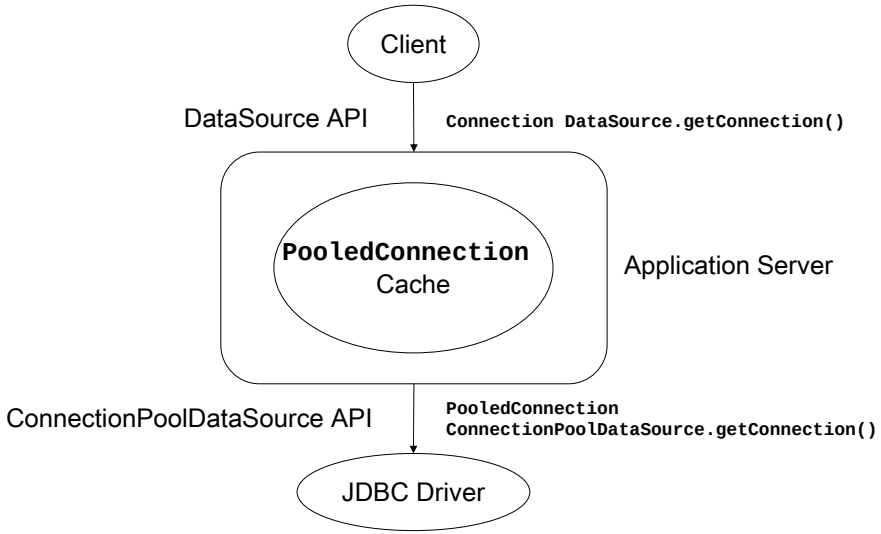
- *Pooling* means creating a pool of reusable resources
- Greatly improves performance if *creating* the resource is expensive (compared to *using* it)
- Should be completely *transparent* to the client

Pooling Schema

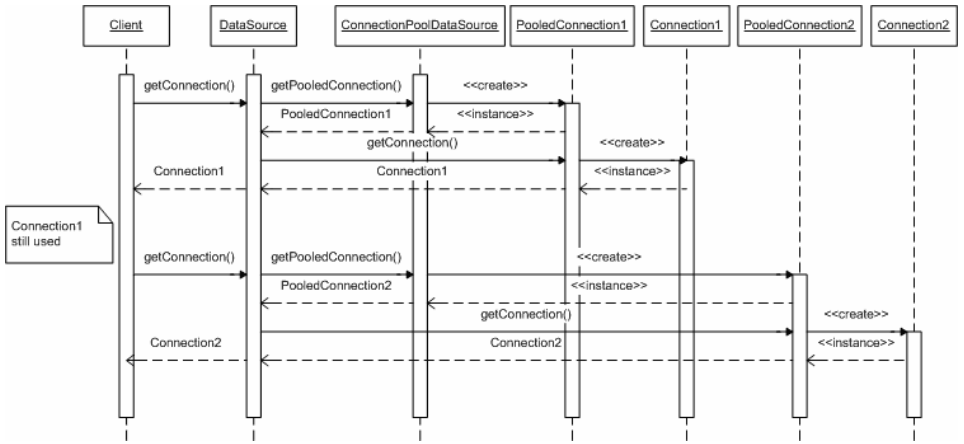


Java Pooling (JDBC)

Java DataBase Connectivity



Pooling Sequence



Java Code Example

JNDI Connection + Pooling

```
Context ctx = new InitialContext();
Object dsRef=ctx.lookup("java:comp/env/jdbc/mydatasource");
DataSource ds=(Datasource) dsRef;
Connection conn=ds.getConnection();
/* use the connection */
conn.close();
```

- Same code as before!
- Complexity completely hidden to developer
- No need to change java sources when pooling parameters change

Pooling Configuration

with JBoss

```
<datasources>
   <local-tx-datasource>
        <jndi-name>comp/env/jdbc/mydatasource</jndi-name>
        <connection-url>jdbc:x:x:@testdd</connection-url>
        <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>
        <user-name>scott</user-name>
        <password>tiger</password>

        <!-- Pooling parameters -->
        <min-pool-size>5</min-pool-size>
        <max-pool-size>100</max-pool-size>
        <blocking-timeout-millis>5000</blocking-timeout-millis>
        <idle-timeout-minutes>15</idle-timeout-minutes>
   </local-tx-datasource>
</datasources>
```

Presentation Overview

- What is “Enterprise Computing”
- Common Problems
- Real World Solutions
- Common Patterns
  - Naming Services
  - Pooling
  - Transaction Management

Transaction Management

What is a transaction?

An atomic unit of work. The work in a transaction must be completed as a whole; if any part of the transaction fails, the entire transaction fails.

Very well know problem that has been “solved” in databases for a long time.

ACID properties

Atomic: the transaction must behave as a single unit of operation. No partial work to commit

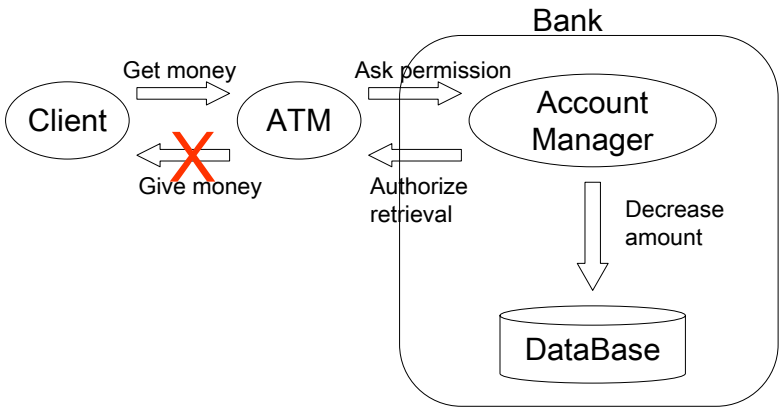
Consistent: either creates a new valid state or rolls back to the previous one

Isolated: a transaction in process and not yet committed must not interfere from all other concurrent transactions

Durable: committed data is saved in a way that the state can be restored even in case of system failure

SO/IEC 10026-1:1992 Section 4

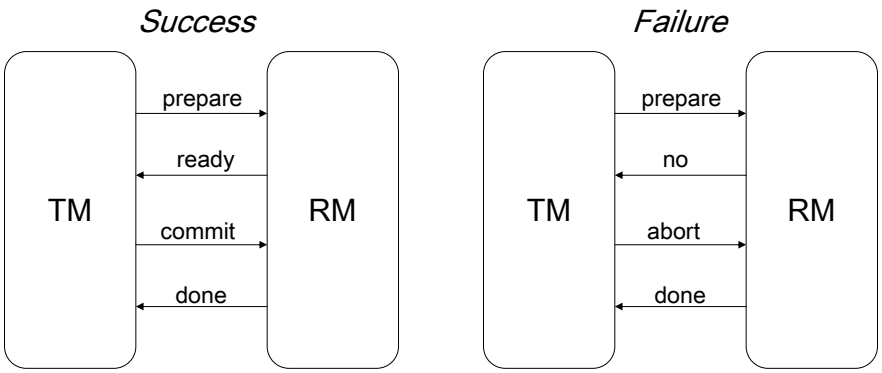
ATM Transaction example



We need to be able to manage distributed transaction to solve this class of problems.

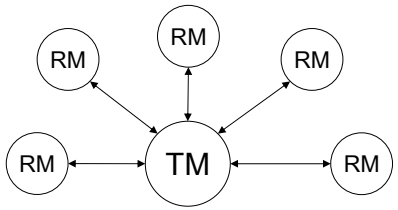
2-phase commit

- Transaction Manager [TM]
- Resource Manager [RM]



A log is kept for all operations, to let the TM recover a valid state in case of system failure

Distributed 2-phase commit



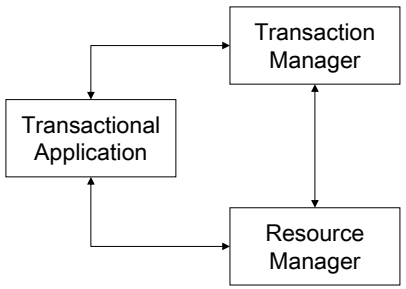
The TM repeats the 2-phase commit with every RM

- If the all RM answer “ready” the TM issues a global “commit”
- If at least one RM answers “no” the TM issues a global “abort”

Java Transactions (JTA)

Java Transaction API

Manage transactions in a *programmatic* way: you are responsible for programming transaction logic into your application code, that is calling begin(), commit(), abort().




```
Context ic = new InitialContext();
UserTransaction ut = (UserTransaction) ic.lookup(strTransJndi);
ut.begin();
// access resources transactionally here
ut.commit();
```


J2EE Declarative Transactions

It’s possible to specify at deploy time the transaction behavior.

The Application Server will *intercept* calls to the components and automatically begin/end the transaction on your behalf

```
<ejb-jar>
   <enterprise-beans>
      <session>
         <ejb-name>SomeName</ejb-name>
         …
         <transaction-type>Container</transaction type>
      </session>
   </enterprise-beans>
</ejb-jar>
```

Transaction types

```
<container-transaction>
    <method>
        <ejb-name>myComponent</ejb-name>
        <method-name>*</method-name>
    </method>
    <trans-attribute>Required</trans-attribute>
</container-transaction>
```

The J2EE application server manages different managed transaction types:

- Required: always run in a transaction. Join the existing one or starts a new one
- RequiresNew: always starts a new transaction
- Supports: joins the client transaction if any. Otherwise runs in no transaction
- Mandatory: transaction must already be running. Otherwise throws exception
- NotSupported: doesn't use transactions. Suspends client transaction if it exists
- Never: cannot be involved in a transaction. Throw exception if client has one

Conclusions

- You can solve any programming problem with an extra level of indirection
- except the problem of too many levels of indirection
- There are frameworks that already solve the most common and complex problems
- Understand the solution. Use the framework.
- Don't reinvent the wheel

Questions?

Resources

- J2EE tutorial (http://java.sun.com/j2ee/1.4/docs/tutorial/doc/)
- JBoss Docs (http://docs.jboss.org/jbossas/jboss4guide/r2/html/)
- Designing J2EE Apps
  (http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/DEA2eTOC.html)