erasm platform
Bild- / Galerieverwaltung

Komponentenprogrammierung
mit J2EE, WebServices, .NET

Lukas Neitsch
Silvan Golega
Gero Decker
Uwe Kylau

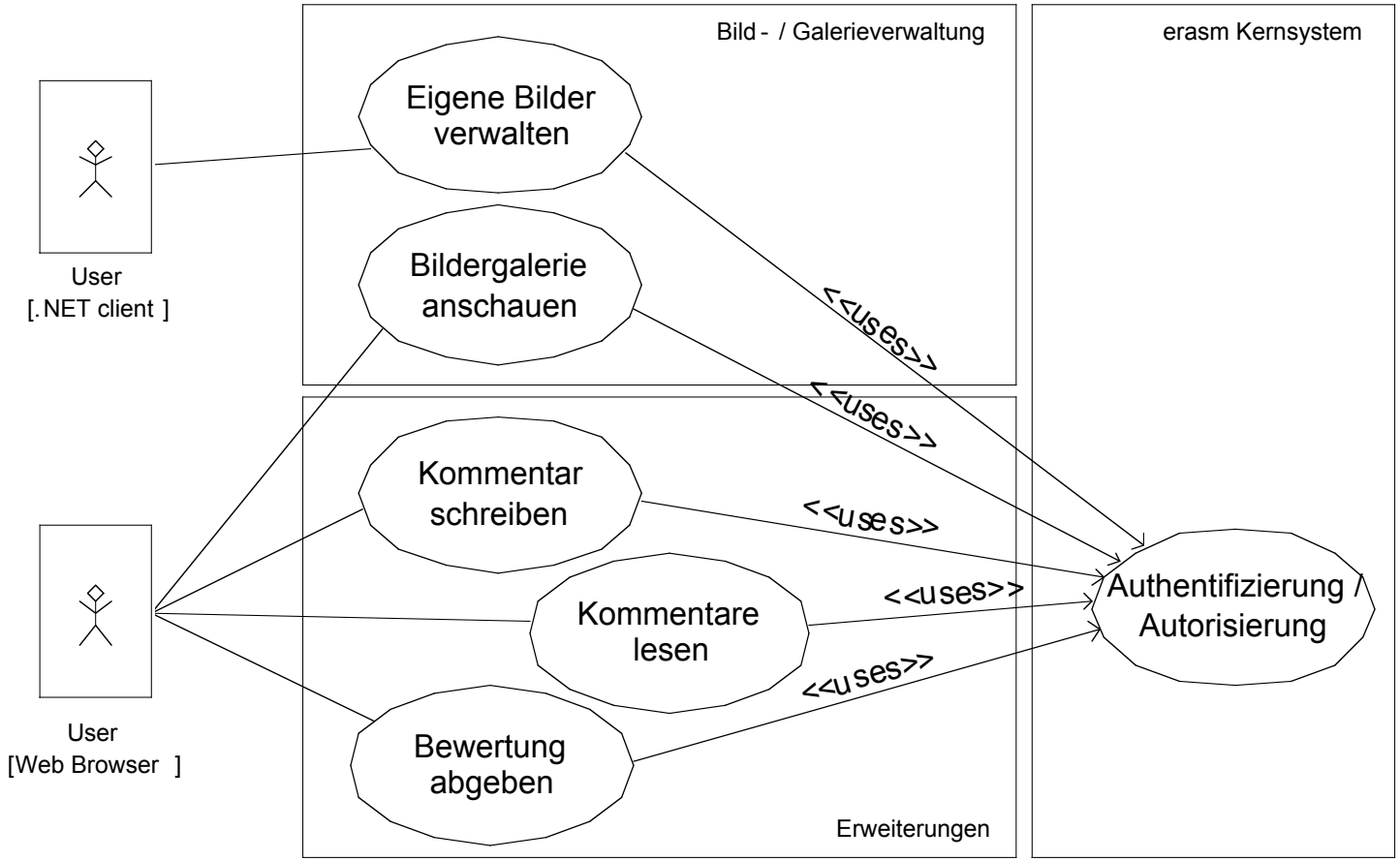
HASSO-PLATTNER-INSTITUT
für Softwaresystemtechnik

Inhalt

- Anforderungen, verwendete Systeme
- Projektplanung / -management
- Architektur
- Vorführung WebGUI / Komponentendeployment
- Test, Bewertung J2EE
- Vorführung .NET Client
- Vergleich WebServices: jboss vs. .NET
- Erweiterungen
- Literatur

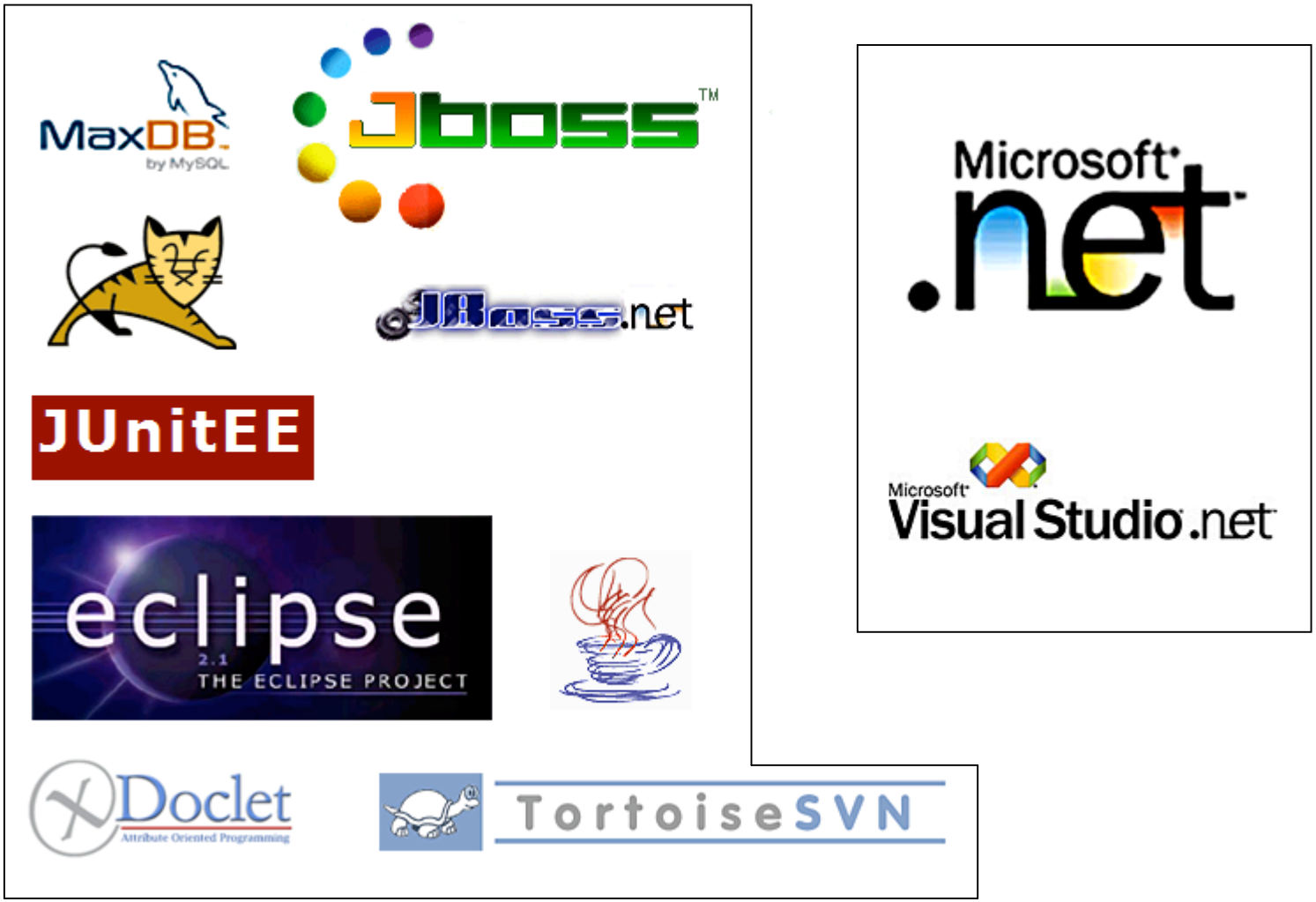
Anforderungen – Use Cases

- **erasm platform**
  - Web-Portal
  - Zentrale Weitergabe / Archivierung von Informationen / Medien
- **Bild- / Galerieverwaltung**

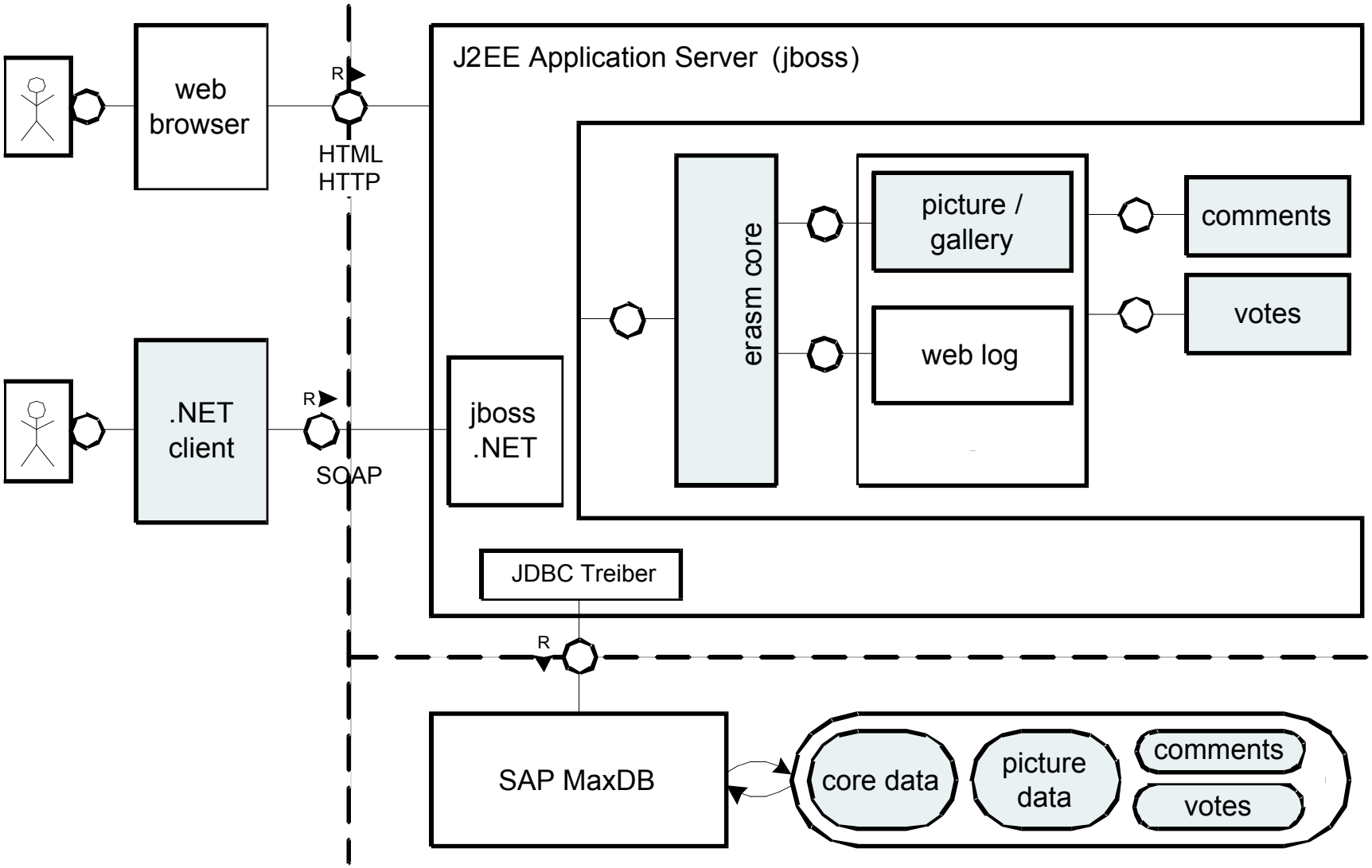


Verwendete Produkte und Technologien

- **Ziel: Möglichst Verwendung von Open Source Systemen**

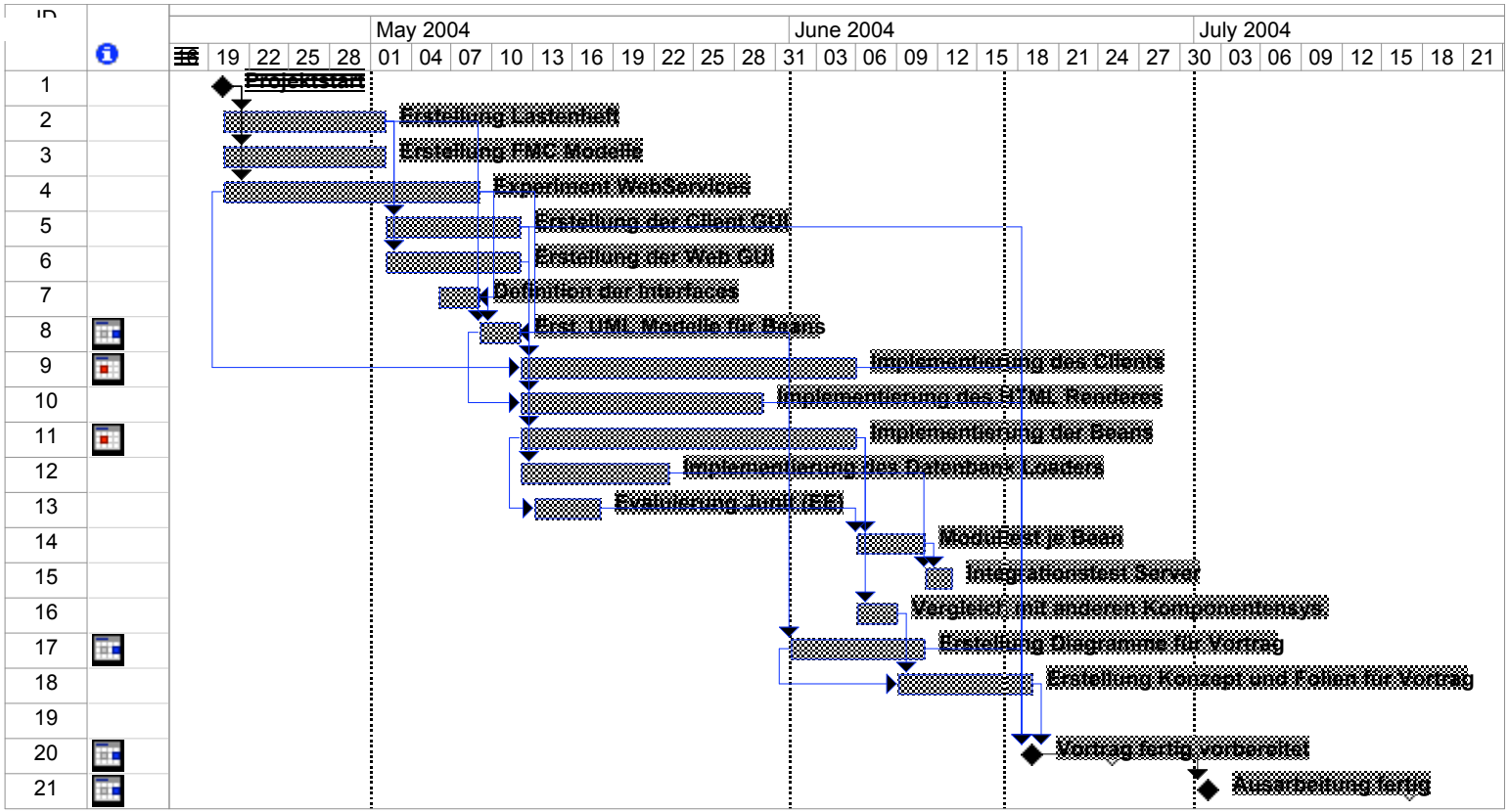


Grober Aufbau



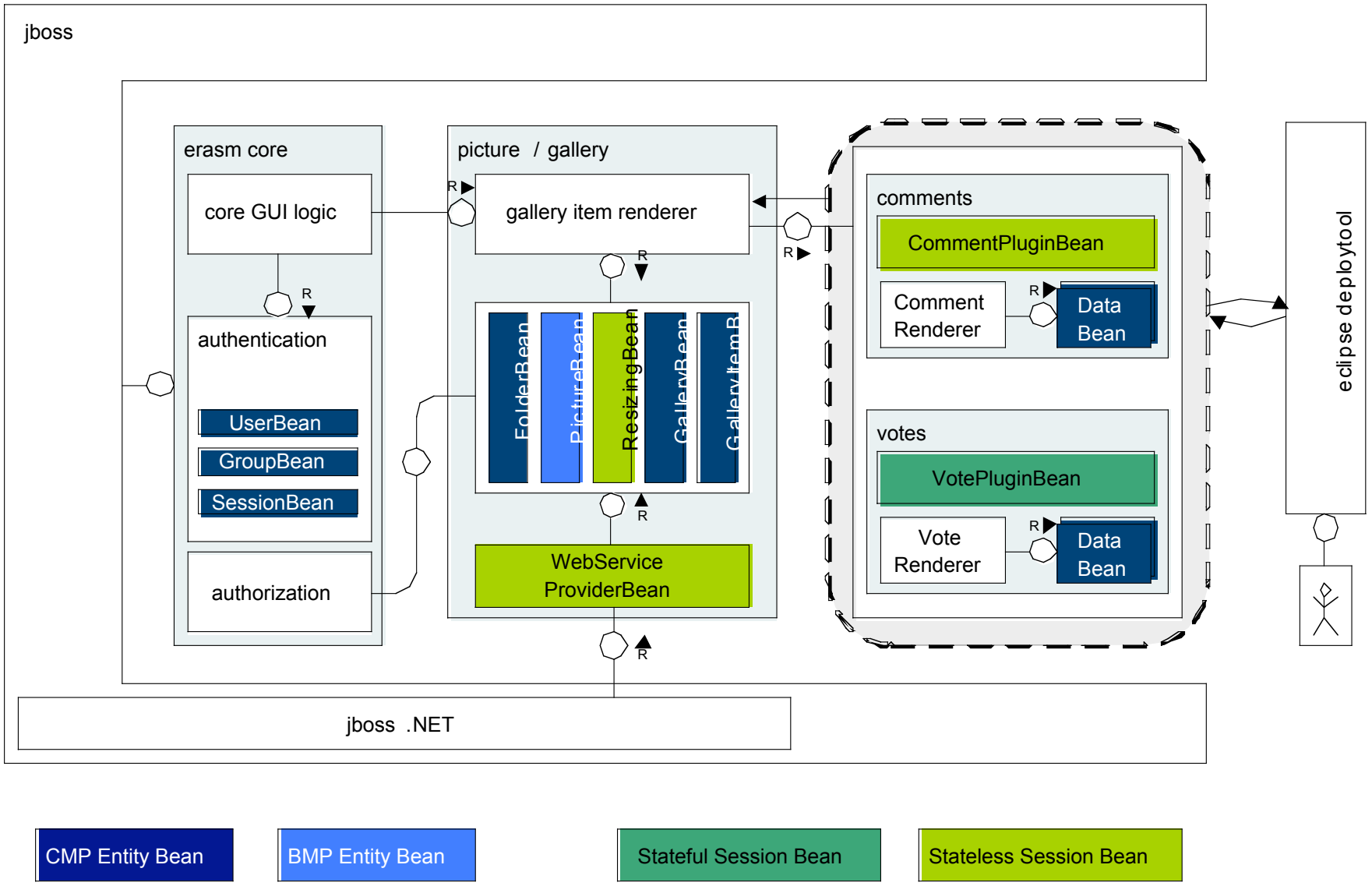
Projektplanung / -management

- **Entwurf Projektplan**
- **Versionsmanagement mittels Subversion Repository und TortoiseSVN**



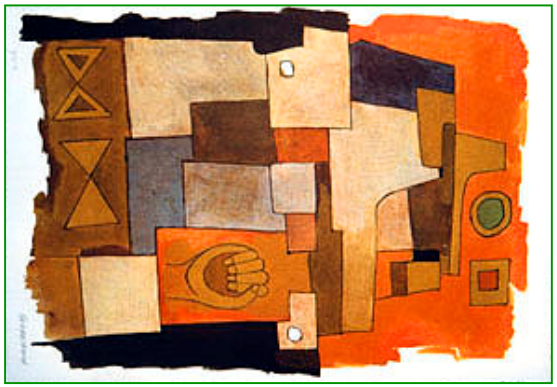
Microsoft Project Professional 2002

Architektur – Aufbau



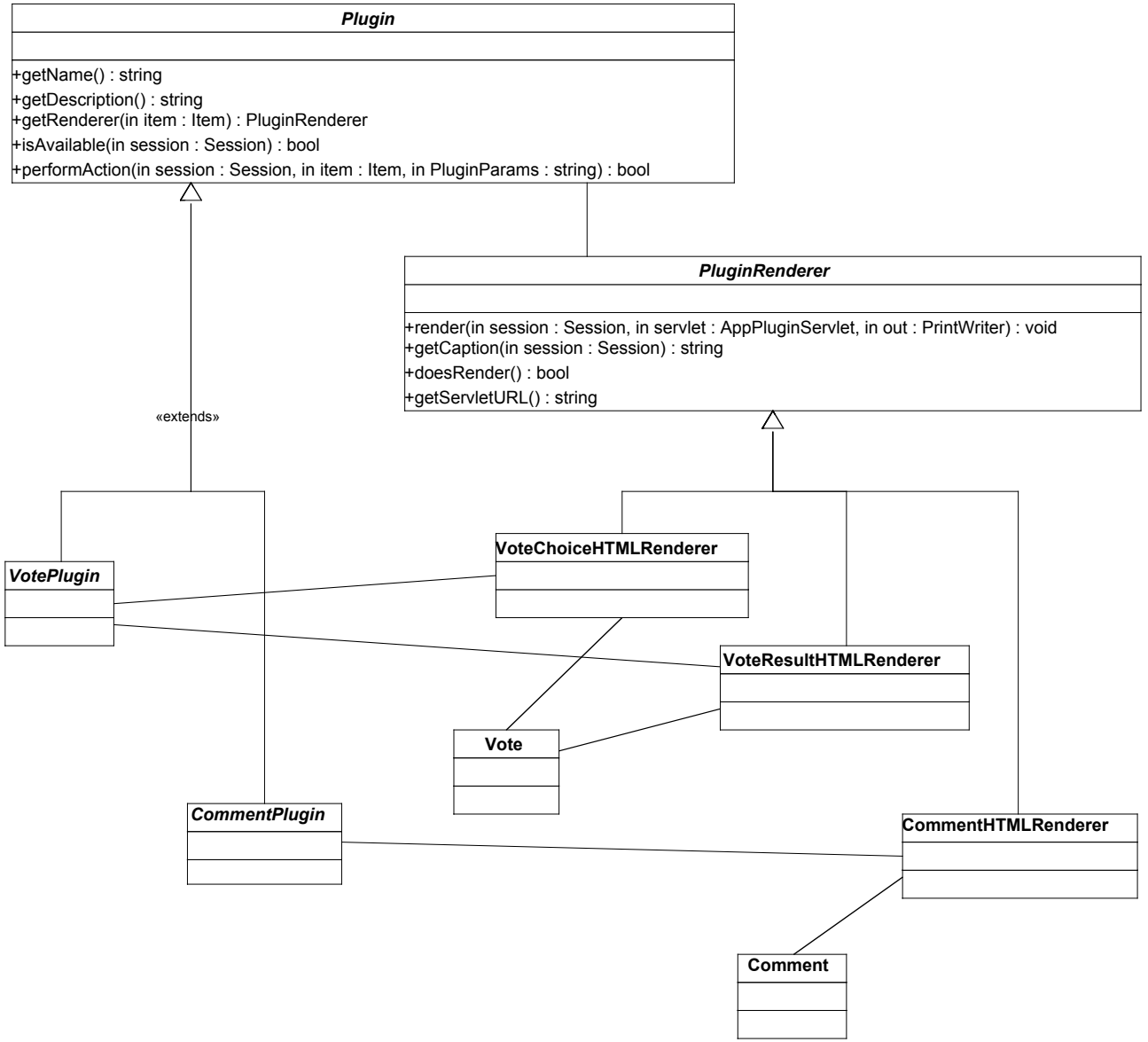
CMP Entity Bean | BMP Entity Bean | Stateful Session Bean | Stateless Session Bean

Architektur – Verwendete Design Patterns



- **Model – View – Controller**
  - Durchgängig durch Servlets / Renderer vs. EJBs
- **Singleton und Factory**
  - UniqueIDFactory, SessionManager und DBConnectionManager
- **Builder**
  - LoaderTask mit LoaderServlet als Director
- **Proxy**
  - Picture für PictureData
- **Façade**
  - WebServiceProvider

Architektur – Plugin Interface

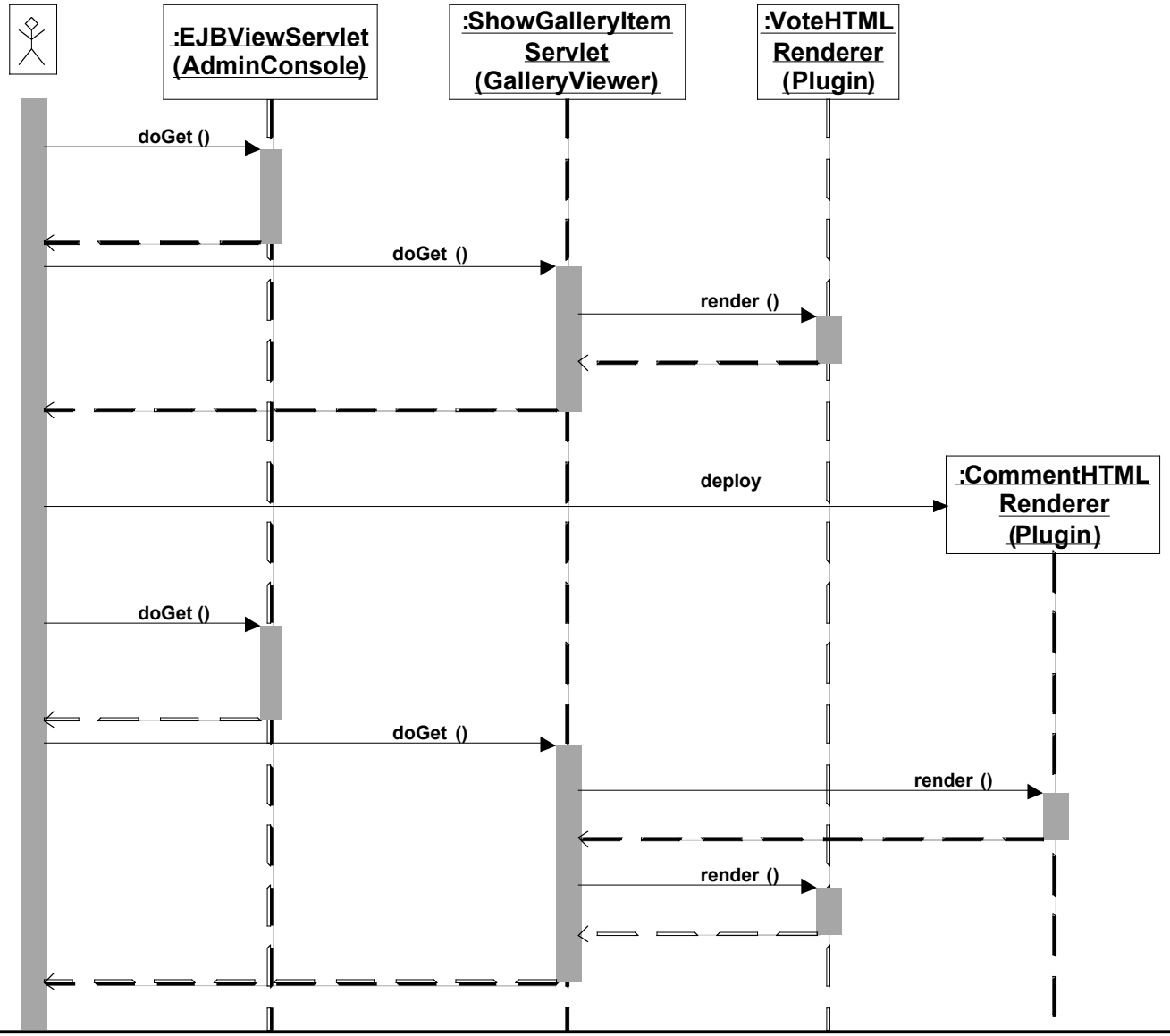


Suchen von Plugins

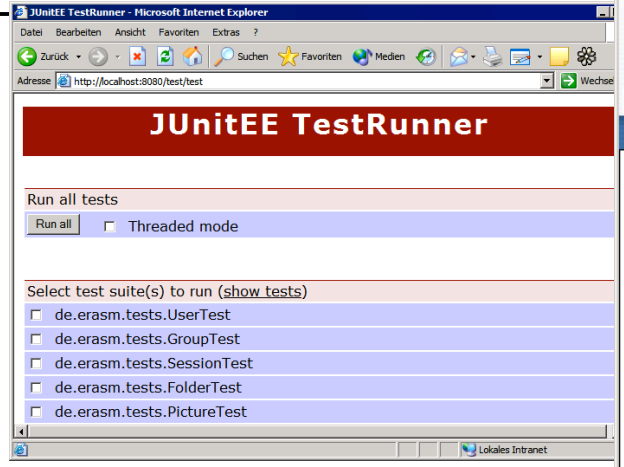
- **Suche aller verfügbaren Plugins erfolgt pro Anfrage**

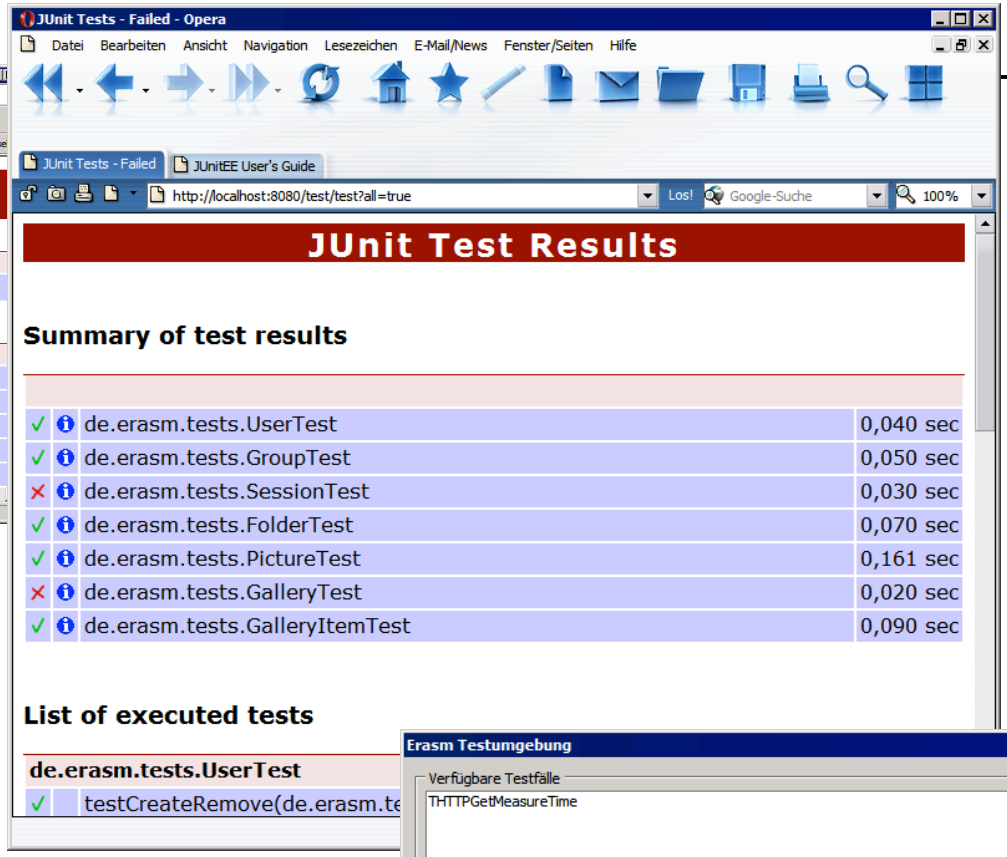
```
try {
    // get initial context
    Context context = new InitialContext();

    // list all objects bound for erasm/ejb/plugins/
    NamingEnumeration enum = context.list(PluginHome.JNDI_CONTEXT);

    // iterate the name class pairs
    while (enum.hasMore()) {
        NameClassPair ncp = (NameClassPair)enum.next();

        // look the object up
        Object obj = context.lookup(PluginHome.JNDI_CONTEXT+"/"+ncp.getName());

        // do we have a plugin home object?
        if (obj instanceof PluginHome) {
            // if so add a plugin instance to the list...
            plugins[pluginCount] = ((PluginHome)obj).create();
            pluginCount++;
            if (pluginCount == 10)
                return plugins;
        }
    }
} catch (Exception e) {
    throw new ServletException("Error finding the plugins ["+e.getClass().getName()+
```

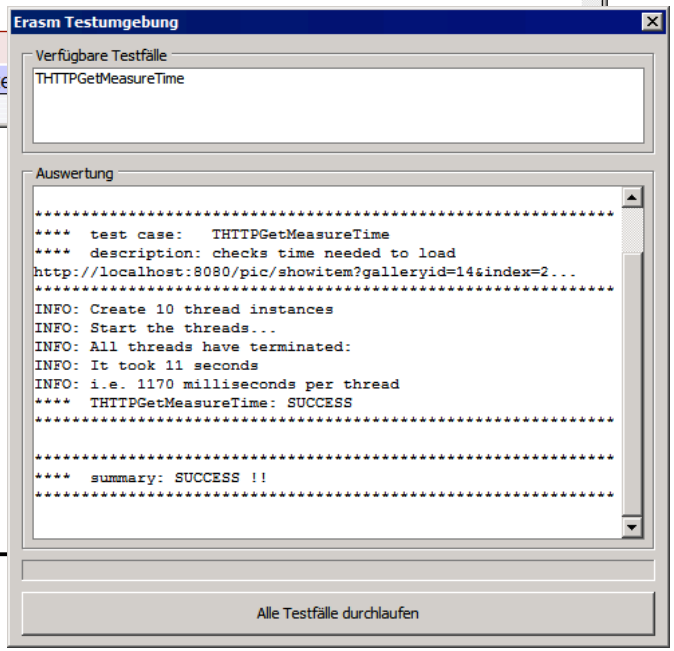
Vorführung Web GUI / Komponentendeployment



Test







- **JUnit als Framework für Testcases**
  - Nicht ausreichend für das Testen von J2EE-Applikationen
- **➔ JUnitEE als Testrunner**
- **Performanztests mit Delphi-Client**
  - WinInetAPI / HTTP

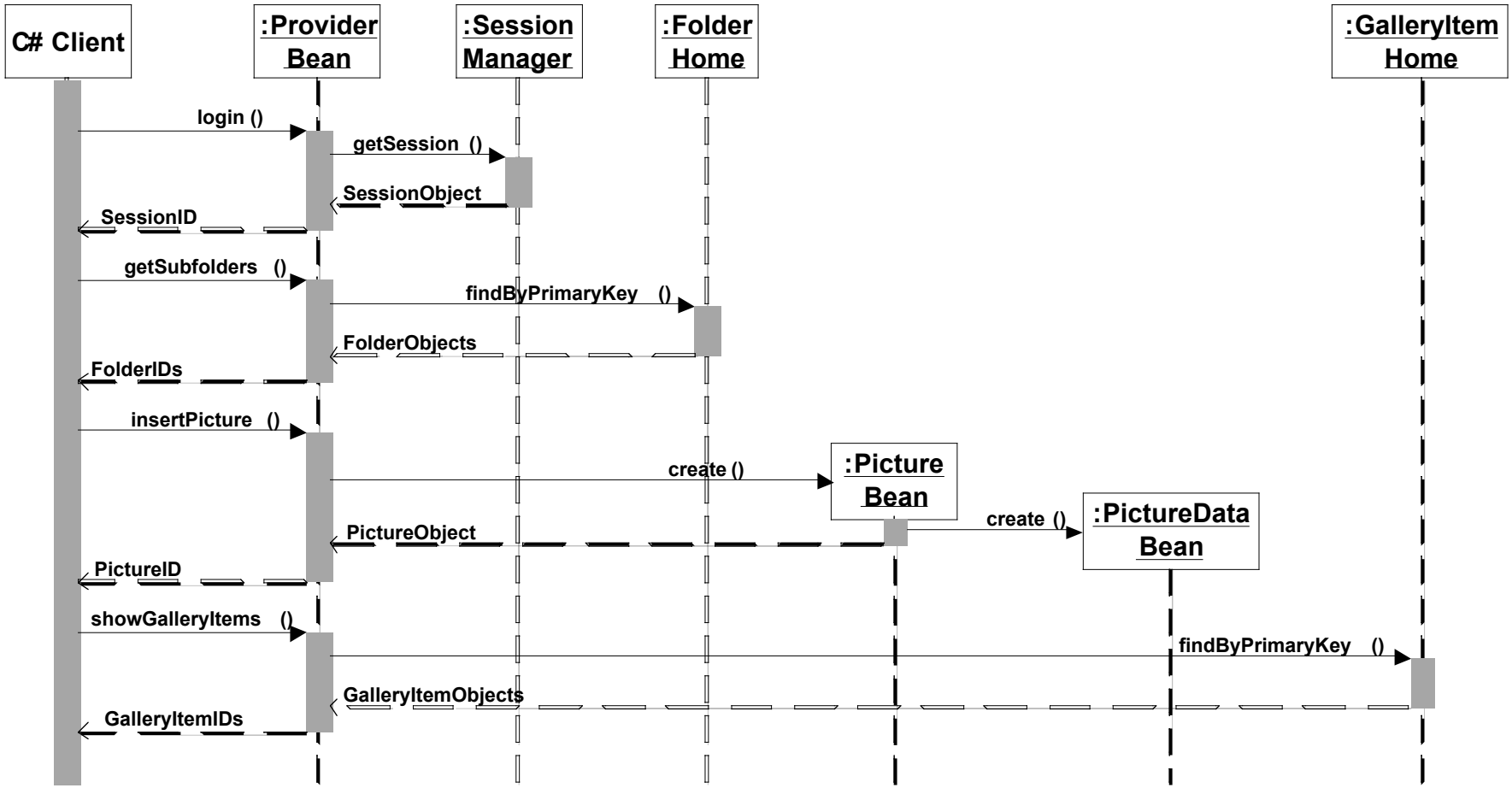
Bewertung J2EE / jboss

- **Nachteile**
  - Typprüfung für EJBs erst zur Deployzeit
  - Performanzprobleme
  - Keine Versionierung von EJBs

- **Vorteile**
  - J2EE: Nützliches Programmiermodell
  - Container-Managed-Persistence ist hilfreich
  - Gute Werkzeugunterstützung: Integration mit eclipse, XDoclet

Vorführung .NET Client

- Demo-Verlauf



Frameworkvergleich WebServices: jboss vs. .NET

Web Services mit JBoss

```
/**
 *@jboss-net web-service urn=„ProviderWS"
 */
public class ProviderBean implements
SessionBean {
```

```
/**
 *@ejb.interface-method view-type="local"
 *@jboss-net.web-method
 */
public long login(String userName,
                  String password) {
```

Web Services mit .NET

```
[System.Web.Services.WebServiceBindingAttribute(Name="ProviderWSSoapBinding",
Namespace="http://localhost:8080/jboss-net/services/ProviderWS")]
public abstract class ProviderLocalService : System.Web.Services.WebService {


[System.Web.Services.WebMethodAttribute()]
[System.Web.Services.Protocols.SoapRpcMethodAttribute("ProviderWS",
RequestNamespace="http://webservices.picture.erasm.de",
ResponseNamespace="http://localhost:8080/jboss-net/services/ProviderWS")]
[return: System.Xml.Serialization.SoapElementAttribute("loginReturn")]
public abstract long login(string in0, string in1);
```

Erweiterungen

- **Was man noch verbessern kann**
  - Bessere Testabdeckung (z.B. Zweigüberdeckung, GUI-Tests)
  - Erhöhung der Robustheit
  - Performanzoptimierung
  - Sicherheit
    - Verwendung von ACLs für jedes erasm-Item
  - Physische Verteilung des Applikationsserversystems
    - z.B. alle Bildoperationen auf separatem Rechner
  - GUI-Verbesserungen

- **Erweiterung der erasm-Platform**
  - Weitere Funktionen (Weblog, etc.)
  - Mehrsprachigkeit
  - Verwendung eines Portalservers (z.B. JetSpeed, Pluto)

Quellennachweise

- **J2EE**
  - Flanagan, D., Fareley, J., Crawford, W.: ***Java Enterprise in a Nutshell***, 2nd Edition, O'Reilly & Associates, 2002
  - JBoss Inc.: JBOSS 3.2 Getting Started, http://www.jboss.org/modules/html/docs/jbossj2ee.pdf
  - SUN Micros., J2EE 2.1 API Specification, http://java.sun.com/j2ee/1.4/docs/api/
  - SUN Micros., J2SE 1.4.2 API Specification, http://java.sun.com/j2se/1.4.2/docs/api/
  - SUN Micros., J2EE 1.4 Tutorial, http://java.sun.com/j2ee/1.4/docs/tutorial/doc/
  - XDoclet Dokumentation, http://xdoclet.sourceforge.net/xdoclet/development/
- **Datenbankanbindung**
  - SAP MaxDB Dokumentation, http://www.sapdb.org/7.4/sap_db_documentation.htm
- **.NET**
  - Liberty, J.: ***Learning C#***, O'Reilly & Associates, 1. November 2002
  - Liberty, J.: ***Programming C#***, O'Reilly & Associates, 1. Juni 2003
- **WebServices**
  - JBoss .NET Info, http://www.nsdev.org/jboss/stories/jboss-net.html
  - Microsoft Developer Network, http://msdn.microsoft.com/webservices/understanding/specs/

- **Alle URLs beziehen sich auf Juni 2004**