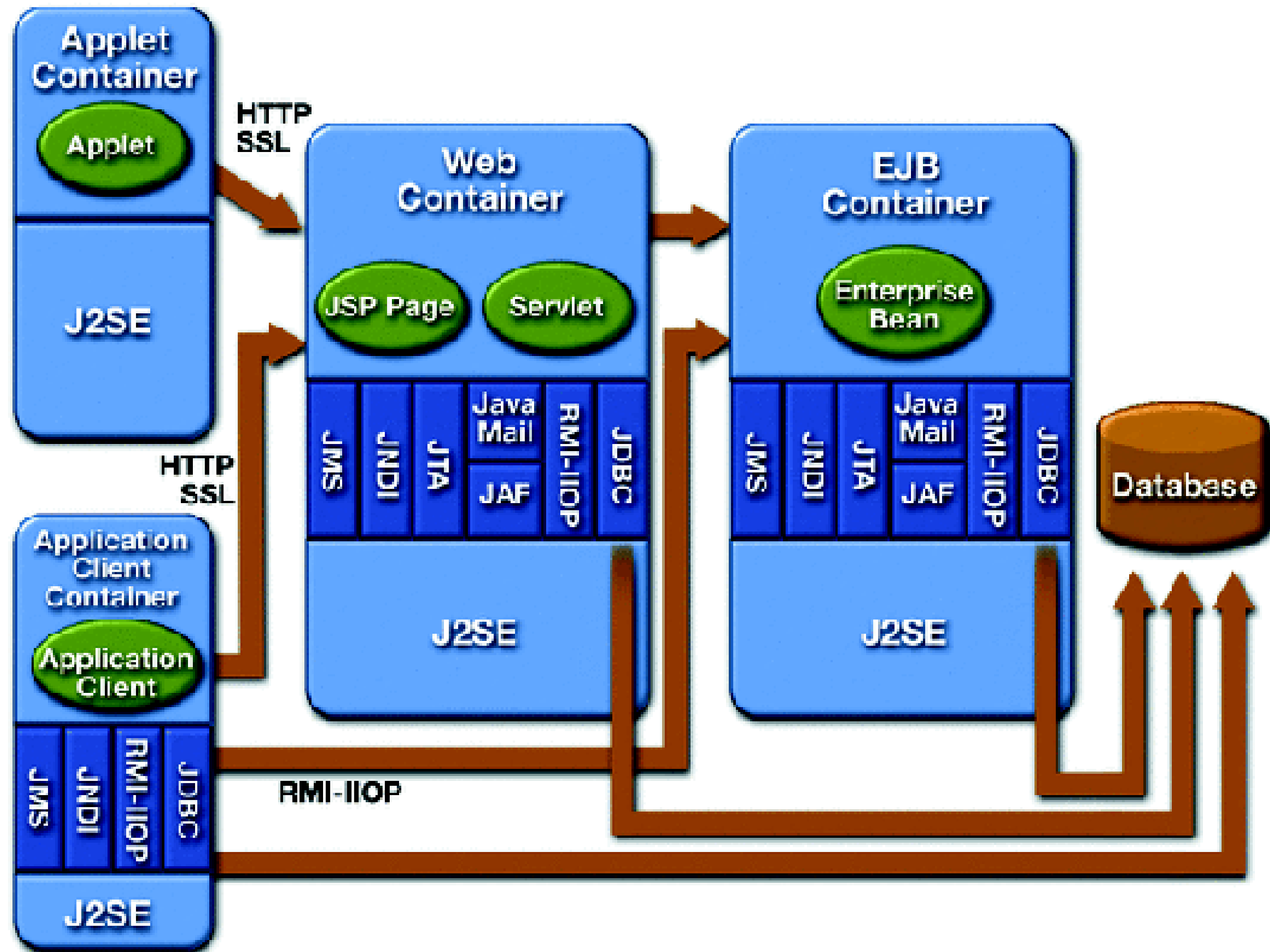
Programmieren verteilter Systeme 1

Modul 12: JMS & JavaMail

Dr. Franz Lackinger

Die J2EE Plattform



Java Mail

Inhalte

- (Internet) Mail Protokolle
- Was ist JavaMail ?
- Verwendung von JavaMail

Mail Protokolle

- Senden von e-mail messages
  - Simple Mail Transfer Protocol (SMTP)
  - Multipurpose Internet Mail Extensions (MIME)
- Empfang von e-mail messages
  - Post Office Protocol 3 (POP3)
  - Internet Message Access Protocol (IMAP)

JavaMail Architektur

JavaMail Client

Aufrufe zum Versenden vom Emails

Aufrufe zum Empfangen und Verwalten vom Emails

Abstrakte Klassen

Transport

Store, Folder, ...

Implementierung:

SMTPTranport

Implementierung:

IMAPStore, IMAPFolder, ...

SMTP Server

IMAP Server

JavaMail API

- `class javax.mail.Session`
  - Einstiegspunkt. Enthält Konfigurationen und Eigenschaften.
- `class javax.mail.Message`
  - Repräsentiert eine Email.
- `class javax.mail.Store`
  - Repräsentiert die „Mailbox“ mit Ordnern und Emails.
- `class javax.mail.Transport`
  - Repräsentiert das Transportprotokoll

Vorteile von JavaMail

- Unabhängig von Platform und Protokoll
- Abstrakte Schicht über Standards wie
  - SMTP, POP3, IMAP
- Offen für neue Protokolle
- High-Level Interface
  - Die Details der low-level Protokolle müssen nicht berücksichtigt werden

Beispiel:

Booking .JSP

Booking EJB

CreditCard

validate( )

Order

create( )

Send E-Mail!

Schritt 1: Setzen der Mail Server properties

```
Properties mprops = new Properties();
mprops.put("mail.transport.protocol",
    "smtp");
mprops.put("mail.smtp.host",
    "smtp.sun.com");
```

Schritt 2: Hole Session Information

```
Session session =
    Session.getDefaultInstance(mprops,
    null);
```

- Erzeugt eine mail session zwischen dem Mailclient und dem Mailserver

Schritt 3: Erzeuge eine neue Message

```
Message msg = new MimeMessage(session);
msg.setFrom(new InternetAddress
    ("franz.lackinger@sun.com));
InternetAddress[ ] address = {new
    InternetAddress(args[0])};
msg.setRecipients(Message.RecipientType.TO,
    address);
msg.setSubject("Bestätigung");
msg.setText("Danke für Ihre Bestellung!");
```

Schritt 4: Send Message

```
Transport.send(msg);
```

MIME und JAF

- **MIME** = Multipurpose Internet Mail Extensions
  - Definiert ein „Message Representation Protocol"
  - „text/plain", „text/html", „image/jpeg", „video/mpeg", ...
- **JAF** = Java Activation Framework
  - weist MIMETypes DataHandler Beans zu
  - einheitliches Interface zu Daten aus verschiedenen Quellen und unterschiedlichen Formaten.

Multipart MIME Messages

Message

Header Attributes

Message Atrribut
ContentType „multipart/mixed“

Content Body

DataHandler Object

Enthält ein Multipart Objekt, anstelle von Daten

MultiPart Object

BodyPart Object

Header Attributes

Message Atrribut
ContentType = „text/plain“

Content Body

DataHandler Object

Enthält Text Daten

BodyPart Object

Kann weitere Daten oder Multipart Objekte enthalten

Weitere Infos

- JavaMail http://java.sun.com/products/javamail/
- JAF (JavaBeans Activation Framework) http://java.sun.com/products/javabeans/glasgow/jaf.html
- Artikel:
  - „JavaMail Quick Start" http://www.javaworld.com/javaworld/jw-10-2001/jw-1026-javamail.html
  - „Java meets e-mail" http://www.developer.com/java/other/article.php/606531
  - Kurzlehrgang: „jGuru: Fundamentals of the JavaMail API" http://java.sun.com/developer/onlineTraining/JavaMail/

JMS – Java Message Service

Szenario

Parts Inventory

Inventory

Factory

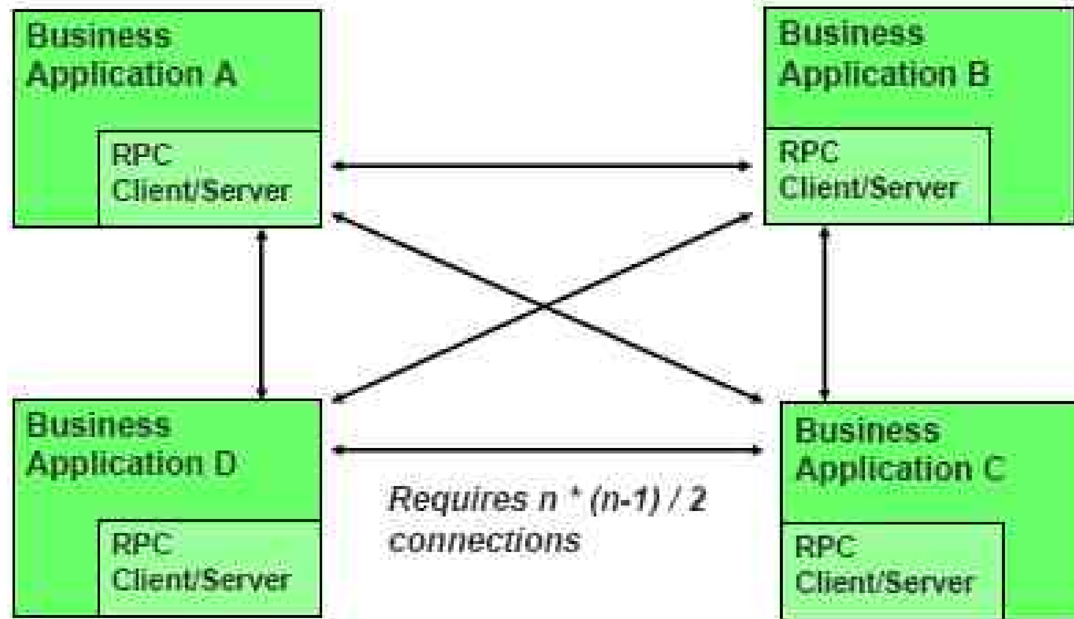
Parts

Parts Order

Sales

Accounting

Herkömmliches RPC



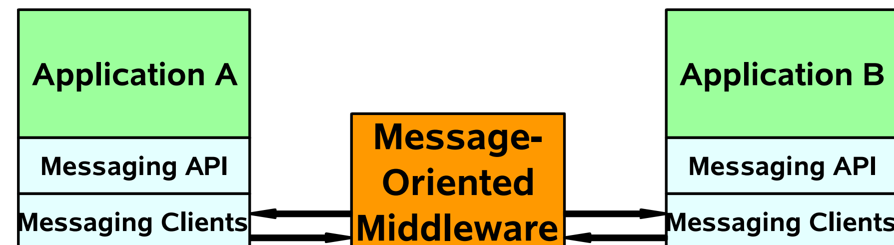
- Point-to-Point
- Benötigt n*(n-1)/2 Verbindungen
- Alle Applikationen müssen online sein
- Detailliertes Wissen über jede Applikation notwendig

Enterprise Messaging mit JMS (1)

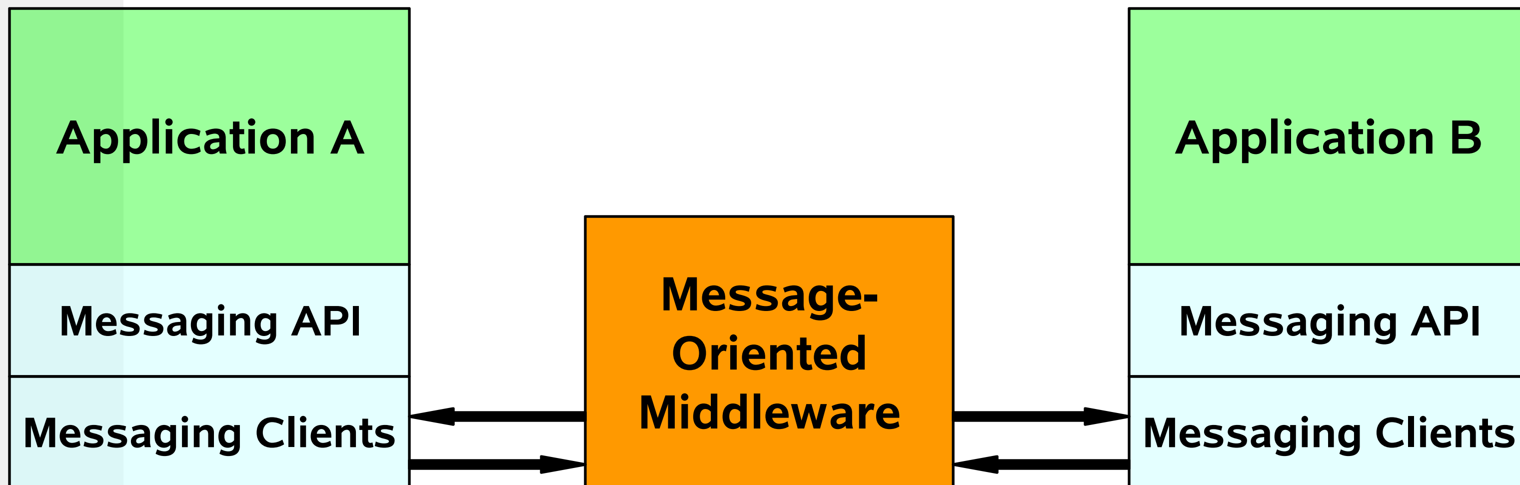
Messaging zur Kommunikation zwischen verschiedenen Applikationen oder Komponenten

- Netzwerk- & protokollunabhängig
- Synchrone & asynchrone Übertragung
- 2 Messaging Domains (Peer-to-Peer &Publish / Subscribe)
- Store-and-Forward Verfahren
  - Speicherung der Messages in der Middleware bei Nicht-Verfügbarkeit des Empfängers (asynchrone Übertragung)
  - Middleware garantiert Auslieferung der Messages
- Unterstützung von Transaktionen

⇒ Leichte Integration von bereits bestehenden Applikationen (EAI)



Enterprise Messaging mit JMS (2)



- Keine Spezifikation eines proprietären Protokolls durch JMS
  - ⇒ Spezifikation durch den Provider (JORAM, SonicMQ, JBossMQ, etc.)
    - ⇒ Client & Server müssen vom gleichen Provider sein
- Spezifikation einer Messaging API für Applikationen durch JMS, welche von den Herstellern zur Verfügung gestellt wird
  - ⇒ Austausch von Providern ohne Anpassung der Applikationen möglich
  - ⇒ Server & Client müssen vom gleichen Provider stammen

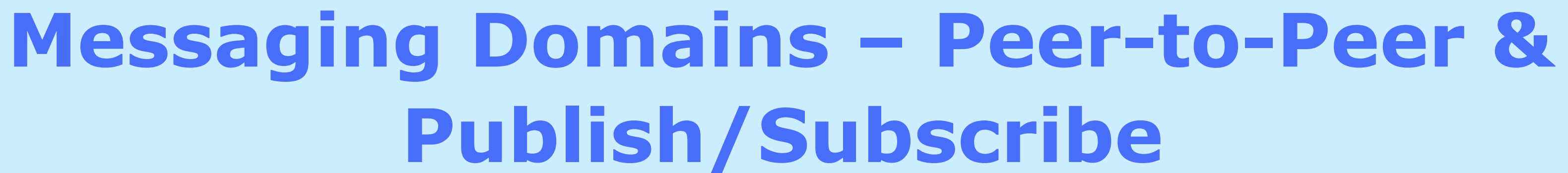
JMS ist ein API

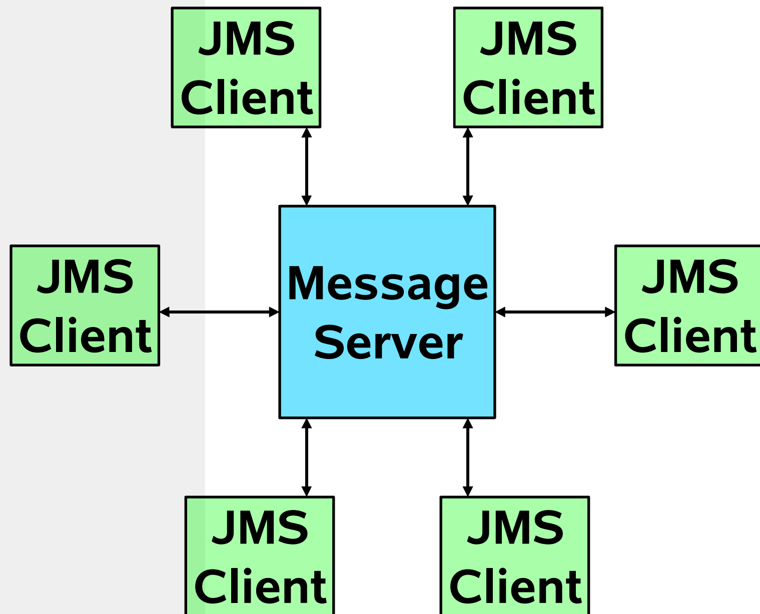
Java™ Application

JMS API

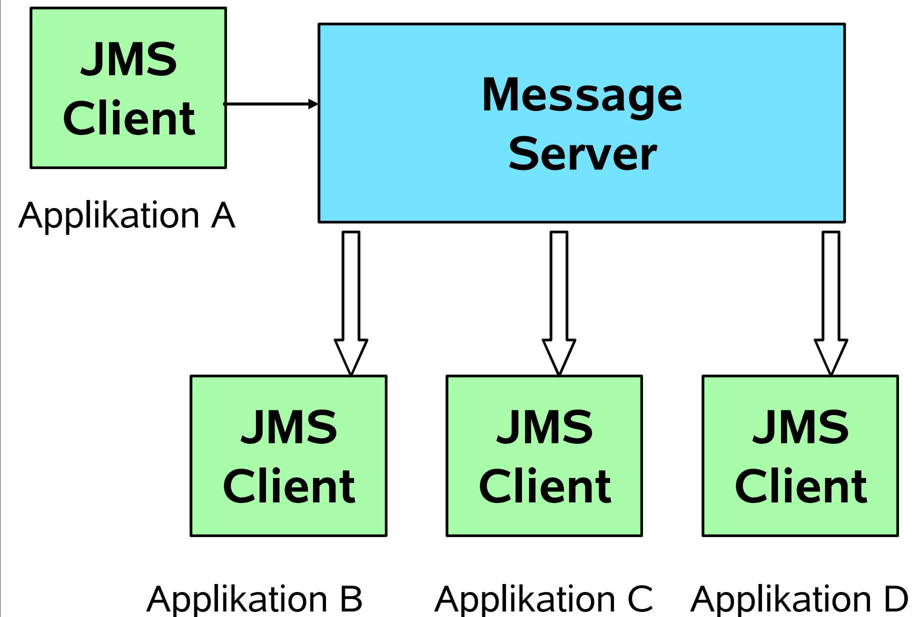
JMS Provider	JMS Provider	JMS Provider	JMS Provider	JMS Provider
IBM MQSeries	Progress SonicMq	Fiorano	BEA	SUN MQ

Messaging Domains – Peer-to-Peer & Publish/Subscribe





- Ein Empfänger pro Message
- Jeder Absender kennt den Empfänger
- Einsatz vorzugsweise, wenn Empfänger den Erhalt der Message bestätigen muss



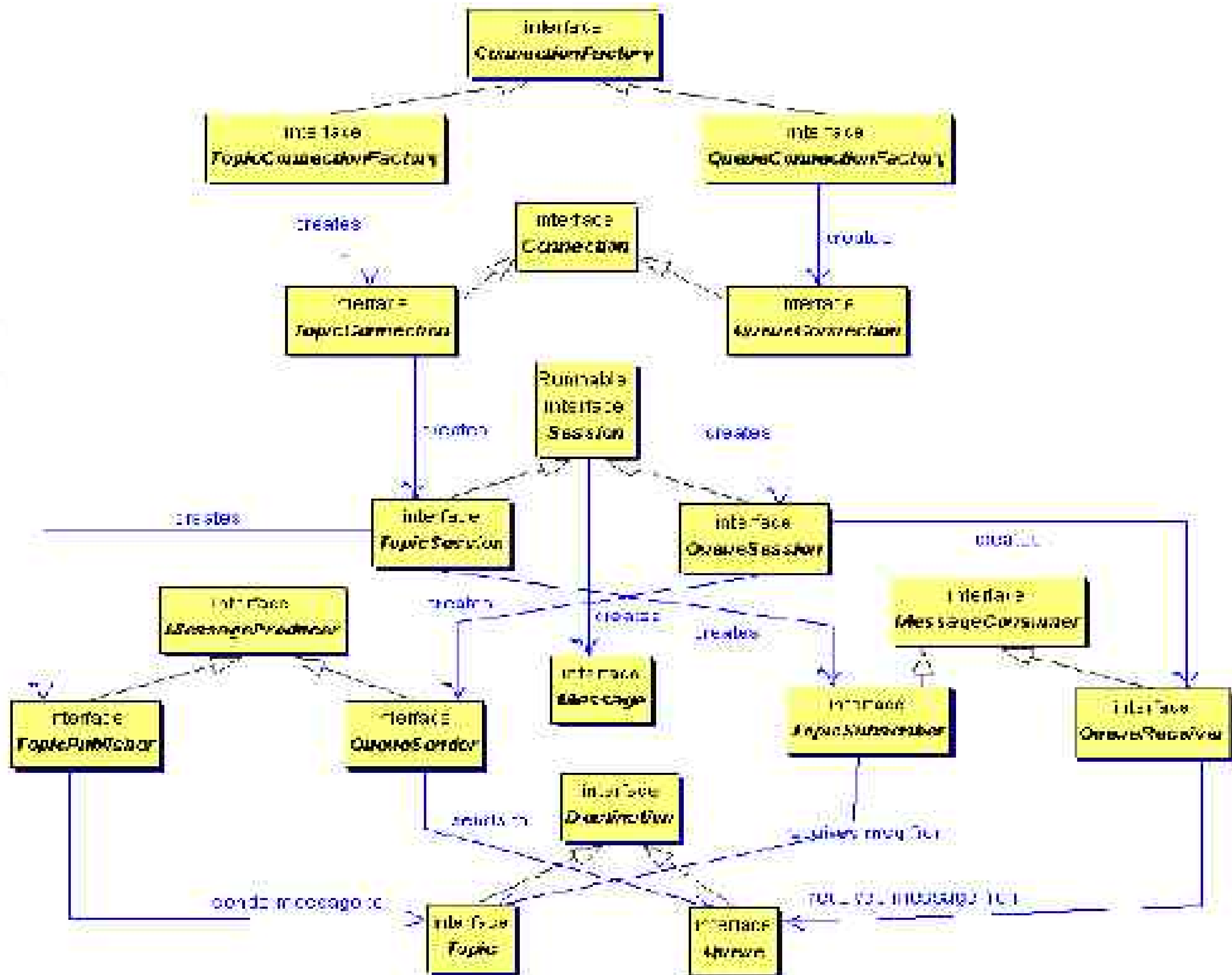
- Viele Empfänger pro Message
- Keine Kenntnis der Applikationen untereinander

JMS - Architektur

Eine JMS Applikation besteht aus folgenden Komponenten:

- **JMS Clients – Java Programme, die Messages senden und empfangen**
- **Non-JMS Clients – Clients, die die ursprüngliche API des Message Systems nutzen.**
- **Messages – Jede Applikation definiert eigene Messages die zur Kommunikation zwischen den Clients verwendet werden.**
- **JMS Provider – Ein Messaging System, das neben einer JMS Implementierung auch Werkzeuge für die Administration und Kontrolle des Systems zur Verfügung stellt.**
- **Administered Objects – Vorkonfigurierte JMS Objekte, die durch einen Administrator erzeugt wurden, und den Clients zur Verfügung stehen. Hierbei gibt es zwei Typen von JMS administered Objects:**
  - **Connection Factory – Ein Objekt, das der Client zur Verbindung mit dem Messaging System benutzt.**
  - **Destination – Dieses Objekt verwendet der Client um seine Messages zu Adressieren (Zielobjekte).**

JMS – Interface(1)

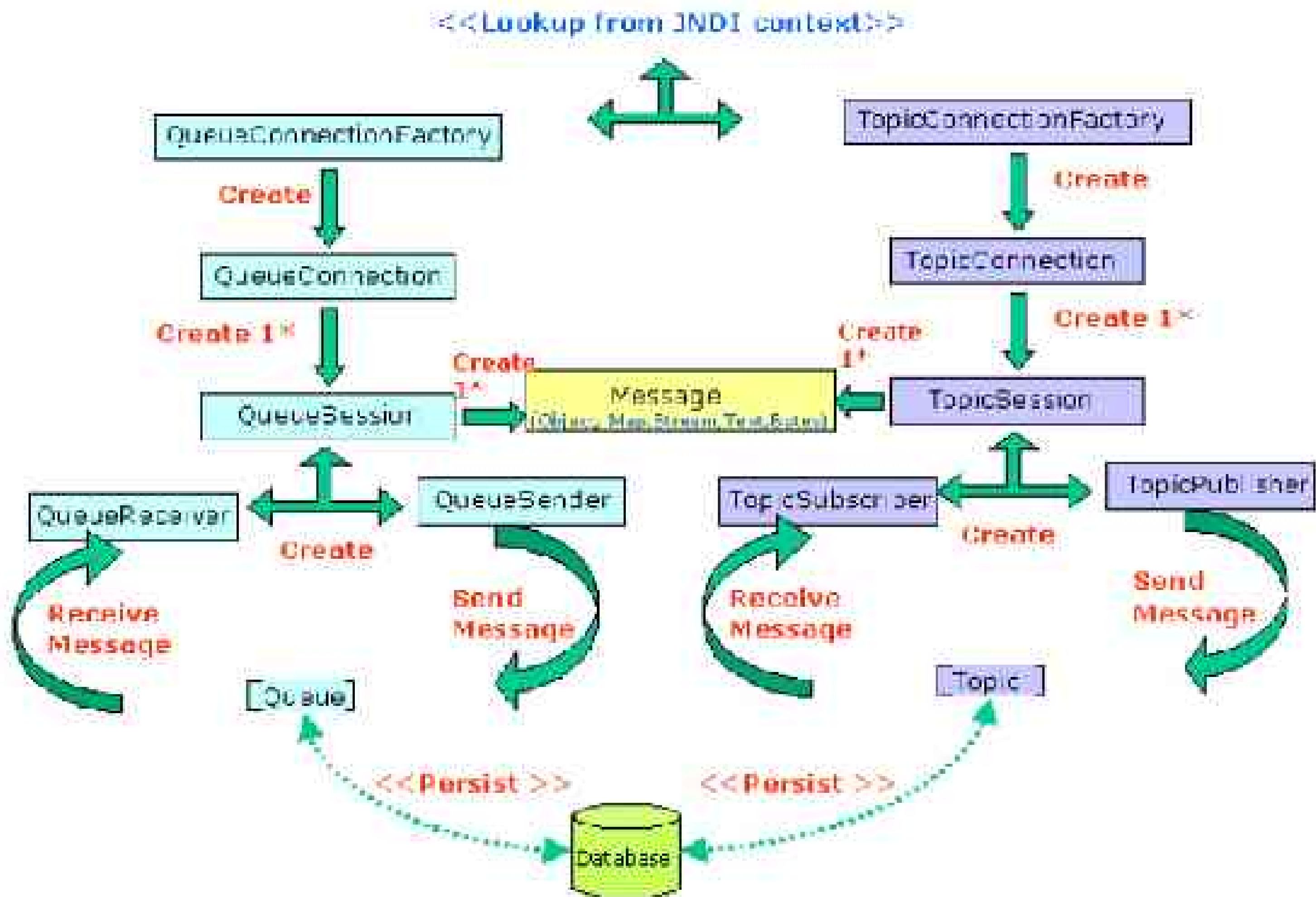


JMS – Interface(2)

Das JMS - Interface:

- ConnectionFactory – ein administered Object das der Client nutzt, um eine Verbindung herzustellen
- Connection – eine aktive Verbindung zu einem JMS Provider
- Destination – ein administered Objekt das die Identität eines Message Empfängers kapselt.
- Session – ein single threaded Context zum Senden und Empfangen von Nachrichten.
- Message Producer – ein durch eine Session erzeugtes Objekt, das zum versenden einer Nachricht an einen Empfänger genutzt wird.
- Message Consumer – ein Objekt das eingehende Messages empfängt.

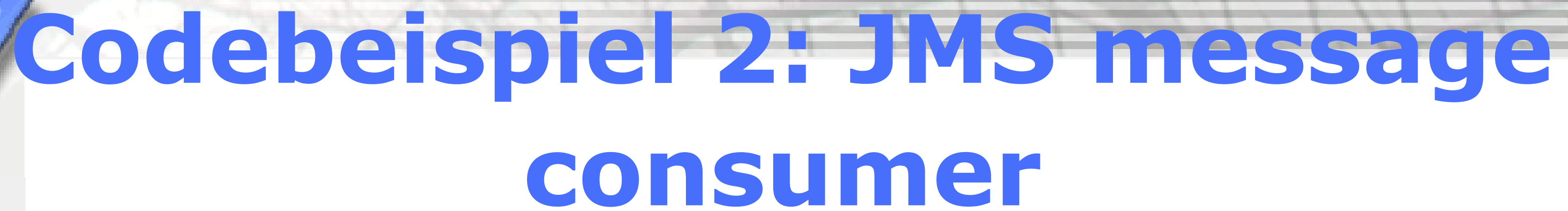
Ablauf der Kommunikation



Codebeispiel: JMS message producer

```
public class HelloQueueSender {
    public static final String D_NAME  = "ex1Queue";
    public static final String CF_NAME = "QueueConnectionFactory";

    public static void main(String[] args) {
        try {
            Context                ctx   = new InitialContext();
            QueueConnectionFactory qcf   = (QueueConnectionFactory) ctx.lookup(CF_NAME);
            Queue                  q     = (Queue) ctx.lookup(D_NAME);
            QueueConnection        qc    = qcf.createQueueConnection();
            try {
                QueueSession       qsess = qc.createQueueSession(false,
                                               Session.AUTO_ACKNOWLEDGE);
                QueueSender        qsnd  = qsess.createSender(q);
                TextMessage        msg   = qsess.createTextMessage("Hello JMS World");
                qsnd.send(msg);
            } finally {
                try {qc.close();} catch (Exception e) {}
            }
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.toString());
        }
    }
}
```



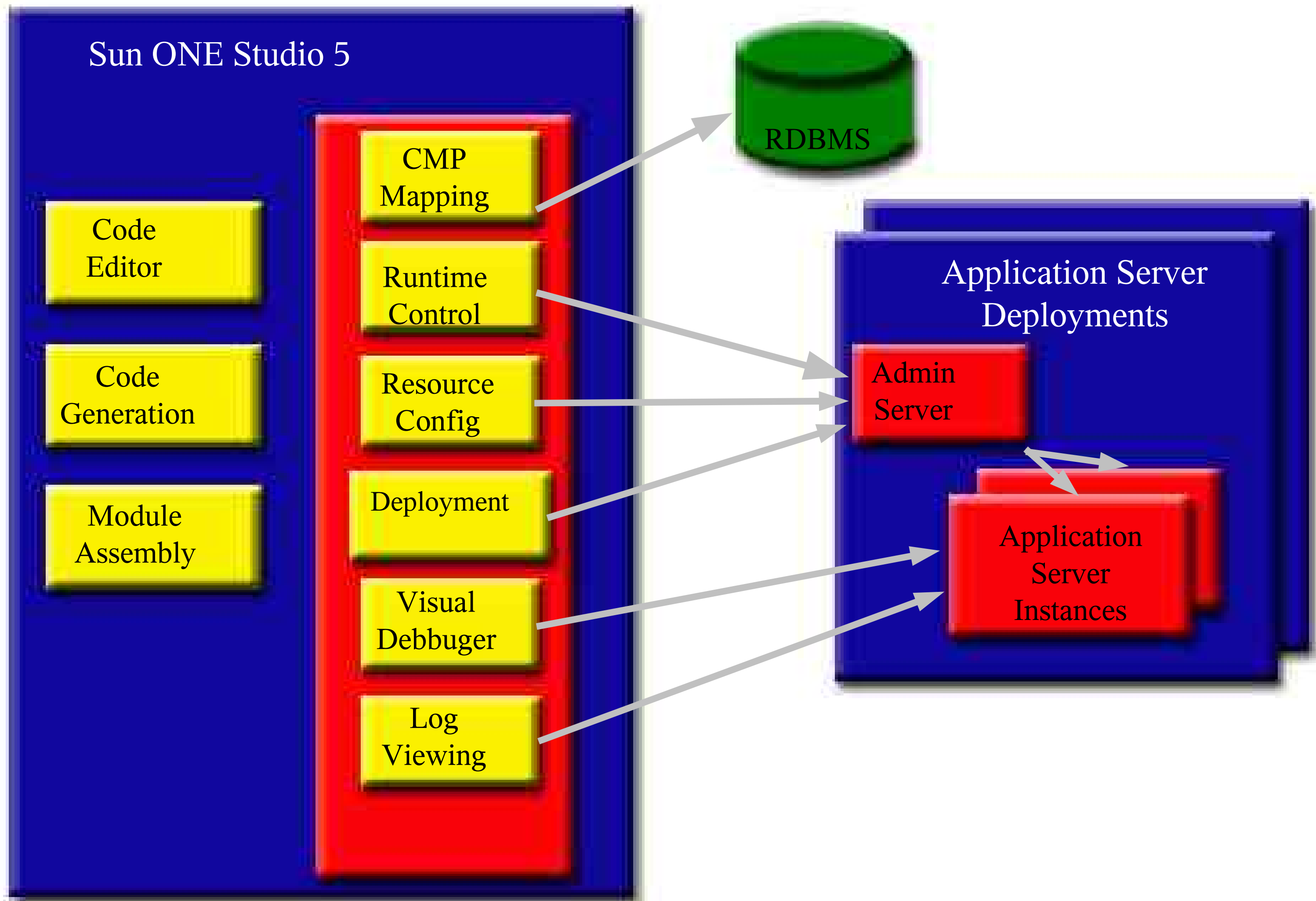
Codebeispiel 2: JMS message consumer

```
public class HelloQueueReceiverAsynch implements MessageListener {
    public static final String D_NAME  = "ex1Queue";
    public static final String CF_NAME = "QueueConnectionFactory";

    public void onMessage(Message message) {
        try {
            TextMessage msg   = (TextMessage) message;
            System.out.println("Received: " + msg.getText());
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.toString());
        }
    }

    public static void main(String[] args) {
        try {
            Context                ctx   = new InitialContext();
            QueueConnectionFactory qcf   = (QueueConnectionFactory) ctx.lookup(CF_NAME);
            Queue                  q     = (Queue) ctx.lookup(D_NAME);
            QueueConnection        qc    = qcf.createQueueConnection();
            QueueSession           qsess = qc.createQueueSession(false,
                                                Session.AUTO_ACKNOWLEDGE);
            QueueReceiver          qrcv  = qsess.createReceiver(q);
            qrcv.setMessageListener(new HelloQueueReceiverAsynch());
            qc.start();
        } catch (Exception e) {
            System.out.println("Exception occurred: " + e.toString());
    }
}
```

Exkurs: SunONE Studio + SunONE Application Server 7



Literatur

- http://java.sun.com/products/jms/
- http://www.jguru.com/faq/JMS
- Monson-Haefel, R.; Chappell, D. : Java Message Service