SOAP - Ein Überblick

- Hauptseminar -

Vortragender : Christian Brien

Betreuer : Thorsten Strufe

Datum: 13.12.2001

Inhalt des Vortrags

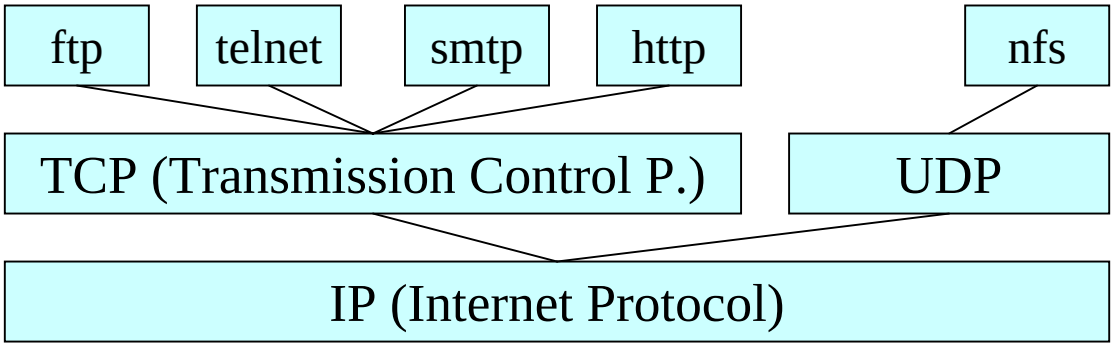
- Überblick VA-Technologien
  - Was ist SOAP ?
- Aufbau von SOAP
  - Envelope
  - Encoding
  - RPC
  - HTTP-Binding
- Anwendungsbeispiel
- Geschwindigkeit
- Verbreitung / Unterstützung durch IDEs
- Nachteile / Fazit

Überblick VA-Technologien

- Sockets
- (Sun-)RPC
- COM/DCOM
- Java RMI
- CORBA
- XML-RPC
- SOAP

Überblick VA-Technologien - Sockets

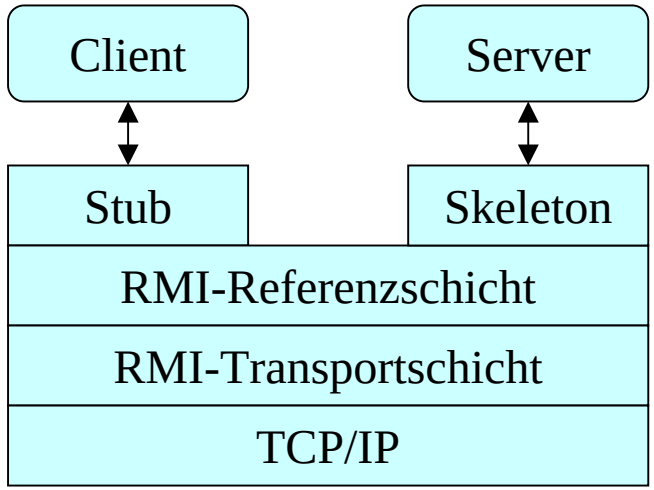
- Bestandteil des TCP/IP-Protokolls
- Adressierung durch IP und Portnummer
- keine Spezifikation des Nachrichtenformates
  - universell einsetzbar
  - immer speziell auf Anwendungsfall zugeschnitten



Überblick VA-Technologien - RPC/DCE

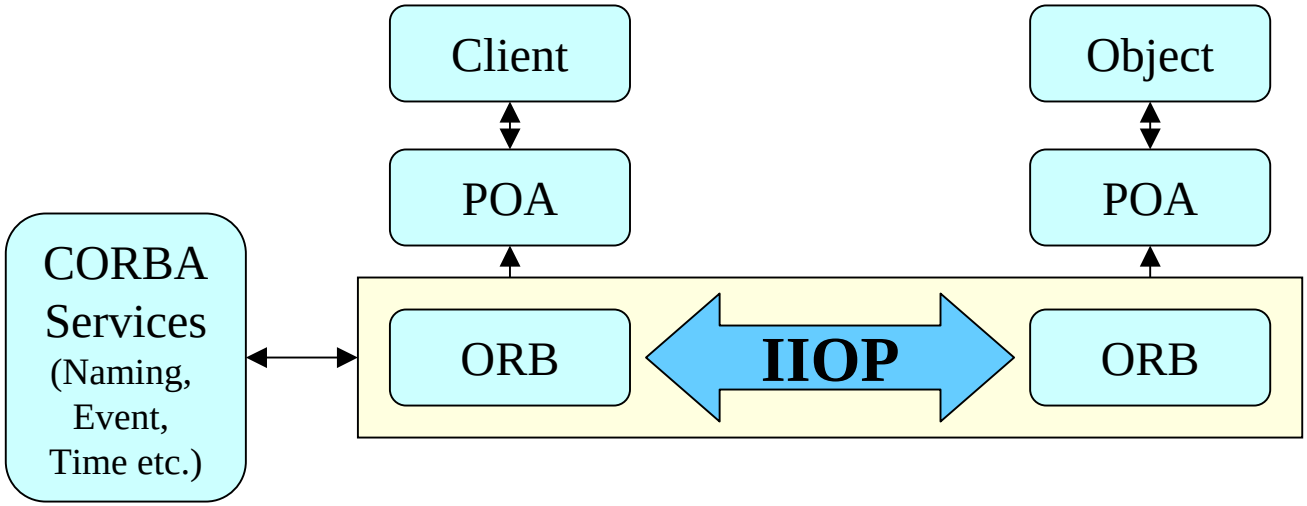
- Remote Procedure Call
- keine Objektorientierung
- External Data Representation (XDR)
- Basis für Distributed Computing Environment (DCE)

- Remote Method Invocation
  - seit JDK 1.1 fester Bestandteil von Java
- Schritt in Richtung Objektorientierung
  - Garbage Collection
- festgelegt auf Java



Überblick VA-Technologien - CORBA

- Common Object Request Broker Architecture
- Object Management Group (OMG)
  - mehr als 800 Unternehmen
- Interface Description Language (IDL)
- platformübergreifend



Überblick VA-Technologien - XML-RPC

- XML basierender RPC-Mechanismus
- Transport via HTTP
- Warenzeichen der UserLand Software, Inc.
- Direkte Adressierung über URL
- leicht zu implementieren
- kaum erweiterbar

Überblick VA-Technologien - XML-RPC Beispiel

```
POST /RPC2 HTTP/1.0
 User-Agent: Frontier/5.1.2 (WinNT)
Host: betty.userland.com
Content-Type: text/xml
Content-length: 181

<?xml version="1.0"?>
<methodCall>
  <methodName>
    examples.getStateName
  </methodName>
  <params>
    <param>
      <value><i4>41</i4></value>
    </param>
  </params>
</methodCall>
```

Client-Anfrage

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 158
Content-Type: text/xml
Date: Fri, 17 Jul 1998 19:55:08 GMT
Server: UserLand Frontier/5.1.2-WinNT

<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>
        <string>South Dakota</string>
      </value>
    </param>
  </params>
</methodResponse>
```

Server-Antwort

Überblick VA-Technologien - SOAP

- Simple Object Activation Protocol (SOAP)
  - „nur“ Protokoll und keine Architektur
- Empfehlung der W3C Organisation
  - aktuelle Version 1.2
- textorientiert, basierend auf XML-Standard
- Bindung an Standard Internet-Protokolle wie HTTP, SMTP, FTP, ...
- nicht verbindungsorientiert, Frage-Antwortmodell mit 1 Netzwerkumlauf
- sehr schlanke Protokollstruktur
- völlig interoperabel

Aufbau von SOAP - Ziele

- Einfachheit
- Erweiterbarkeit

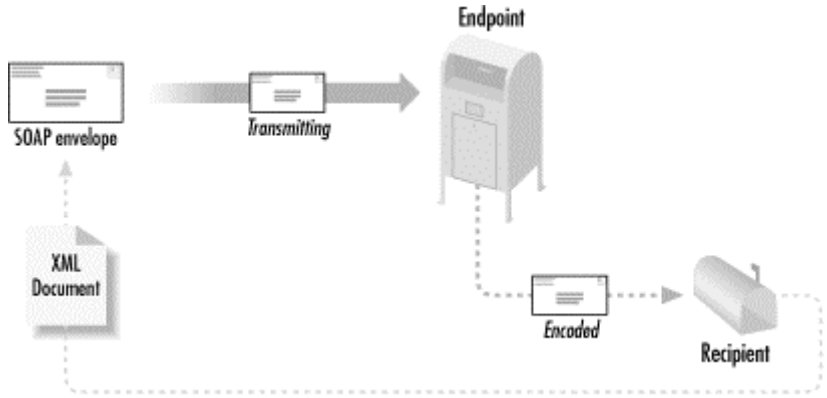
- ...unter Berücksichtigung von
  - verteilter Garbage Collection
  - Zusammenfassen und Stapeln von Nachrichten
  - Objektreferenzen
  - Objektaktivierung

Aufbau von SOAP - Spezifikation

- Envelope:
  - Framework zur Beschreibung von Inhalt, Emfänger und Verarbeitungsvorschriften der Nachricht
- Transport Binding:
  - abstrakte Vorschrift um SOAP-Nachrichten über ein darunterliegendes Protokoll auszutauschen (z.B. HTTP)
- Encoding:
  - Serialisierungsmechanismen zum Austausch von komplexen Datentypen
- RPC:
  - Konvention zur Darstellung von entfernten Prozedur-Aufrufen und Antworten

Aufbau von SOAP - Envelope

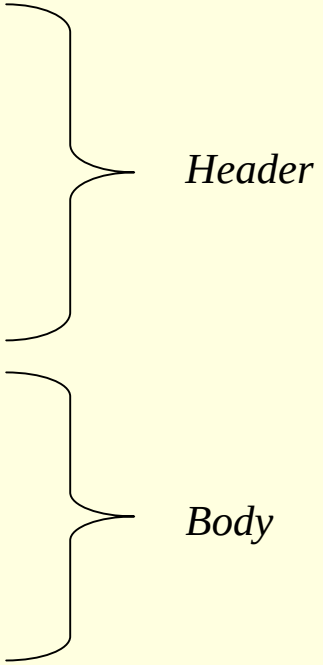
- Kopf einer jeden Nachricht
  - „Umschlag“
  - Informationen über Empfänger
  - kann Anforderungen an den Server für die Verarbeitung enthalten
  - Verwendung von XML-Namensräumen für die Syntaxüberprüfung
  - Nachricht kann entlang eines Nachrichtenpfades über mehrere SOAP-Nodes weitergeleitet werden



Aufbau von SOAP - Envelope

```
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>

 <env:Body>
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

SOAP-Message



- Header und Body enthalten kein,ein oder mehrere SOAP-Blöcke
- beim Durchlaufen mehrerer SOAP-Nodes werden jeweils verarbeitete Blöcke entfernt

Aufbau von SOAP - Envelope (Faults)

```
<env:Envelope xmlns:env='http://www.w3.org/2001/09/soap-envelope'>
  <env:Header>
    <abc:Extension1 xmlns:abc='http://example.org/2001/06/ext'
                    env:mustUnderstand='1' />
  </env:Header>
  <env:Body>
  . . .
  </env:Body>
</env:Envelope>
```

SOAP-Message

Erweiterung, die vom Sever verstanden werden muß

```
<env:Envelope xmlns:env='http://www.w3.org/2001/09/soap-envelope'
                    xmlns:f='http://www.w3.org/2001/09/soap-faults' >
  <env:Header>
    <f:Misunderstood qname='abc:Extension1'
                          xmlns:abc='http://example.org/2001/06/ext' />
</env:Header>
  <env:Body>
    <env:Fault>
      <faultcode>env:MustUnderstand</faultcode>
      <faultstring>One or more mandatory headers not understood</faultstring>
    </env:Fault>
  </env:Body>
</env:Envelope>
```

SOAP-Response

Fehler-String

Aufbau von SOAP - Transport Binding (HTTP)

- Bis jetzt noch keine allgemeinen Spezifikationen
- Spezielle Form: HTTP-Binding
  - Nachrichten werden ähnlich dem XML-RPC über HTTP-Post-Requests verschickt
  - zusätzliches optionales Attribut *SOAPAction* mit URI
  - *Content-Type: text/xml*
  - die Antwort erfolgt über ein HTTP-Response
  - Rückgabe immer 200=OK da eine getrennte Fehlerbehandlung erfolg
- auch Binding an POP/SMTP möglich
- ...

Aufbau von SOAP - Encoding / RPC

- Nutzung von XML-Schemas zur Deklaration von neuen Datentypen
  - Gruppierung
  - Vererbung
  - Kardinalitäten
  - Aufzählungen
  - Listen
  - Unions
  - etc.

```
<xsd:schema
  xmlns:xsd=„http://www.w3.org/2001/XMLSchema“
  xmlns=„Nexus “
  targetNamespace=„Nexus “ >
  <xsd:element name =„ViLiS“ type=„VIT“/>
  <xsd:complexType name = „VIT“>
     <xsd:element name=„x“ type=„xsd:int“ />
     <xsd:element name=„y“ type=„xsd:int“ />
     <xsd:attribut name=„identifier“ type=„xsd:string“ />
  </xsd:complexType>
</xsd:schema>
```

- Erweiterung der Basistypen durch Implementierung von eigenen Mapping-Klassen

```
<xsd:mappings>
<xsd:map encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="urn:cd-catalog-demo" qname="x:cd"
   javaType="javaxml2.CD"
   java2XMLClassName="org.apache.soap.encoding.soapenc.BeanSerializer"
   xml2JavaClassName="org.apache.soap.encoding.soapenc.BeanSerializer"/>
</xsd:mappings>
```

Aufbau von SOAP - Encoding / RPC

- Abbildung von RPCs auf SOAP-Requests

```
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV=„http://schemas.xmlsoap.org/soap/envelope/“>
 <SOAP-ENV:Header>
    <t:TransactionCode xmlns:t=„my-URI“ xsi:type=„xsd:int“
       mustUnderstand=„1“
       actor=„http://schemas.xmlsoap.org/soap/actor/next “>
       156533245
    </t:TransactionCode>
 </SOAP-ENV:Header>
 <SOAP-ENV:Body>
    <method:getNextViLiS xmlns:method=„my-URI2“>
       <xCoord>15</xCoord>
       <yCoord>124</yCoord>
    </method:getNextViLiS>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

NameSpace mit Soap-Vokabular

Header mit Zusatzinfos

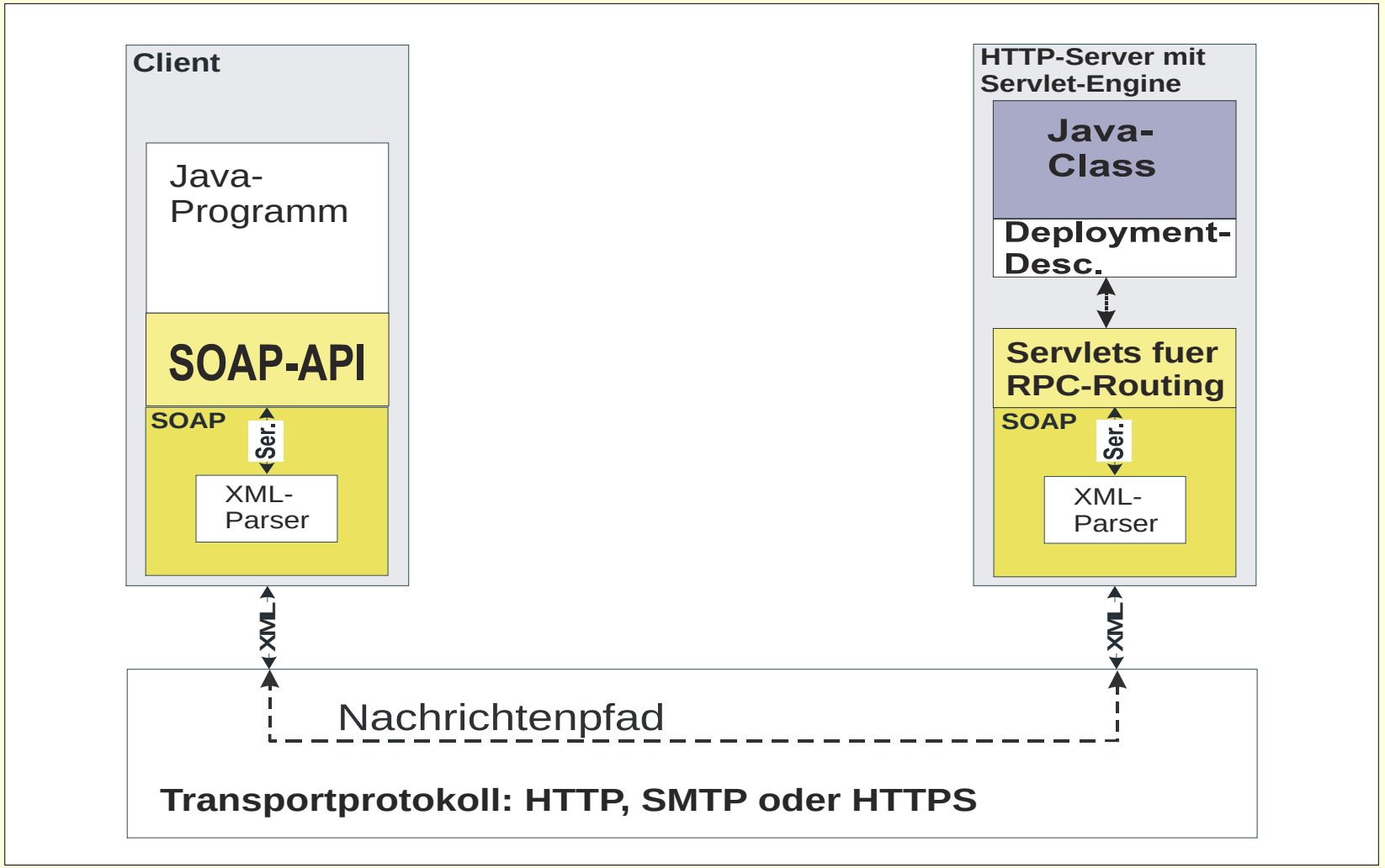
Methodenaufruf mit Parametern

Anwendungsbeispiel

Umgebung:

- Apache WEB-Server
- Tomcat Servlet-Engine
- Apache SOAP
  - soap.war nach \WebApps
- JAXP-kompatibler XML-Parser (Xerces)
  - mit NameSpace-Unterstützung
- JavaBeans Activation Framework (Sun)
- Mail-Package (POP3/SMTP)

Anwendungsbeispiel - Apache SOAP Architektur



Anwendungsbeispiel - Server (Class)

```
package javaxml2;
import java.util.Hashtable;

public class CDCatalog {
  private Hashtable catalog;
  public CDCatalog( ) {
    catalog = new Hashtable( );
    catalog.put("Nickel Creek", "Nickel Creek");}

  public void addCD(String title, String artist) {
    if ((title == null) || (artist == null)) {
      throw new IllegalArgumentException("Title and artist cannot be null.");}
    catalog.put(title, artist);}

  public String getArtist(String title) {
    if (title == null) {
      throw new IllegalArgumentException("Title cannot be null.");}
    return (String)catalog.get(title);}

  public Hashtable list( ) {
    return catalog;}
}
```

CDCatalog.java

- Normale Java-Klasse
- Wiederverwend-barkeit von bereits vorhan-denen Klassen

Anwendungsbeispiel - Server (ClassDD)

```
<isd:service xmlns:isd="http://xml.apache.org/xml-soap/deployment"
id="urn:cd-catalog"
>
<isd:provider type="java"
scope="Application"
methods="addCD getArtist list"
>
<isd:java class="javaxml2.CDCatalog" static="false" />
</isd:provider>

<isd:faultListener>org.apache.soap.server.DOMFaultListener</isd:faultListener>
</isd:service>
```

CDCatalogDD.xml

- Deployment Descriptor, um die Klasse beim Apache anzumelden

C:\java org.apache.soap.server.ServiceManagerClient
http://localhost:8080/soap/servlet/rpcrouter deploy xml\CDCatalogDD.xml

Anwendungsbeispiel - Client (Ablauf)

- Erstelle einen SOAP-RPC-Aufruf
- Setze die Type-mappings für eigene Parameter
- Setze die URI des SOAP-Services
- Lege fest, welche Methode aufgerufen werden soll
- Setz das zu benutzende Encoding
- Füge die Parameters zum RPC-call hinzu
- Verbinde mit dem SOAP-Service
- Empfange und verarbeite die Antwort

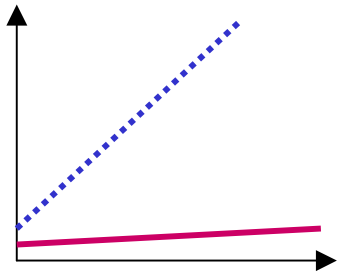
Anwendungsbeispiel - Client

```
public class CDAdder {
  public void add(URL url, String title, String artist)
  throws SOAPException {
    System.out.println("Adding CD titled '" + title + "' by '" +artist + "'");
    Call call = new Call( );
    call.setTargetObjectURI("urn:cd-catalog");
    call.setMethodName("addCD");
    call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

    Vector params = new Vector( );
    params.addElement(new Parameter("title", String.class, title, null));
    params.addElement(new Parameter("artist", String.class, artist, null));
    call.setParams(params);

    Response response;
    response = call.invoke(url, "");
    if (!response.generatedFault( )) {System.out.println("Successful CD Addition.");}
    else {Fault fault = response.getFault( );
      System.out.println("Error encountered: " + fault.getFaultString( ));}
   }

public static void main(String[] args) {
...
```

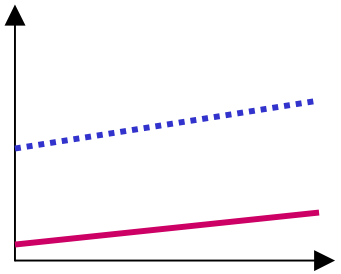
CDAdder.java

Geschwindigkeit

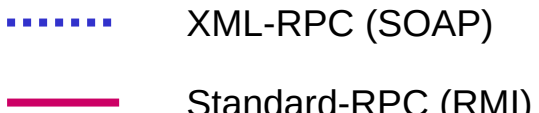
- Problem des Parsens bei jedem Aufruf
- Zeitverlust durch Syntaxcheck
- extrem bemerkbar bei vielen kleinen Aufrufen
  - viel Protokoll-Overhead, wenig Nutzdaten
- bei Verwendung von großen Strings (oder Base64 encodeten Bytestreams) verringert sich die Wirkung des Protokoll-Overheads
- Zeitverhalten pro Aufruf >Faktor 10 schlechter zu RMI



Zeit / Anzahl Calls



Zeit / Größe Calls



Verbreitung / Unterstützung durch IDEs

- Stetig wachsend
- nutzt vorhandene Architekturen
- große Bedeutung innerhalb der WebServices (IBM)
  - UDDI und WSDL
- Kommunikationsbasis für Microsoft .NET Technologie
- Software AG mit Tamino XML-Server
  - Application Integration Broker -> RPC
- IBM DB2 XML-DB (Websphere)
- etc...

- Microsoft Visual Studio
- Borland Delphi 6 (VCL-Komponenten), Kylix 2

Nachteile

- Keine direkte Unterstützung von Sicherheitskonzepten oder Transaktionen
  - nur indirekt in Form von z.B. HTTPS
- größerer Aufwand für Aufbereitung und Transport der Daten
  - „aufgeblähtes“ XML-Format
  - Einsatz von XML-Parsern an den Endpunkten

Fazit

- Zunehmend schnelle Verbreitung
- Einfach in bestehende Architekturen zu integrieren
- noch in der Entwicklungsphase
  - Bemühungen des W3C für SOAP-Nachfolger XML Protocol
  - geplante Features von XMLP:
    - *komplette Architektur für verteilte Anwendungen*
    - *bessere Unterstützung von Zwischenstationen*
    - *Integration von Mechanismen für Sicherheit*

Fragen ??

Danke für die Aufmerksamkeit !

- ENDE -