Raytracer
in C# und .NET

Kolja Schötzau, Daniel Gülzow, Thomas Adam, Steffen Heinrich, Stephan Brumme

Gliederung

1. RayTracing
2. Anforderungsspezifikation
3. Architektur
4. Web Services
5. Testen
6. Wahl der Plattform
7. Ausblick

1.

RayTracing

1. RayTracing: Beispiele

1. RayTracing: Beispiele

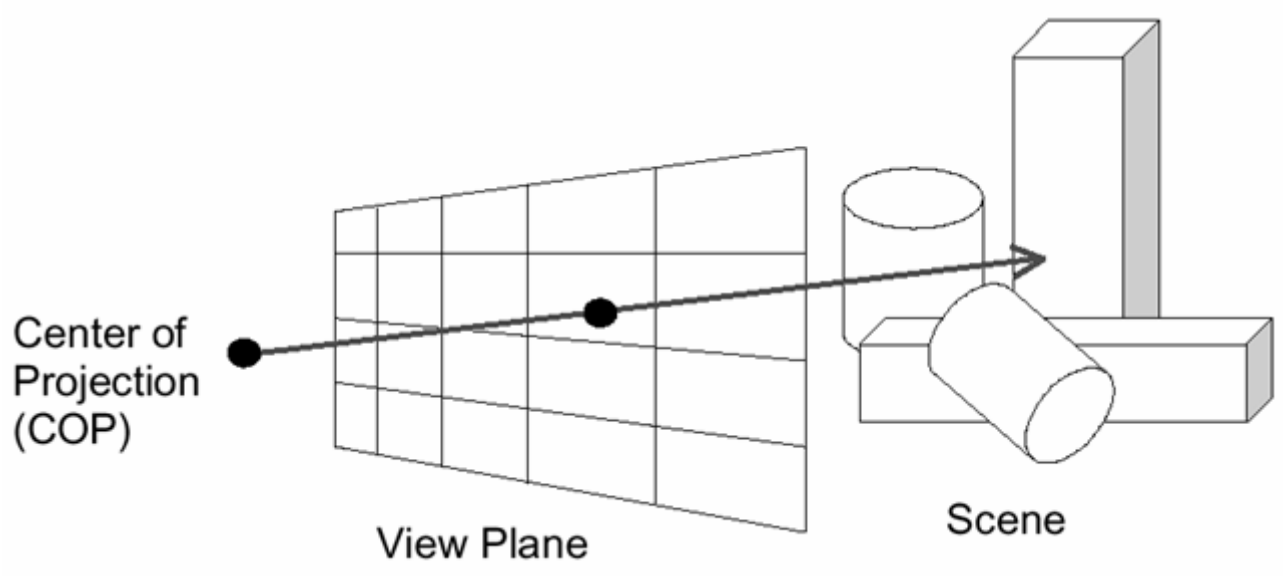
1. RayTracing: Beispiele

1. RayTracing: Beispiele

1. RayTracing: Prinzipien

Prinzipelle Arbeitsweise:

- Simulation der Lichtausbreitung: Verfolgung unendlich dünner, gerader Strahlen
- Approximation physikalisch–basierter Beleuchtung

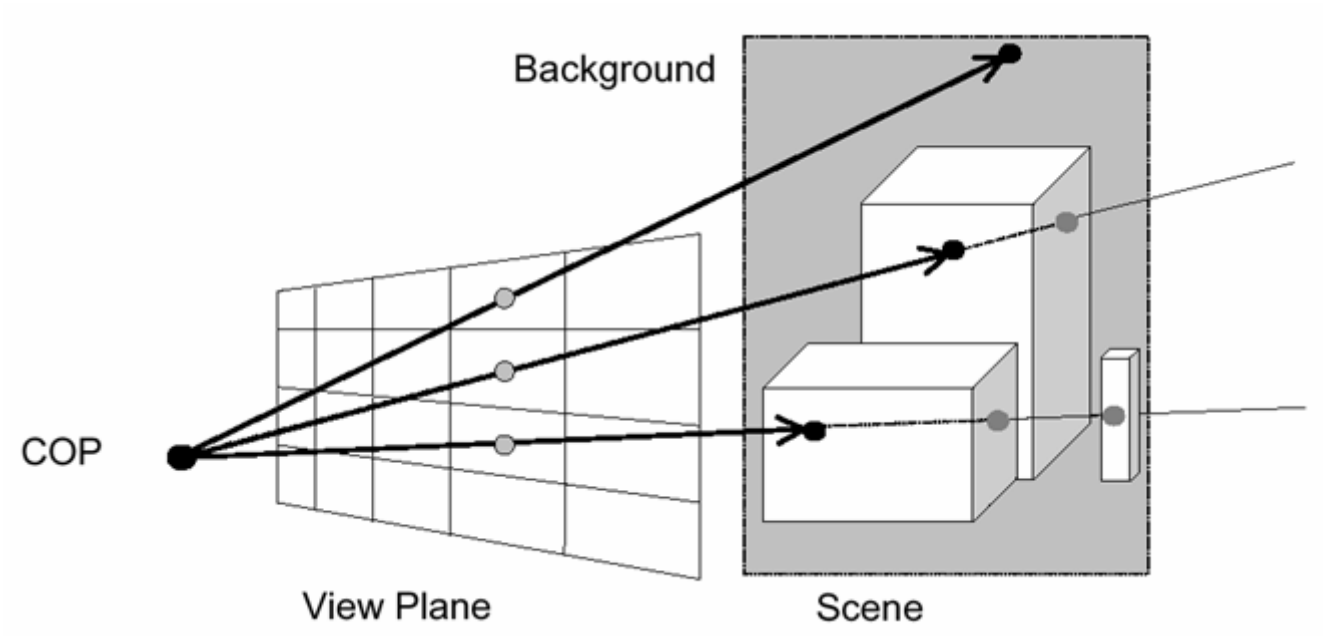


Entsenden von Strahlen vom Kameraprojektionszentrum durch die Sichtebene in die Szene

1. RayTracing: Prinzipien

Strahlenverfolgung:

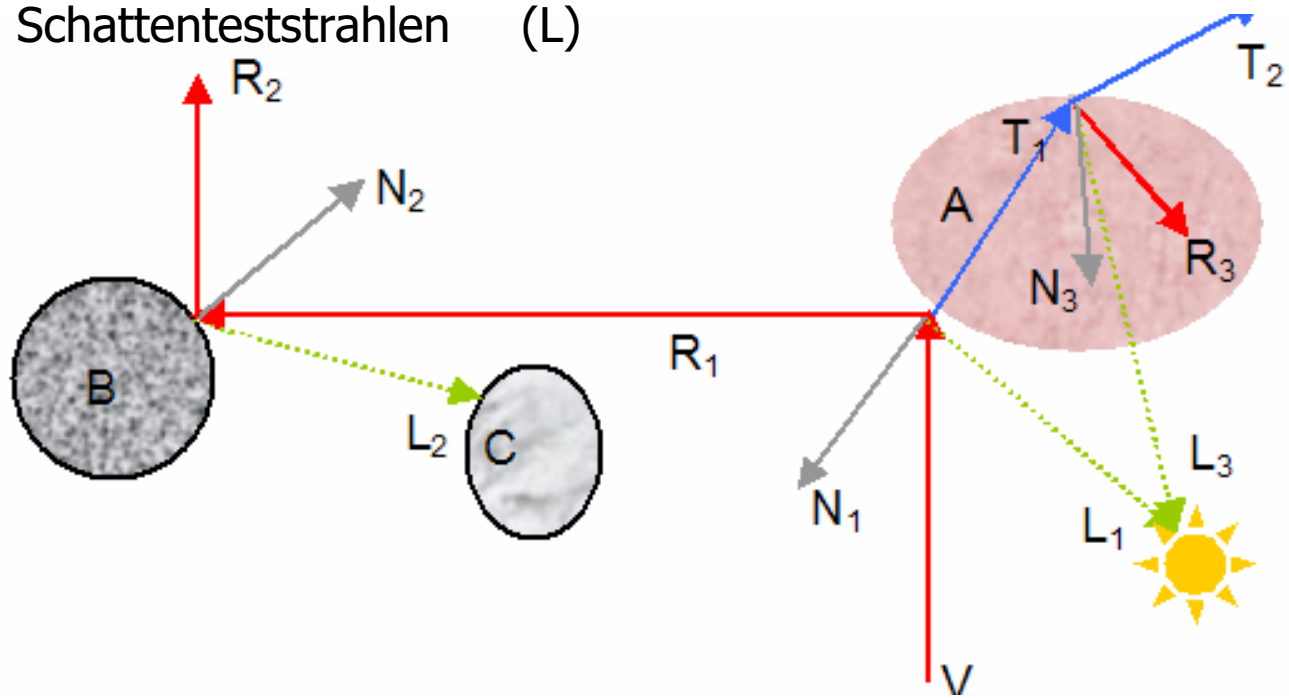
- Intersektionsberechnung zwischen Strahl und Szenenobjekt
- Intensitätsberechnung am Schnittpunkt
  - Direkte Beleuchtung durch Lichtquellen
  - Indirekte Beleuchtung durch reflektierte und transmittierte Strahlen



1. RayTracing: Prinzipien

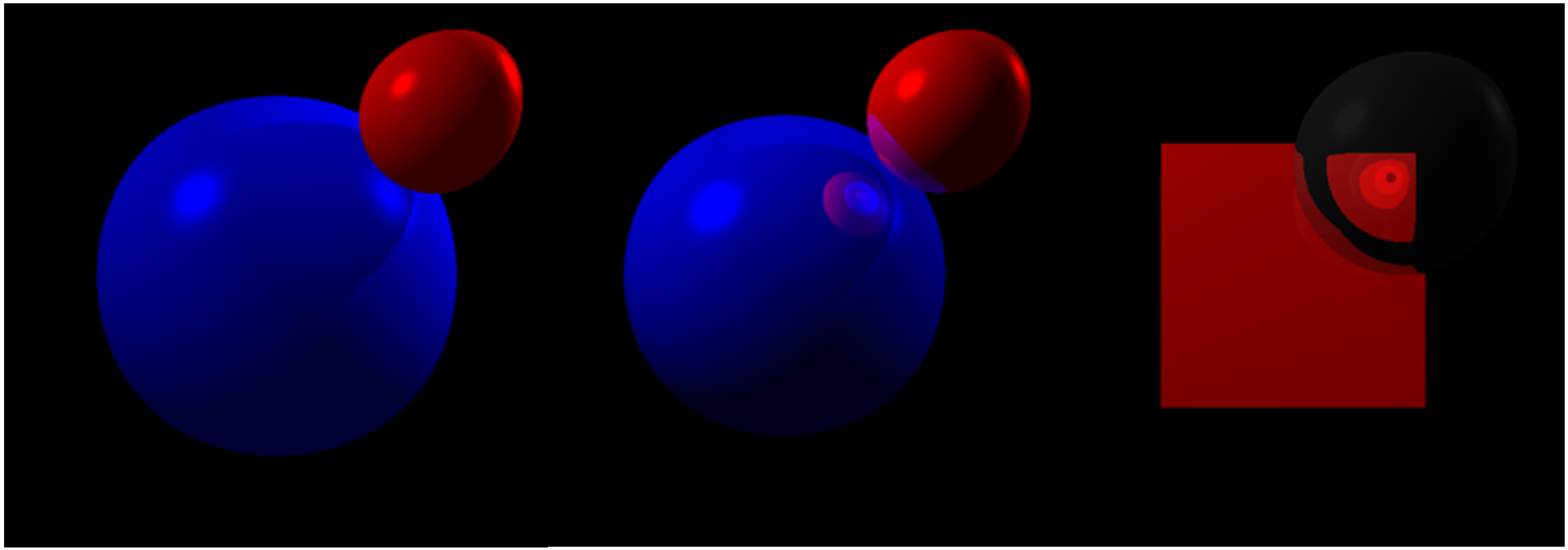
Rekursion: Strahlen n–ter Ordnung

- Berücksichtung von Schatten, Reflexion (Spiegelbilder) und Transmission (Lichtbrechung)
- Primärstrahl erzeugt mehrere Sekundärstrahlen
  - Reflexionsstrahlen (R)
  - Brechungsstrahlen (T)
  - Schattenteststrahlen (L)



1. RayTracing: Prinzipien

Rekursion: Beispiele

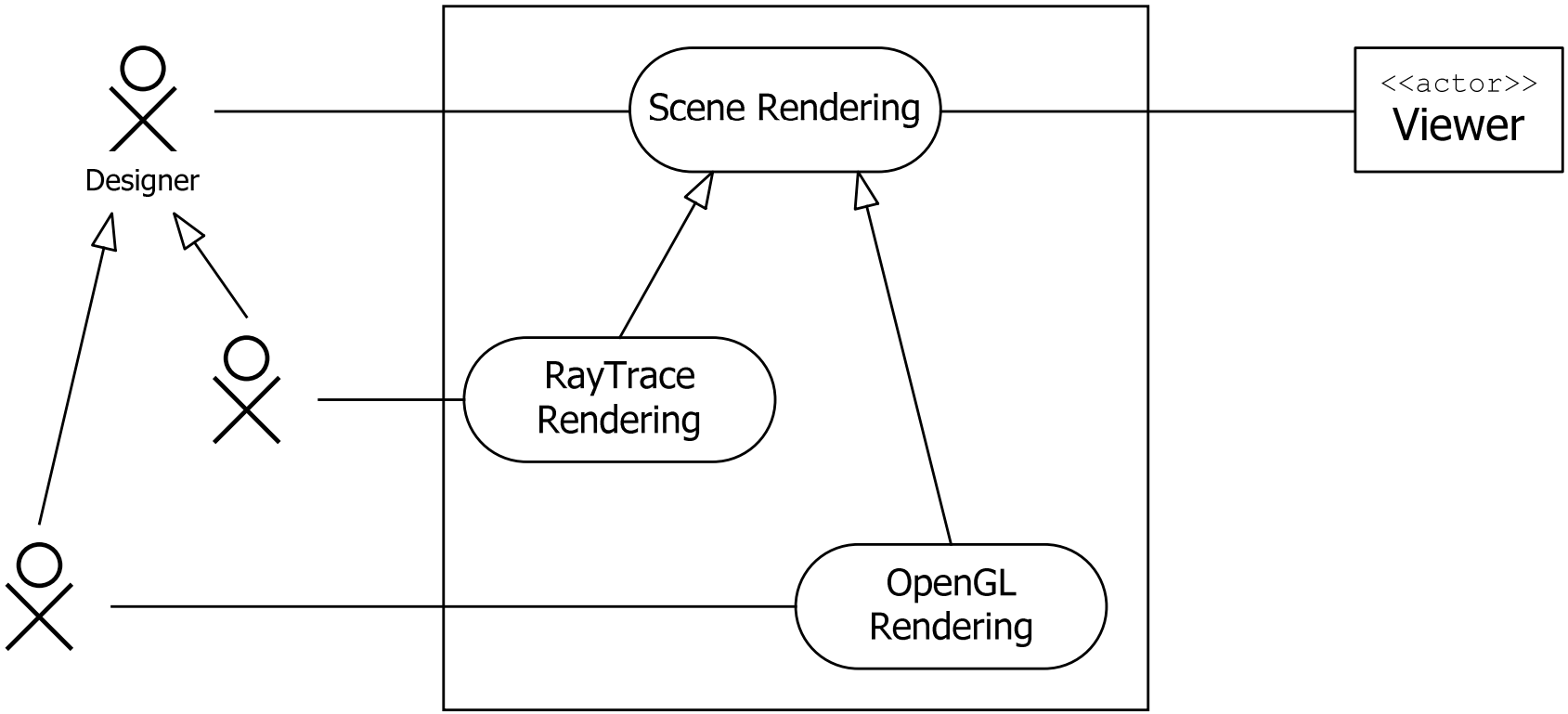


Schatten | Reflexion | Transmission & Brechung

Anforderungsspezifikation

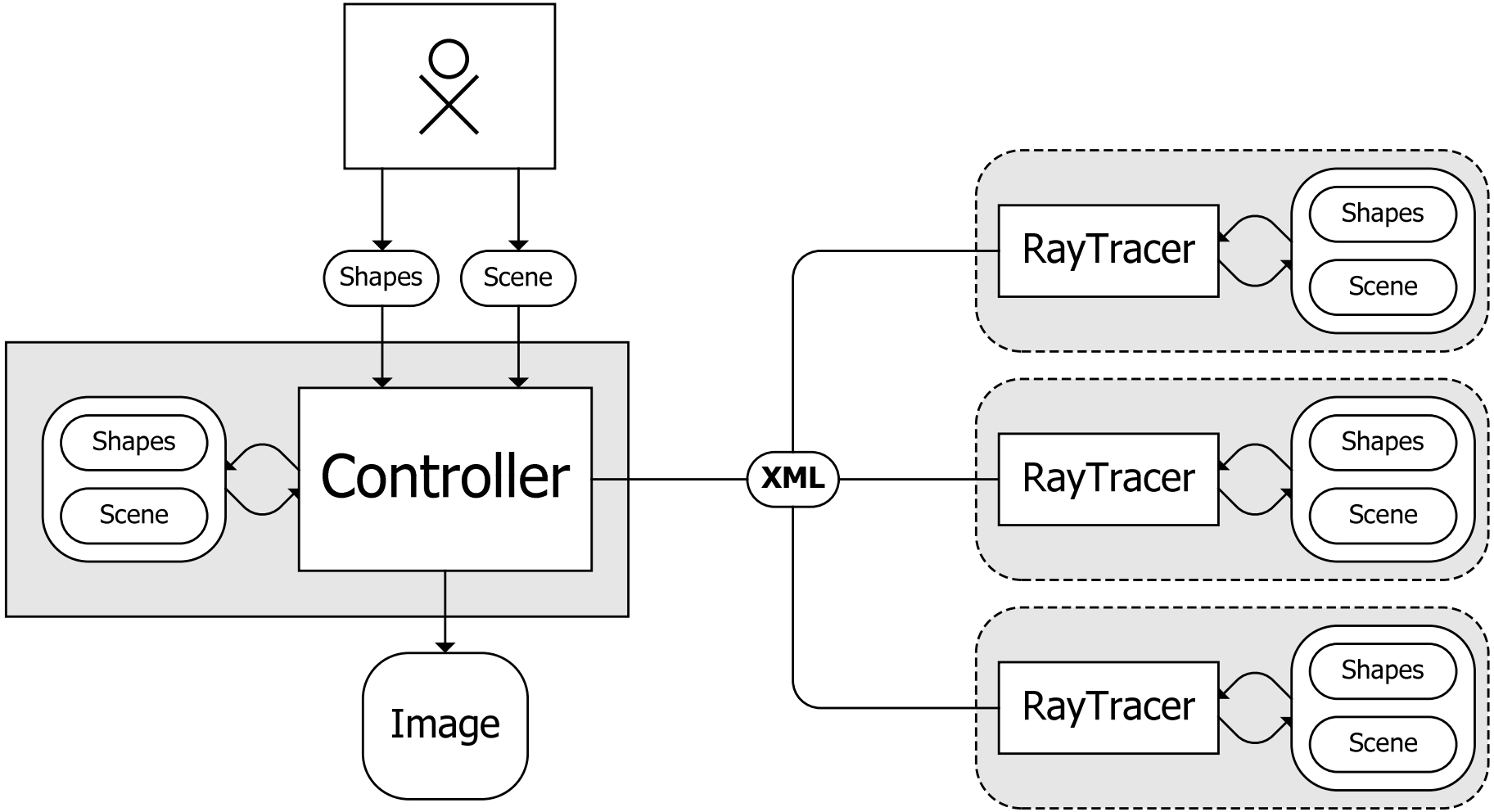
2. Anforderungsspezifikation

Anwendungsfälle:



2. Anforderungsspezifikation: RayTracing

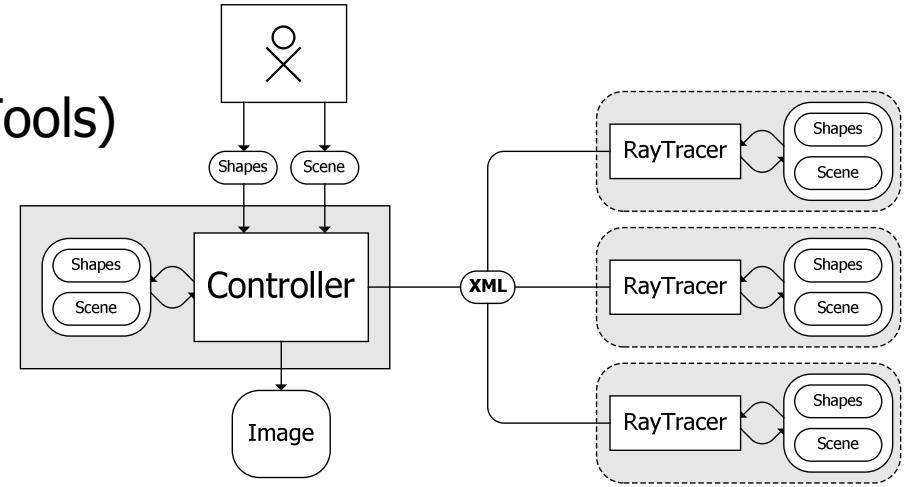
Verteilte Berechnung der Szene



2. Anforderungsspezifikation: RayTracing

Benutzer

- Beschreibt die Szene (Einsatz von Tools)
- Stellt notwendige Spezifikation der Shapes zur Verfügung



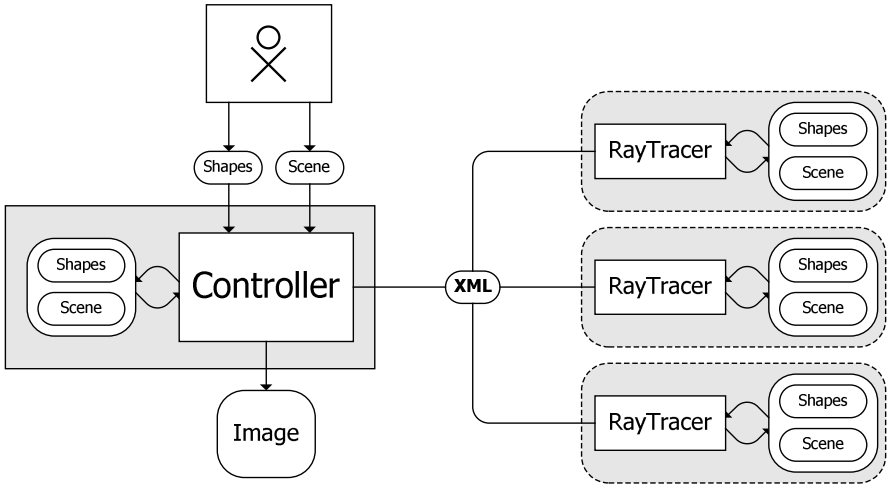
Controller

- Kapselt Szene & Shapes in Szenenbeschreibung; (De-) Serialisierung
- Leitet Szenenbeschreibung an Worker weiter
- Setzt Ergebnisse der Worker zum fertigen Bild zusammen

2. Anforderungsspezifikation: RayTracing

Worker (RayTracer)

- Implementiert als WebService
- Empfängt Szenenbeschreibung
- Deserialisiert sie; dynamisches Erzeugen geeigneter Shapes (spezialisiert auf RayTracing)
- Berechnet zugewiesenen Bereich der Szene; Ergebnis sind Farbwerte

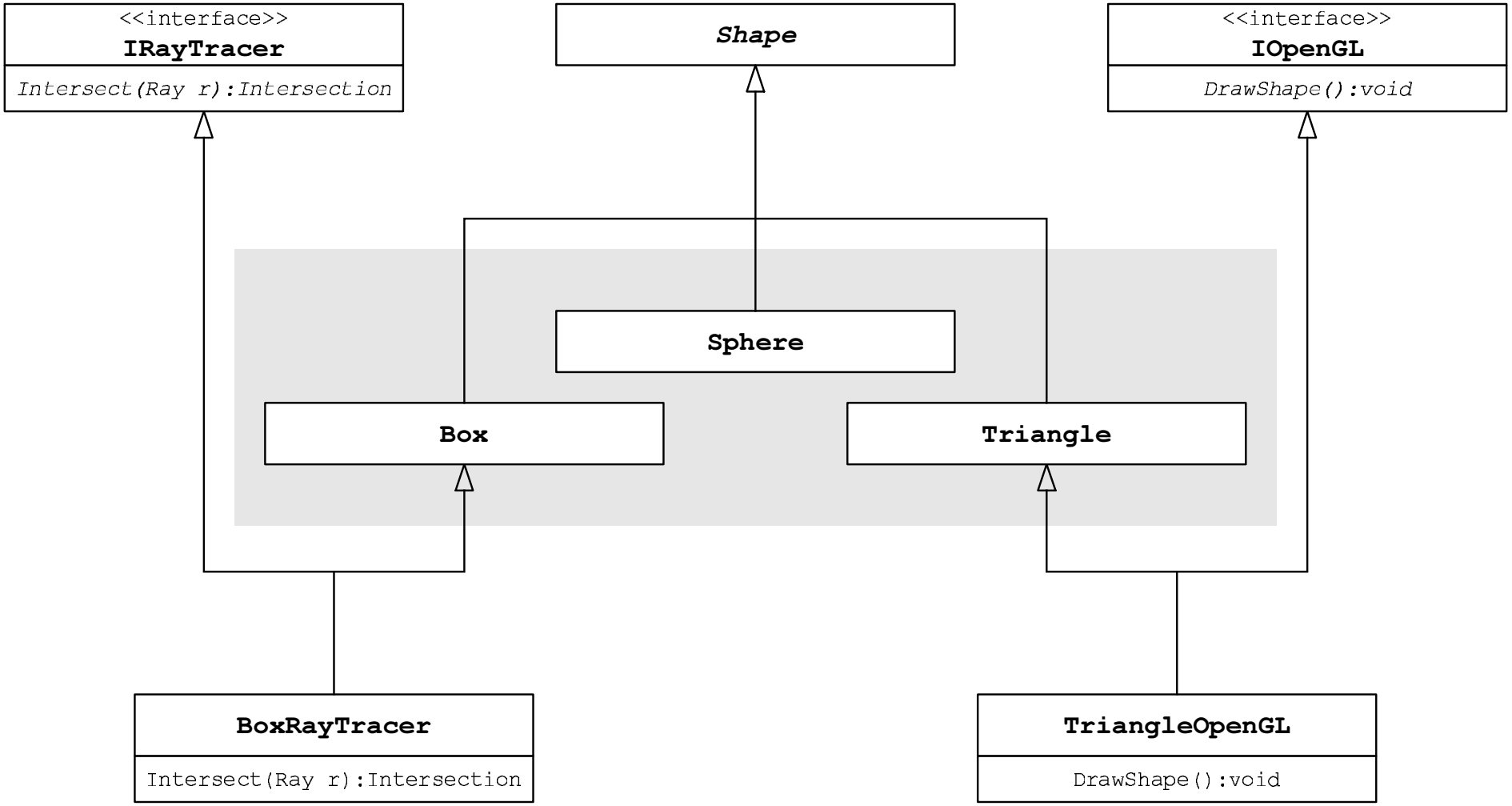


Demonstration

3.

Architektur

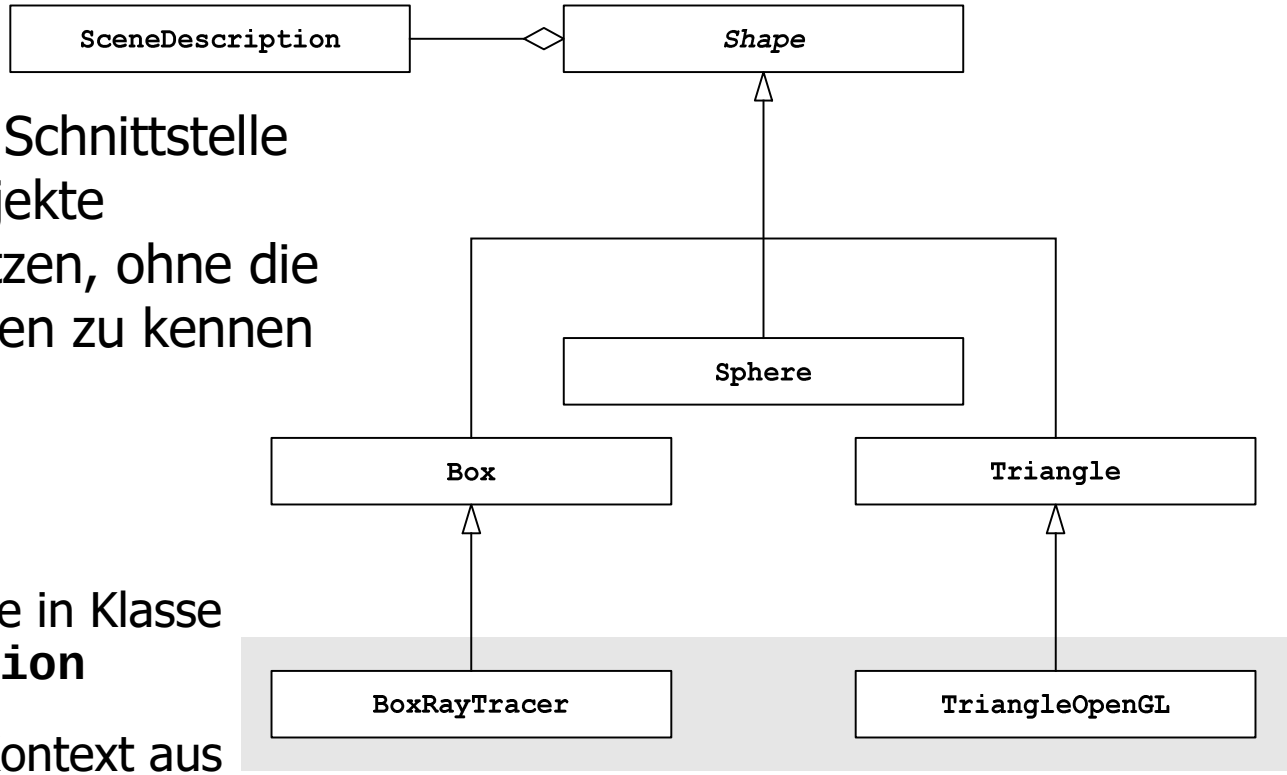
3. Architektur: Klassendiagramm



3. Architektur: Design Patterns

Abstract Factory

- Zugriff auf eine Schnittstelle um über sie Objekte zusammenzusetzen, ohne die konkreten Klassen zu kennen



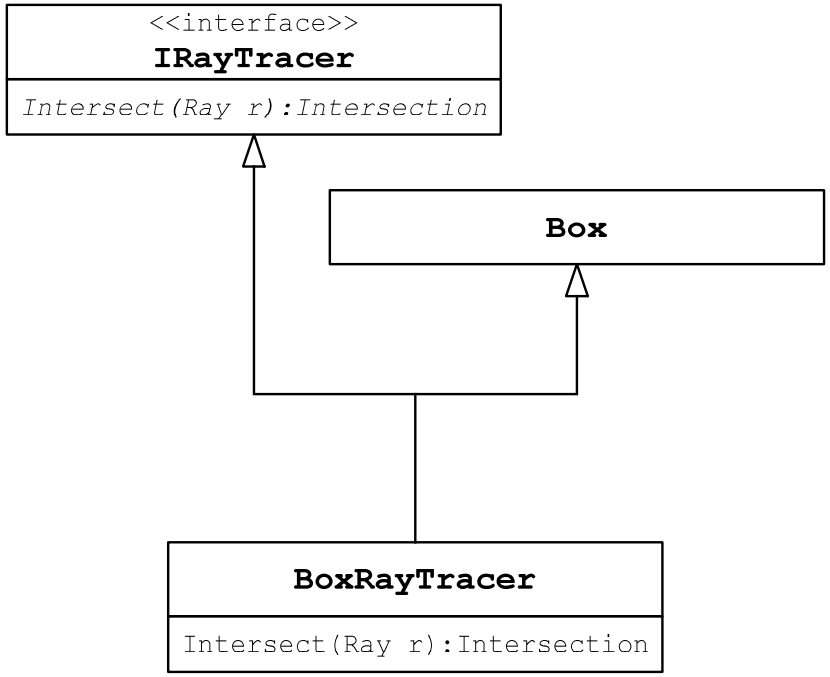
Beispiel

- Casting – Methode in Klasse **`SceneDescription`**
- Erzeugt je nach Kontext aus Szenenbeschreibung die korrekten Shapes; also RayTracer– oder OpenGL–Objekte

3. Architektur: Design Patterns

Decorator

- Ergänzung der Funktionalität eines Objekts; "Umwickeln" des Objekts durch Anbieten einer erweiterten Schnittstelle



Beispiel

- Interfaces **`IRayTracer`** sowie **`IOpenGL`** werden zu Shapes hinzugefügt
- Erweiterung der geometrischen Objekte um Darstellungsfunktionalität
  `Intersect(Ray r);`
  `DrawShape();`

3. Architektur: Design Patterns

Interpreter

- Repräsentation einer Grammatik; Objekt, das Repräsentation nutzt, um Sätze darin zu interpretieren

Beispiel

- Szenenbeschreibung in einer XML – Datei kodiert
- Im .NET – Framework durch Serialisierung / Deserialisierung gegeben

```
<?xml version="1.0" ?>
- <SceneDescription xmlns:xsd= ... >
- <m_Camera>
  <From X="0" Y="0" Z="2" />
  <To X="0" Y="0" Z="0" />
  <Up X="0" Y="1" Z="0" />
  <View X="0" Y="0" Z="2" />
  <FieldOfViewY>60</FieldOfViewY>
  </m_Camera>
- <m_Light>
- <Light>
  <Position X="1" Y="1" Z="1" />
  <Intensity>1</Intensity>
  <Ambient>0.5</Ambient>
  </Light>
  </m_Light>
- <m_Shape>
- <Shape xsi:type="Triangle">
- <Vertices>
  <Vector X="-1" Y="-1" Z="-1" />
  <Vector X="1" Y="-1" Z="-1" />
  <Vector X="1" Y="1" Z="0" />
  </Vertices>
  <Normal X="0" Y="-0.44721359549995793"
    Z="0.89442719099991586" />
  </Shape>
  </m_Shape>
...
```

3. Architektur: Design Patterns

Proxy

- Zugriffskontrolle auf ein Objekt mit Hilfe eines vorgelagerten Stellvertreterobjekts

Beispiel

- Vom .NET – Framework bei Verwendung von WebServices automatisch erzeugt

3. Architektur: Design Patterns

MVC – Paradigma

- Trennung der Zuständigkeitsbereiche
  - Model (Datenhaltung)
  - View (Darstellung)
  - Controller (Steuerung)

Beispiel

– Möglichkeit der Einflussnahme, wie die Szene dargestellt wird
  Bild vs. OpenGL–Fenster

Web Services

4. Web Services

Was sind Web Services

- Web Services ermöglichen den Aufruf von Applikationslogik mit Hilfe von Standard-Protokollen (HTTP, SMTP, SOAP) über Rechnergrenzen hinweg
- Vergleichbar mit CORBA, aber universeller

In welchem Umfeld sind Web Services einsetzbar?

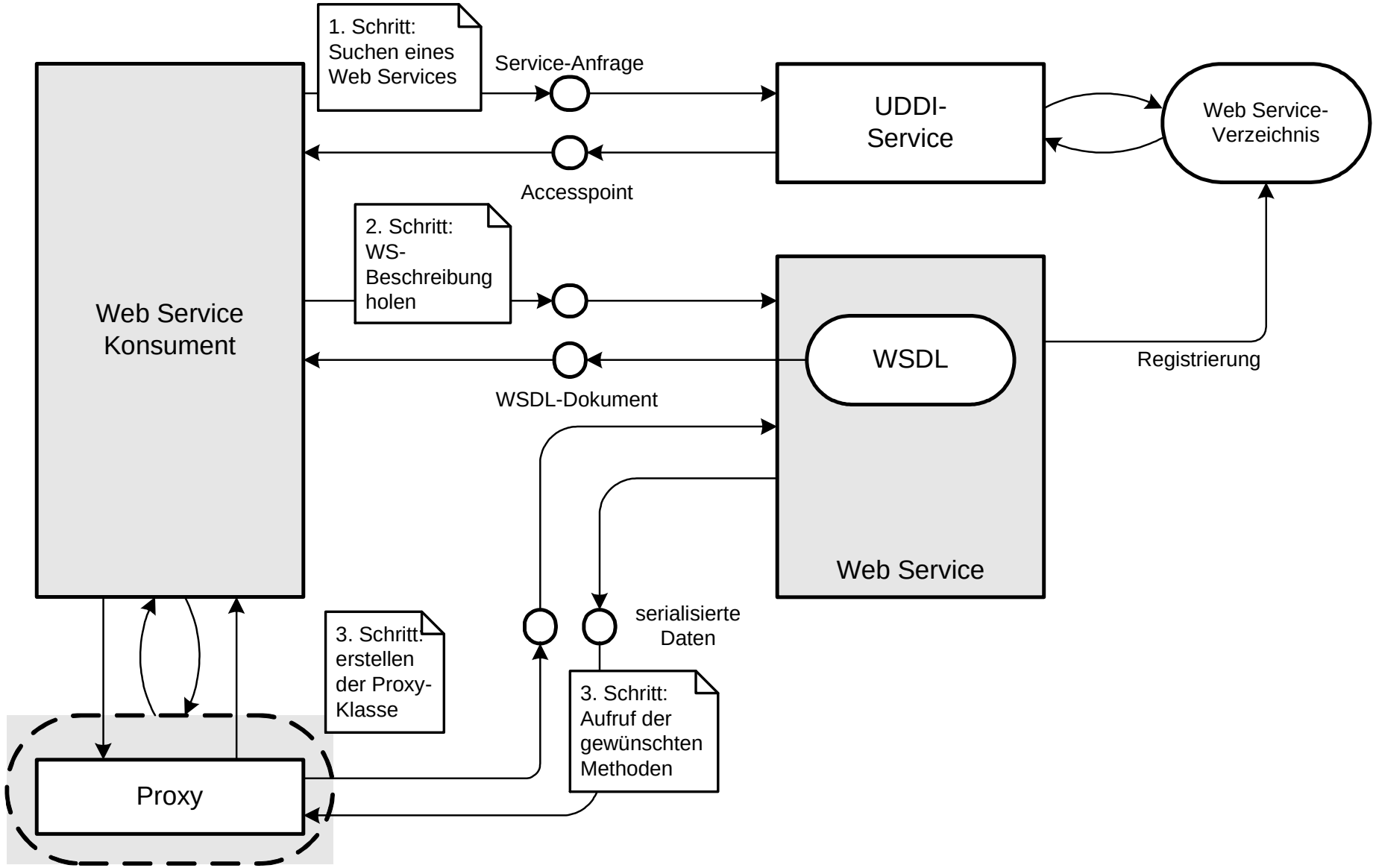
- Web Services können auf jedem per HTTP erreichbaren Rechner implementiert sein und können von den verschiedensten Endgeräten konsumiert werden (PDA‘s, Handys, PC‘s usw.)

4. Web Services - Technologien

Wie funktionieren Web Services : Technologien

- UDDI
- WSDL
- Protokoll (SOAP)
- XML / Serialisierung

4. Web Services - Technologien



4. Web Services - UDDI

UDDI – Universal Description, Discovery and Integration

- UDDI ist ein Verzeichnisdienst, der Informationen über verfügbare Webservices bereithält – Gelbe Seiten
- Enthält Informationen über ein Unternehmen, welche Web Services angeboten werden und welche WSDL-Dokumente zu ihrer Beschreibung benutzt werden
- Um Services besser auffinden zu können, werden Informationen nach Kategorien sortiert

4. Web Services - UDDI

- UDDI selbst wird als Webservice implementiert
- verschiedene Implementationen entsprechend der Spezifikation von www.UDDI.org verfügbar (Microsoft, IBM, HP usw.)
- Registrierung und Suche nach Web Services können sowohl über Webinterfaces oder programmatisch über eine API erfolgen
- UDDI API auch für .NET erhältlich

Interessant:

- Interface-Beschreibungen werden als Technology Model gespeichert (tModel)
- Durch sog. Bindings wird Zuordnung zu Web Services getroffen, welche das Interface implementieren
- so können verschiedene Web Services ein Interface implementieren

4. Web Services - WSDL

Web Service Description Language

- WSDL ist ein XML-Format um Web Services zu beschreiben
- Diese Beschreibung beinhaltet Details des Webservices:
  - wo der Service gefunden werden kann
  - welche Methoden unterstützt werden
  - die verwendeten Datentypen
  - Die unterstützten Protokolle
- Ein WSDL-Dokument genügt, um die Proxy-Klasse zu erstellen.


```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://tempuri.org/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encod
ing/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/
" xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://tempuri.org/"
xmlns="http://schemas.xmlsoap.org/wsdl/">- <types>-
<s:schema elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">- <s:element
name="setScene">- <s:complexType>-
<s:sequence>  <s:element minOccurs="0" maxOccurs="1"
name="strXmlScene" type="s:string"
/>  </s:sequence>  </s:complexType>  </s:element>-
<s:element
name="setSceneResponse">  <s:complexType
/>  </s:element>- <s:element name="render">-
<s:complexType>- <s:sequence>  <s:element
minOccurs="1" maxOccurs="1" name="from" type="s:int"
/>  <s:element minOccurs="1" maxOccurs="1" name="to"
type="s:int"
/>  </s:sequence>  </s:complexType>  </s:element>-
<s:element name="renderResponse">- <s:complexType>-
<s:sequence>  <s:element minOccurs="0" maxOccurs="1"
name="renderResult" type="s0:ArrayOfAnyType"
/>  </s:sequence>  </s:complexType>  </s:element>
```

4. Web Services – Proxyklasse

Die Proxy-Klassen

- Proxy-Klasse ist der Stellvertreter des Web Services auf Client-Seite
- Entspricht den Stubs bei CORBA
- Übernimmt Serialisierung und Deserialisierung der Methodenaufrufe und Antworten
- WSDL-Dokument enthält alle nötigen Informationen, um Proxy-Klasse zu generieren (WSDL.exe)

4. Web Services – Protokolle

ASP.NET unterstützt drei Protokolle:

- HTTP-GET
- HTTP-POST
- SOAP

HTTP-GET und HTTP-POST

- Die Unterstützung dient der Rückwärtskompatibilität nach ASP 3.0
- Über diese Protokolle (bzw. HTTP-Methoden) können nur einfache Datentypen übergeben werden (String, Int usw.)
- so können ältere ASP-Applikationen ebenfalls Web Services (eingeschränkt) konsumieren
- Einsatz z.B. auf `DefaultWsdlHelpPage.asmx` (dazu später mehr)

4. Web Services – SOAP

SOAP – Simple Object Access Protocol

- Protokoll zum Austausch von strukturierter und typisierter Information über das Web
- XML-basierend
- HTTP einziges für SOAP-Nachrichten standardisiertes Transportprotokoll. Aber: andere Protokollbindungen möglich
- Durch den SOAP-Standard definiert:
  - Aufbau von SOAP-Nachrichten (Envelope, opt. Header, Body)
  - Mechanismen zum (De-) Serialisieren von abstrakten Daten
  - RPC-Konventionen
  - Bindung an HTTP
- durch Benutzung von HTTP: Request schicken -> Response bekommen

4. Web Services – Protokolle

Eine Beispielnachricht, eingebettet in einen HTTP-Request:

```
POST /Aktien HTTP/1.1
Host: www.aktien.de
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "MeineURI„

 <SOAP-ENV:Envelope xmlns:
       SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
                 <SOAP-ENV:Body>
                                 <M:HoleLetztenKurs xmlns:M="MeineURI">
                                                 <Aktie>SUN</Aktie>
                                 </M:HoleLetztenKurs>
                 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

...und die darauf erfolgte Antwort:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn

<SOAP-ENV:Envelope xmlns:
       SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
       SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/envelope/">
                 <SOAP-ENV:Body>
                                 <M:LetzterKurs xmlns:M="MeineURI">
                                 <Kurs>107.0</Kurs> </M:LetzterKurs>
                 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

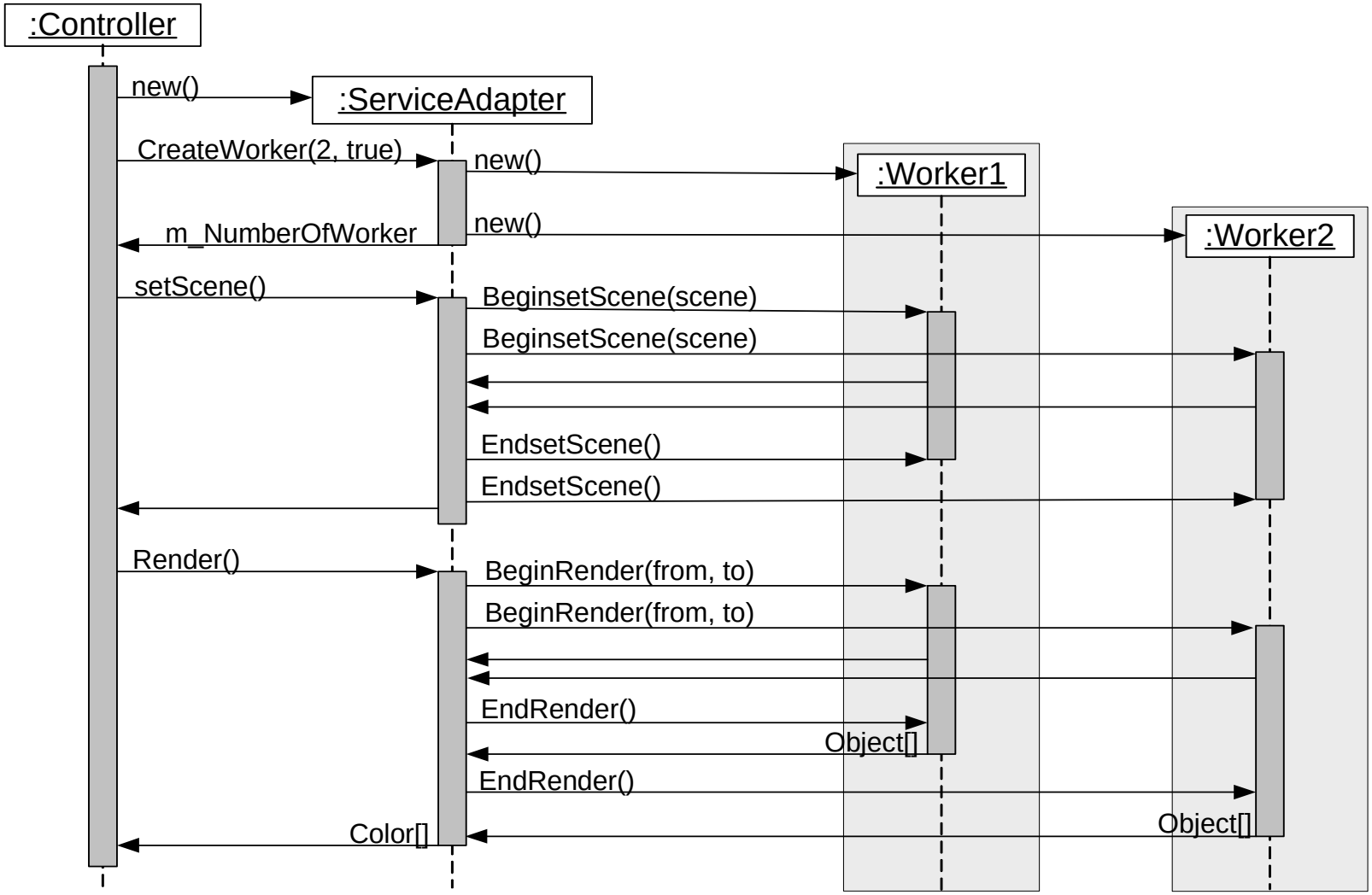
4. Web Services – SOAP

RPCs mit SOAP

- RPC‘s werden in SOAP-Nachrichten als XML gekapselt
- dadurch wird ein einheitlicher, systemunabhängiger Standard für Remote Procedure Calls definiert
- -> darum auf Client- und Serverseite
  - keine Benutzung eines identischen Protokolls (DCOM, CORBA)
  - Benutzung der gleichen Programmiersprache nötig (wie z.B. Remote Method Invocation in Java)

4. Web Services – Einsatz im Raytracer

Sequenzdiagramm: Aufruf von Web Methoden



4. Web Services – SOAP

Serialisierung von Daten in .NET

- Binäre Darstellung, SOAP oder XML möglich
- vom .NET-Framework direkt unterstützt:
  - Binär: **`BinaryFormatter`**
  - SOAP: **`SOAPFormatter`**
  - XML: **`XMLSerializer`**
- von uns genutzt: SOAP für Kommunikation mit Web Services, XML zur persistenten Speicherung

5.

Testen

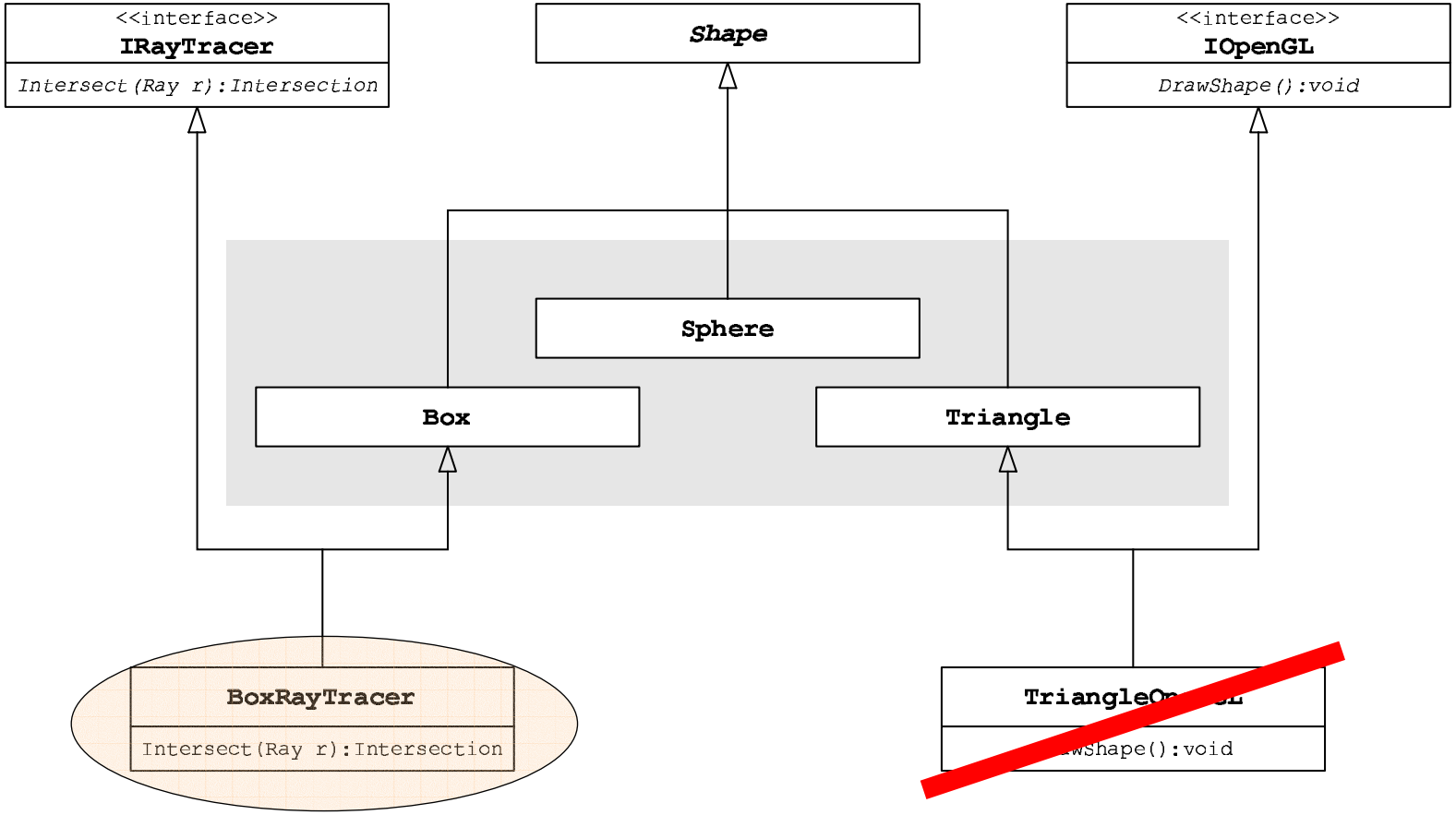
5. Testen

- CodeReview
- Klassentest
- Integrationstest

5. Testen: Klassentest – Was?

Alle Shapes, aber Verzicht auf...

- Testen der Funktionen von OpenGL, da als vorgefertigte Bibliothek vom Hersteller hinreichend getestet



5. Testen: Klassentest – Wie?

Ein Weg:

- Ergebnis– und Kontrollanzeigen über Standard–Ausgabe
- Führen einer Log – Datei

Ein anderer Weg: Debugger

- Analyse und Manipulation einzelner Ausdrücke, ohne neu kompilieren zu müssen

Aber:

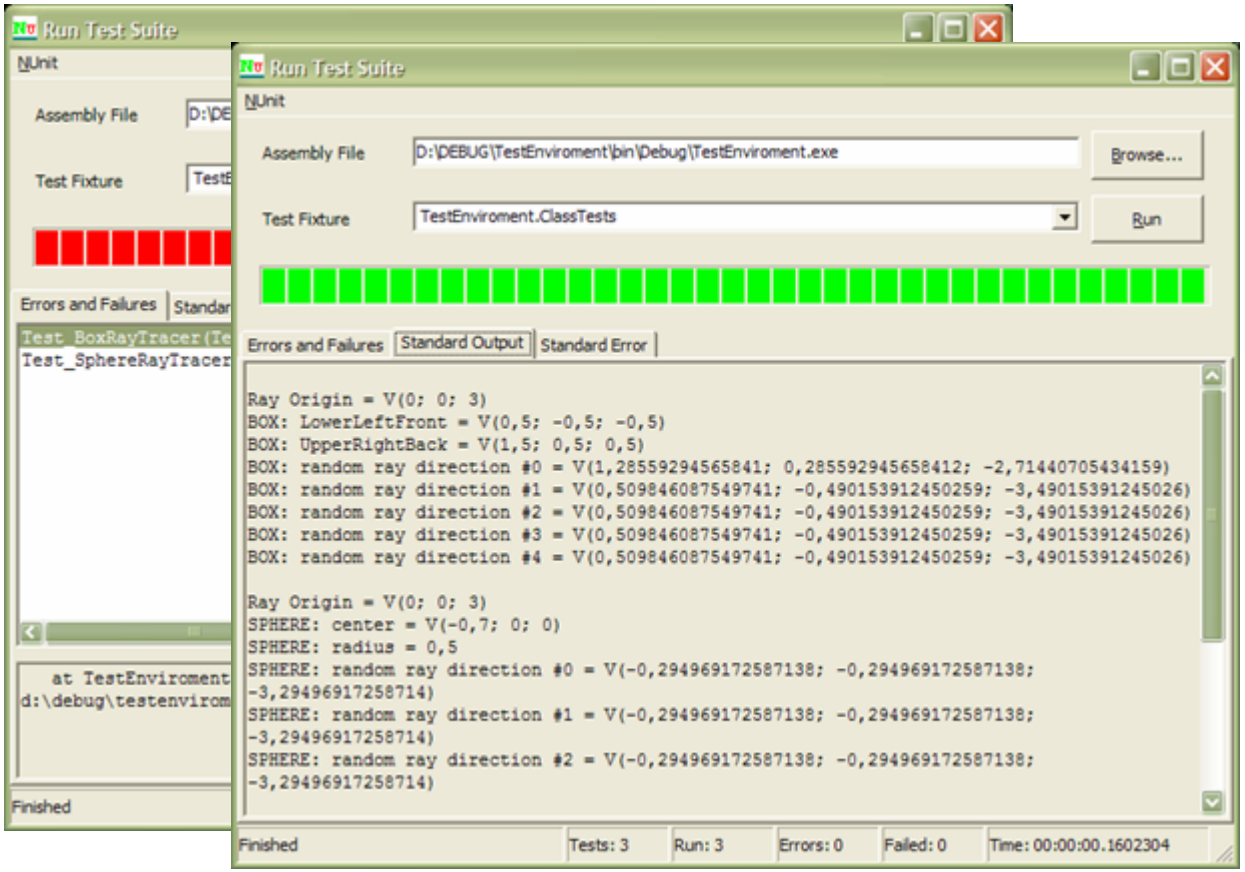
- Benötigen menschliche Kontrolle und Steuerung
- Unhandlich...
  - Festhalten der Ergebnis– / Kontrollanzeigen aufwendig, Gefahr der "Scroll Blindness"

5. Testen: Klassentest – Wie?

Einsatz eines Test – Frameworks

- Automatische Durchführung und Auswertung der Ergebnisse
- Test parallel zur Programmentwicklung

NUnit



5. Testen: Klassentest – Wie?

NUnit: Vorgehensweise

- Definieren einer Unterklasse der Klasse **`TestCase`**
- Testen benötigt eine wohldefinierte Menge an zu testenden Objekten; Diese Menge wird dann in verschiedenen Tests wiederverwendet
  - Überschreiben der Methode **`SetUp();`**
  - Freigabe evtl. alloziierten Speichers; Überschreiben der Methode **`TearDown();`**
- Eigentliche Testfälle in eigene Methoden kleiden
  - Präfix **`Test`** von Vorteil; NUnit erkennt solch benannte Methoden über Reflektion selbstständig

5. Testen: Klassentest – Wie?

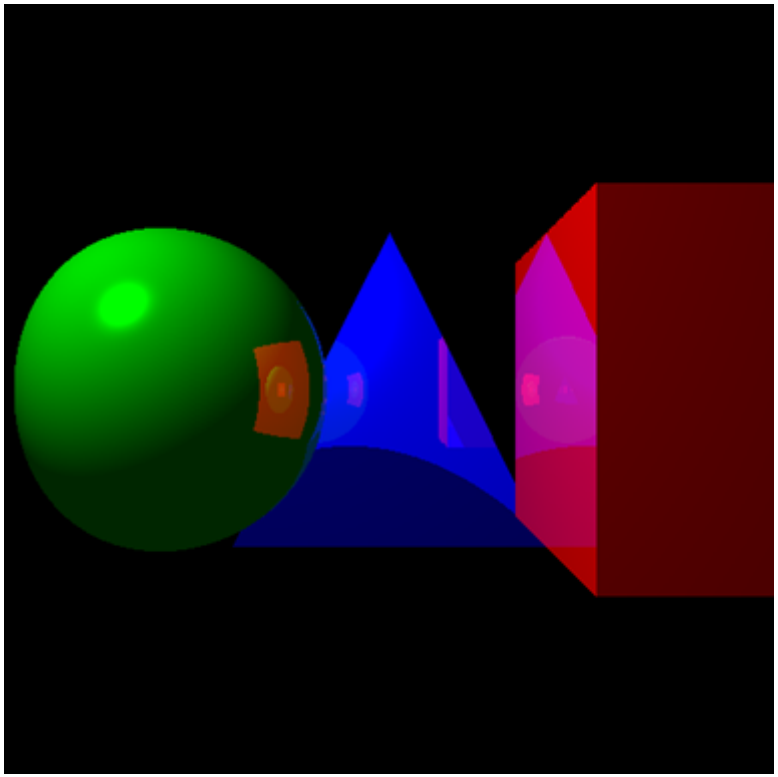
Beispiel

```
public class ClassTests : TestCase
{
    //  add an instance variable for each part of the fixture
    public BoxRaytracer      testBox;
    public SphereRaytracer   testSphere;
    public TriangleRaytracer testTriangle;

    public ClassTests(string method) : base(method) {}

    //  initialize the variables
    protected override void SetUp() {
        testBox      = new BoxRaytracer();
        testSphere   = new SphereRaytracer();
        testTriangle = new TriangleRaytracer();
    }

    public void Test_BoxRayTracer() {
        //  ARE THE VERTICES DIFFERENT?
        Assert("BOX: vertices aren't different",
            !testBox.m_LowerLeftFront.Equals(testBox.m_UpperRightBack));
        ...
    }
```

5. Testen: Integrationstest

Weiteres effektives Testverfahren hier möglich:

- Nutzer vergleicht Erwartungshaltung mit berechnetem Bild



Warum C# und .NET ?

6. Warum C# und .NET?

- Hohe Performance, da kein interpretierter Code wie in Java verwendet wird
- Sehr gute Integration von Web Services
  - Verteiltes Programmieren einfach gemacht
- Breite Unterstützung von XML
- Bereits vertraute Entwicklungsumgebung

  (vgl. Visual Studio 6.0 – Visual Studio .Net)

7.

Ausblick

7. Ausblick

Benutzung des UDDI Services

- Veröffentlichung eines WSDL-Interfaces
- Finden der Raytracer-Implementationen zur Laufzeit
- Vorteile:
  - RaytracerServices müssen nicht vorher bekannt sein
  - Verschiedene Implementationen möglich

Dynamisches Hinzufügen von neuen Shapes

- Worker melden, wenn Shape fehlt
- -> eine .DLL-Datei wird an den entsprechenden Service gesendet und dynamisch geladen
- mit Microsoft.CSharp.Compiler-Klasse eventuell Assembly erst dynamisch zur Laufzeit erzeugen

7. Ausblick

Stärkerer Einsatz von Techniken zur Teamarbeit

- konsequenter Einsatz eines Namensgebungsschematas
- CVS
- Anwendung von Prozessmodellen
- Vertiefung der Roll-Out-Phase
- Test auf mehreren Rechnersystemen

7. Ausblick - CVS

Bedeutung CVS

- Abkürzung für Concurrent Versions System
- Programm zur Verwaltung der Arbeit mehrerer Entwickler am gleichen Code
- Speicherung mehrerer Versionen der Sourcen an zentraler Stelle (Repository)

→ alle Entwickler nutzen den aktuellen Code

- kontrollierte Auflösung von Versionskonflikten

7. Ausblick - CVS

Checkout:

- Arbeitskopie auf lokalem Rechner erzeugen

Commit:

- Änderungen in Repository einfügen

Update:

- Arbeitskopie durch aktuelle Version aus Repository ersetzen

Potentieller Konflikt:

- Version der Arbeitskopie entspricht beim Commit nicht der bisherigen Version im Repository
  - halbautomatisches Verknüpfen (Merge)
  - Verzweigung der Versionen im Repository (Branching)