Distributed Systems

Seminar 8 – Component-based Distributed Systems

Dr. Ing. Thomas Springer
Technische Universität Dresden
Chair of Computer Networks
Basics of Enterprise JavaBeans

a) Name the available types of EJBs and give an example of their usage!

b) What is the purpose and difference of local and remote interfaces?

c) What is the difference between application and container managed persistence?

d) What is the purpose and content of a deployment descriptor?
EJB Defines 3 component entity types

- **Session Beans**
  - Example: Shopping Basket
- **Message Driven Beans**
  - Example: Purchase Order Processing
- **Entity Beans**
  - Example: Customer
Solution E8.1b

- Local and Remote Interface

- Optimization for local Bean communication

- No overhead for marshalling/unmarshalling, stubs and skeletons
  - local calls can be about 10 times faster than remote calls

- Declared by java annotation
  - @local
  - @remote
- **Container-Managed:**
  - Automatic persistence mechanisms managed by container
  - With a *container-managed entity manager*, an *EntityManager* instance’s persistence context is automatically propagated by the container to all application components that use the EntityManager instance within a single Java Transaction Architecture (JTA) transaction.
  - EntityManager generated by Container
  - EntityManager context propagated by container

- **Application-managed Persistence:**
  - Bean has to implement when and how data is persistently stored
  - With application-managed entity managers, on the other hand, the persistence context is not propagated to application components, and the life cycle of EntityManager instances is managed by the application.
  - Own EntityManager provided by Application
Distribution Specifics and Deployment

- Deployment descriptor: name and type of beans, interfaces, class implementations
- Assembly descriptor: transaction management, security properties
- For EJB 3.0: configuration using annotations in source code
  - More convenient
  - But changes require change of source code

Example Deployment Descriptor:

```xml
<ejb-jar>
  <display-name>ShoppingBasket</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>ShoppingBasketBean</ejb-name>
      <remote>tud.onlinetrader.orderprocessing.ShoppingBasketRemote</remote>
      <ejb-class>tud.onlinetrader.orderprocessing.ShoppingBasketBean</ejb-class>
      <session-type>Stateful</session-type>
      <transaction-type>Container</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor> ... <assembly-descriptor>
</ejb-jar>
```
**Distribution Specifics and Deployment**

```xml
<assembly-descriptor>
  <container-transaction>
    <method>
      <ejb-name> ShoppingBasketBean </ejb-name>
      <method-name> initiateOrder </method-name>
    </method>
    <trans-attribute> RequiresNew </trans-attribute>
  </container-transaction> ...
  <method-permission>
    <role-name> CorporateClient </role-name>
    <method>
      <ejb-name> ShoppingBasketBean </ejb-name>
      <method-name> corporateOrder </method-name>
    </method>
    </method-permission> ...
  <assembly-descriptor>
```
Specify a simple banking application using EJBs which allows managing customers with multiple accounts, access customer and account data and performing money transactions.

a) Explain purpose and design steps of the four views on components using the banking application as example.

b) Sketch an UML component diagram for the banking application. Discuss the advantages of an explicit specification of required interfaces of components.

c) Discuss the architecture of the banking application with respect to an n-tier application.

d) What beans of what types are necessary to implement the application?

e) Specify the interfaces of these beans.

f) Discuss criteria that are relevant to choose an appropriate application server for the banking application?
Four Views on Component according to Cheeseman and Daniels:

1. Component Specification
   - Defines “casing” of a component; independent of its implementation
   - Specifies component’s offered and required interfaces
   - Interfaces specified separately from other component parts
   - Consideration of architecture independent from individual implementations
   - *Example*: Account management component with operations, e.g. using UML component and class diagrams

1. Component Implementation
   - Defines implementation of a component according to a given specification
   - Defines the “content” of the “casing”
   - Different implementations can belong to a particular specification
   - *Example*: Account Management Session Bean component with operations and binding to EJB like inherited methods for session bean type, specified using UML class diagram
3. Installed Component (Deployment)
   - Installation of component implementations on a component platform
   - Registers component implementation with component platform
   - Configuration information included separate from component code
     - Describes behaviour with/use of platform services by the component
     - E.g. transaction processing, persistence, security
   - *Example:* Account Management Session Bean deployed in JBoss application server, e.g. JBoss Deployment tool

4. Component Object
   - Describes a component in runtime
   - Represents an instance of a component
   - Possesses an identity
   - Encapsulates application logic and state information
   - *Example:* Account Management Session Bean instantiated in JBoss container (Pool for Session Beans), e.g. Runtime management tools
Components – Interfaces and Relationships

Extensive notation:

Usage relationship
ShoppingBasket requires IGoodsList in order to fully operate

Realization relationship
Ordering realizes behaviour of IGoodsList

Shortened notation (“lollipop Symbol” / “Ball & Socket”):

Dr. Thomas Springer
Solution E8.2b

Component-based Distributed Systems
Elements of UML component diagrams

UML Profile for EJB

- Component stereotypes
  - Extension mechanisms of UML
  - Specific component and interface types defined
  - Defined as specialization of Class

  - <<EJBEntityBean>>
  - <<EJBSessionBean>> (tagged Value={stateful}/{stateless})
  - <<EJBMessageDrivenBean>>

- Interfaces
  - <<EJBLocalInterface>>
  - <<EJBRemoteInterface>>
Solution E8.2d

- **Session**
  - AccountManagement (Stateful)
  - CustomerManagement (Stateful)
  - Transaction Management (Stateless)

- **Entity**
  - Account
  - Customer
  - Transaction
CustomerManagement (Stateful Session Bean)

- Long createCustomer(CustomerDetails details)
- void removeCustomer(Long customerId)

- List<CustomerDetails> getCustomersOfAccount(Long accountId)
- List<CustomerDetails> getCustomersOfLastName(String lastName)

- CustomerDetails getDetails(Long customerId)

- Helper Class: CustomerDetails for bundling all Customer attributed
  - Avoids sequence of get-requests
AccountManagement (Stateful Session Bean)
- Long createAccount(AccountDetails details, Long customerId)
- void removeAccount(Long accountId)
- void addCustomerToAccount(Long customerId, Long accountId)
- void removeCustomerFromAccount(Long customerId, Long accountId)
- List<AccountDetails> getAccountsOfCustomer(Long customerId)
- List<Long> getCustomerIds(Long accountId)
- AccountDetails getDetails(Long accountId)
- Helper Class: AccountDetails
TransactionManagement (Stateless Session Bean)

- void deposit(BigDecimal amount, String description, Long accountId)
- void withdraw(BigDecimal amount, String description, Long accountId)
- void transferFunds(BigDecimal amount, String description, Long fromAccountId, Long toAccountId)

- TxDetails getDetails(Long txId)
- List<TxDetails> getTxsOfAccount(Date startDate, Date endDate, Long accountId)

- Helper Class: TxDetails
Solution E8.2e

- **Customer**
  - customer_id (PK)
  - last_name
  - first_name
  - middle_initial
  - street
  - city
  - state
  - zip
  - phone
  - email

- **Account**
  - account_id (PK)
  - type
  - description
  - balance
  - credit_line
  - begin_balance
  - begin_balance_time_stamp

- **Tx**
  - tx_id (PK)
  - account_id (FK)
  - time_stamp
  - amount
  - balance
  - description

- **Customer_Account_Xref**
  - customer_id (FK)
  - account_id (FK)
What criteria are relevant for the selection of an application server product in a project?

- Cost/Ownership
- Appropriateness according to adopters requirements
- Security
- Extensibility
- Standard Conformance
- Interoperability
- Configurability
- Integration of existing data sets
- Integration of legacy systems
- Integration into existing communication and IT infrastructure
- Scalability support
- Robustness and availability
- Multi-user support
- Performance
- Short development time/time to market for new applications
OSGi is a java-based component platform.

a) How is a component in OSGi defined?

b) Compare the OSGi and EJB component model!
Components in OSGi are called Bundles
- interface definition
- java classes and further resources
- bundle description - manifest file

Unit of composition is service
Unit of deployment is bundle

Bundle compiled into JAR
Manifest
- describes contents of JAR
- deployment information, e.g. import and export of java packages for shared use

Can provide services for other bundles
- based on interface specification (standard java interface)
- registered at registry service of OSGi platform (Service Registry Service)

Framework itself contains of bundles
Solution E8.3a

Bundle - Lifecycle

Install
- Evaluation of manifest file
- Checking availability of bundle classes and resources
- Persistent saving

Resolve
- Resolution of static dependencies to external bundles and java packages

Uninstall
- Must take place explicitly through the OSGi platform
- Removal of all persistently saved parts of the bundle
<table>
<thead>
<tr>
<th>Tier</th>
<th>EJB</th>
<th>OSGi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component installation</td>
<td>Server-side component model</td>
<td>(Client-side) local component model (extensions for remote comm: via WS)</td>
</tr>
<tr>
<td>Component installation</td>
<td>Component installation during deployment, later fix and not changeable (hot deployment)</td>
<td>Search and installation of bundles at runtime supported</td>
</tr>
<tr>
<td>Binding</td>
<td>JNDI naming service for binding between components</td>
<td>Bundles registered in local repository</td>
</tr>
<tr>
<td>Remote Communication</td>
<td>Remote communication (RMI) via remote interface specification</td>
<td>Local communication between bundles only, remote comm. Not integrated into platform (discovery, registration, invocation)</td>
</tr>
</tbody>
</table>
Debugging of Distributed Systems

a) Explain the problems of “The lack of a global state”, “Indeterminism” and “Interference”. Suggest a solution for each problem.

b) Three processes communicate with each other as described in the sequence diagram below. Insert the values of the event counter according to Lamport. (Local events and inter-process events are marked by points).

c) With reference to the sequence example in the lecture (slide 8.28), give a causal distributed break point for the logical time point t34.

d) For this example, give two ordered and unordered event pairs.
### Indeterminism
- Execution of distributed apps on distributed nodes
  - Separated memory
  - No unique/synchronized time base
  - Parallel execution of numerous processes and threads
  - Different/varying load/usage of network and hosts
- Execution time of components not fixed/known – not predictable
- Order of messages/events not predictable
- Even with same input different program flow and results possible
  - System settings in case of errors hardly reproducible

### Lack of global state
- Up-to-date state from every node on one node for debugging
- Not possible due to latency in message exchange and indeterminism
- Messages have to be exchanged to get the distributed states to a central node
  - Message delay causes reordering and inconsistency
  - Timing, logic ordering of events
- No synchronized clocks

### Interference
- Between Debugger and „System under test“
- Debugger consumes resources and thus, influences control flow of system under (CPU, memory, network, etc.)
- Delay of local processing and on network due to debugging
**Solution E8.4a**

### Solutions for Indeterminism and Lack of global state

- Testing of different possible execution sequences via distributed Single Step
- Re-execution / Replay via output recording
- Use Lamport approach to record all events
  - Establishing global partial order between events in distributed processes
  - Allows reproduction of event order

**Approaches:**
- Re-execution (whole system)
  - Recording of all inter-process events
  - Control of repeated execution based on recorded events
- Replay (single process and inter-process events)
  - Also involves recording of all inter-process events
  - Replay of only a single process possible by feeding in its recorded inputs
  (important also for technical/engineering processes with peripheral devices)

### Solution for Interference

- Minimizing the influence of debugger to running system by minimizing resource consumption and communication overhead of debugger
- Replay limits debugging on a single node, no communication overhead
- Support for Distributed Breakpoints allows execution in „normal“ speed and stopping at defined breakpoints
• Realization of the algorithm
  • Each process has local event counter $Z$ (initially “0”)
  • Each inter-process event has a number $N(E)$, as well as the messages ($\delta = N(E)$)

• Intra-process Event:
  o $Z := Z + 1$
  o $N(E) := Z$

• Sending:
  o increment of $Z$ ($Z := Z + 1$)
  o mark Sending Event: $N(E) := Z$
  o mark message: $\delta := Z$

• Receiving of message with number $\delta$
  o if $\delta > Z$ (Receiver) set $Z := \delta + 1$
  o otherwise set $Z := Z + 1$
  o Receiving Event $N(E) := Z$

\[ \text{If } E_\sigma \rightarrow E_\rho \text{ then } N(E_\sigma) < N(E_\rho) \]
• Each process can create local order of events
• Across processes this local order is not necessarily valid
• E.g. 7 is used for different events by P1 and P3
• Adding process identifier for unique distinction
  • P1-7, P3-7
Solution E8.4c

- **Problem:**
  - Time delay after issuing of a halt-command

- **Approach:**
  - Backtracking to consistent state directly before a stopping event ("reset line")

- **Procedure:**
  - Backtracking of the causal contexts regarding to the predecessor-relation of messages
  - Minimum requirement: sender(s) of received message(s) must be identifiable in breakpoint (i.e. send event is part of the breakpoint)

![Solution Diagram]

Break point at time $t_{34}$: HP bei $t_{14} ; t_{23}$
Ordered/unordered event pairs

- Ordered event pairs: \((t_{31}; t_{34}), (t_{32}; t_{12})\)
- unordered event pairs: \((t_{32}; t_{11}), (t_{23}; t_{33})\)