Software Reuse

8. Web Services

Peter Sturm
Universität Trier

Ausgangspunkt

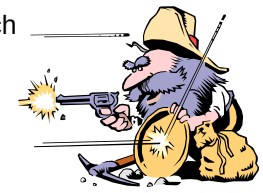
- Client/Server-Systeme
  - Traditioneller RPC
  - OO-Pendant RMI (CORBA)
- Probleme
  - Installationbedarf auf Clientseite
  - Aufwendige Installation auf Serverseite
  - Hoher Aufwand bei Management und Pflege
  - Proprietäre Plattformen
  - Eingeschränkte oder keine Interoperation
  - Verbindungsaufbau in abgesicherten Netzen problematisch
  - Sicherheit und Authentifizierung

Web Services

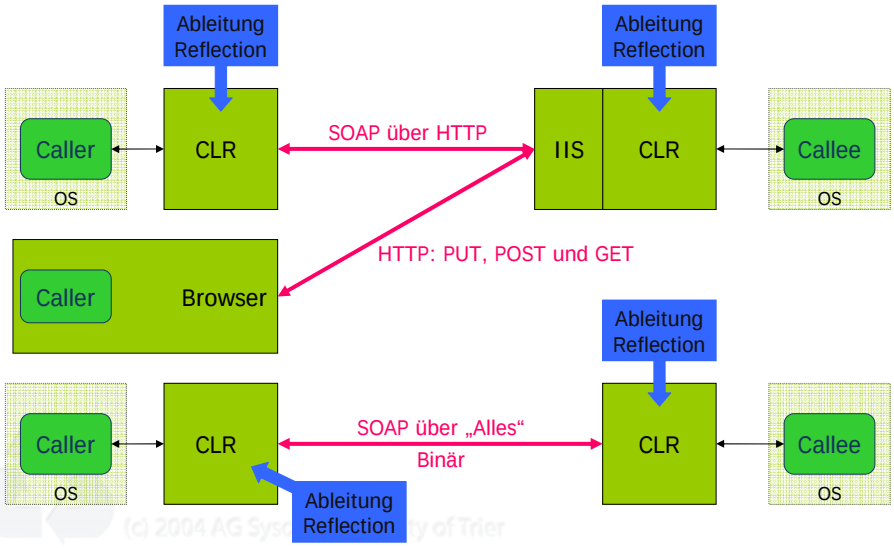
- Deskriptiven RPC
  - XML-basiert ⇒ Interoperabilität
- Reflektive und virtuelle Sprachplattformen
  - IDL-Compiler und statische Stubs unnötig
  - Interoperabilität
- Vereinfachtes Deployment auf Serverseite
  - Assemblies im .NET Framework
  - EJBs in J2EE
- Keine Installation auf Clientseite
  - WWW-Browser mit Applet o.ä.
  - Zugang über XML-basierter RPC (kein UI)
- HTTP wird Tunnelprotokoll

Stand 2003

- Konkurrenzkampf zwischen Java und .NET
- .NET basiert auf SOAP
  - Weitreichende Integration in Visual Studio .NET und IIS
  - Noch keine Application Server Funktionalität verfügbar
- JAVA
  - definiert eigenen XML-basierten RPC (JAX-RPC)
  - es geht auch über SOAP (java.xml.soap Package)
- Noch viele offene Fragen
  - Skalierbarkeit, Schutz, …
- OBDA: Nachfolgende Folien sind .NET-spezifisch



Remoting, Web Form und Web Service



Bausteine

- Discovery
  - Suche eines gewünschten Dienstes
  - Erweiterter Directory-Dienst
  - Semantik
  - Forschungsgebiet Ontologien
- Transport
  - Praktisch alle Transport-mechanismen nutzbar (auch TCP etc.)
  - Konzentration auf HTTP und SMTP (Allgegenwärtig)

Software Reuse

Simple Object Access Protocol (SOAP)

Briefumschläge

- SOAP Envelop umfaßt
  - SOAP Header: Informationen über die Nachricht (optional)
  - SOAP Body
  - und ggf. anderes

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/>
    <soap:Header>
    …
    </soap:Header>

    <soap:Body>
    …
    </soap:Body>
</soap:Envelope>
```

Actors

- Default Actor
  - Eigentlicher Empfänger der Nachricht
- Intermediary
  - Delegation des Aufrufs
  - Kann Seiteneffekte haben
  - Kann Nachricht verändern

SOAP Header

- Optionale Information
- Einsatzbeispiele
  - Information über im Body genutzte Komprimierungsverfahren
  - Authentifizierung
  - Digitale Signatur
  - Routing-Informationen
  - Kontextinformation für Transaktionen
  - Bezahlung
- Empfänger kann optionale Header-Einträge ignorieren
  - Attribut mustUnderstand

SOAP Body

- Jeder Envelope braucht seinen Body ☺
- Inhalt kann beliebig sein
  - Zeichenkette
  - Kodierte Bytefolge
  - XML
- Zwei Kategorien
  - Prozedurorientiert (RPC)
  - Dokumentorientiert

Darf XML-Syntax natürlich nicht verletzen

Fault Element

- Standardmechanismus um Fehler zurück-zumelden
- Fault Codes
  - VersionMismatch
    - Ungültiges Element
  - MustUnderstand
  - Client
    - Gesendete Nachricht fehlerhaft
  - Server
    - Serverseitiger Fehler

```
<?xml version="1.0"?>
…
    <soap:Body>
        <soap:Fault>
            <soap:faultcode>
                Client.Security
            </soap:faultcode>
            <soap:faultstring>
                Raus hier :-(
            </soap:faultstring>
            <soap:faultactor>
                http://hier
            </soap:faultactor>
            <soap:detail>
                …
            </soap:detail>
        </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

SOAP Encoding

- Einfache Typen
- Zusammengesetzte Typen
  - Structures
  - Arrays
- Referenzparameter möglich
  - Beispiel f(x,x) mit benutzerdefiniertem Typ x
- Entwicklungsunterstützung
  - IDE setzt Standardsemantiken automatisch um
  - Spezialfälle (Referenzen) u.U. manuell realisieren

Protokollbindung

- SOAP beschreibt RPC-Verkehr
- In SOAP selbst ist lediglich die HTTP-Bindung spezifiziert
  - HTTP tunnelt Firewalls (Port 80 immer offen)
  - Wohldefinierte Erweiterung von HTTP
    ⇒ Ebene-7-Firewall kann filtern (noch Ausnahme)
  - HTTP mächtige Werkzeugunterstützung
  - Inhärent Zustandslos
  - Request-Response-Struktur direkt nutzbar

HTTP-Einbettung

```
POST /SomeWebService HTTP/1.1
Content-Type: text/xml
SOAPAction: http://somewhere/someservice.wsdl
Content-Length: 243
Host: sshort3

<?xml version="1.0" encoding="utf-8"?>

<soap:Envelope xmlns:soap=http://schemas...
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/">

    <soap:Body>
        <Add>
            <x>22</x>
            <y>20</y>
        </Add>
    </soap:Body>

</soap:Envelope>
```

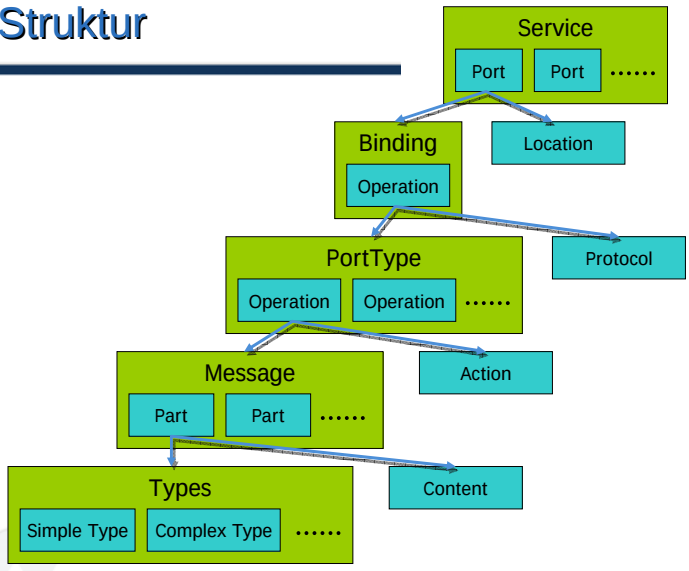

Software Reuse

Web Service Description Language (WSDL)

WSDL

- Vollständige Beschreibung eines Web Service
- Dazu gehört
  - XML Schema Definition der kommunizierten Typen
  - Definition der Nachrichten auf der Grundlage der eingeführten Typen
  - Die Schnittstellen des Dienstes
  - Erlaubte Protokollbindungen
- WSDL ist selbst XML-basierte Sprache

Struktur



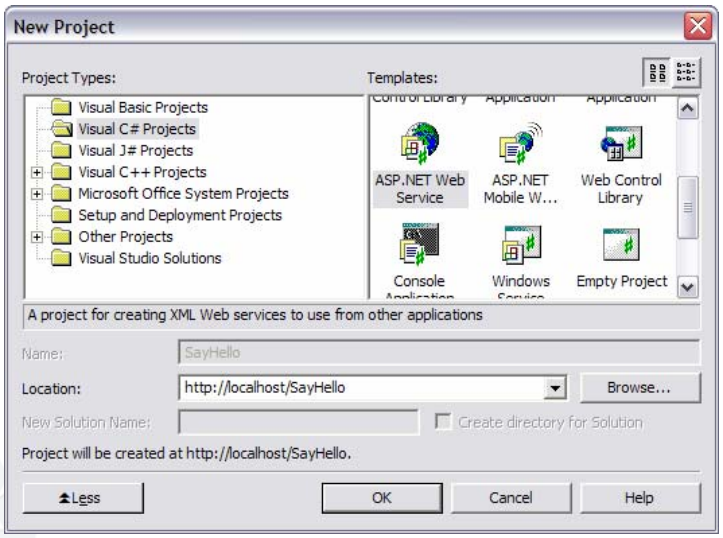
Software Reuse

Ein Web Service Beispiel

SayHello

- Implementierung eines einfachen Grüßdienstes:
  - string Greetings (string name)
    - Greetings(“Peter”) liefert “Hallo Peter”
- C#-Implementierung
- Zugang über IIS

Projekt: ASP.NET Web Service

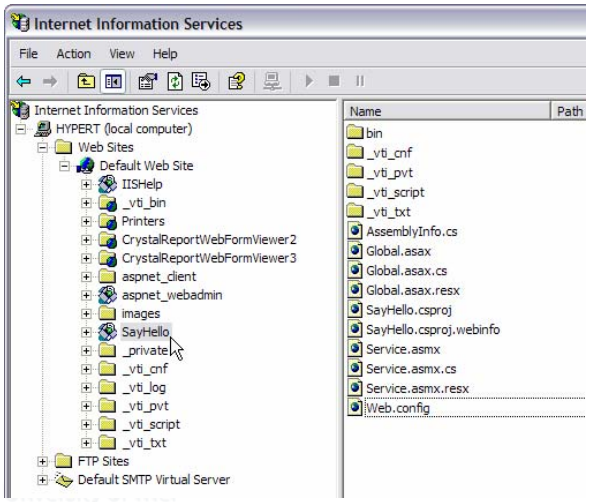


Implementierung

```
SayHello.Service                                   Service()

using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace SayHello
{
    /// <summary>
    /// Summary description for Service.
    /// </summary>
    [WebService(Namespace="http://sturm.de/webservices/")]
    public class Service : System.Web.Services.WebService
    {
        public Service()
        {
            //CODEGEN: This call is required by the ASP.NET Web Services Designer
            InitializeComponent();
        }

        Component Designer generated code

        [WebMethod]
        public string Greetings ( string name )
        {
            return "Hello " + name;
        }
    }
}
```

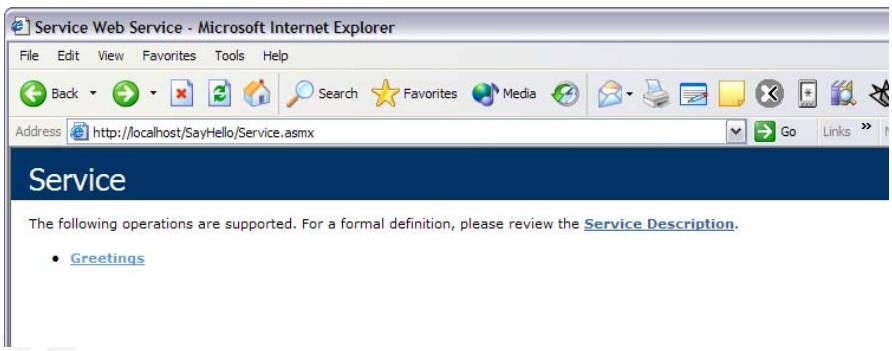
Build Solution

- Automatisches Deployment auf dem angegebenen WWW-Host
- Trennung von Dienst und Code im produktiven Umfeld sinnvoll

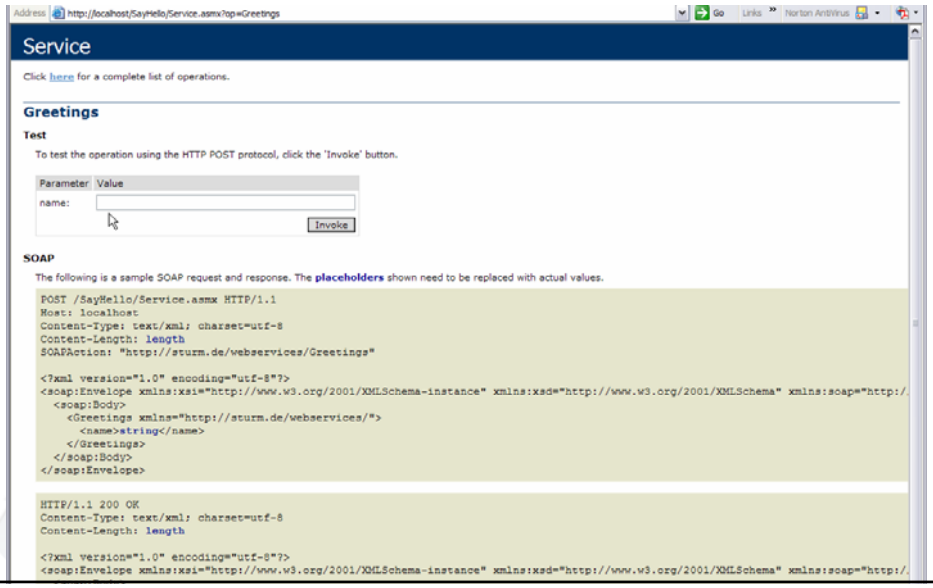


Testen des Services

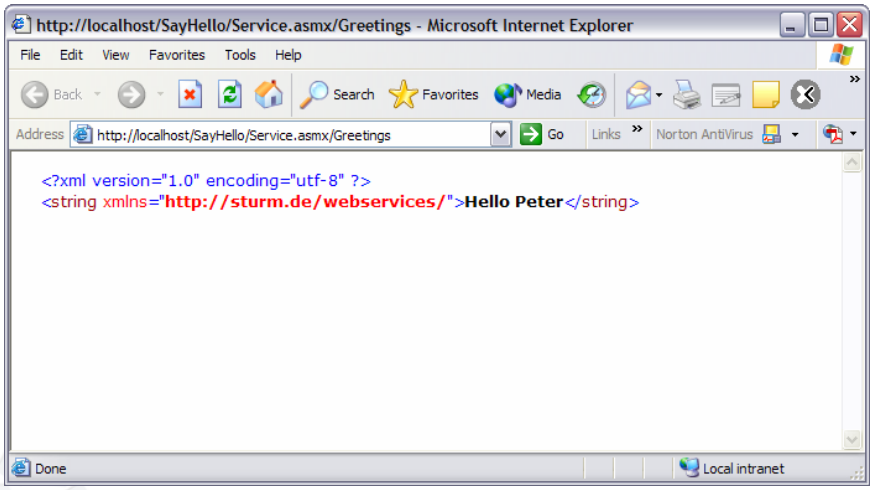
- Direkte Unterstützung des IIS
- Alternative Tools wie z.B. XMLSpy u.a.



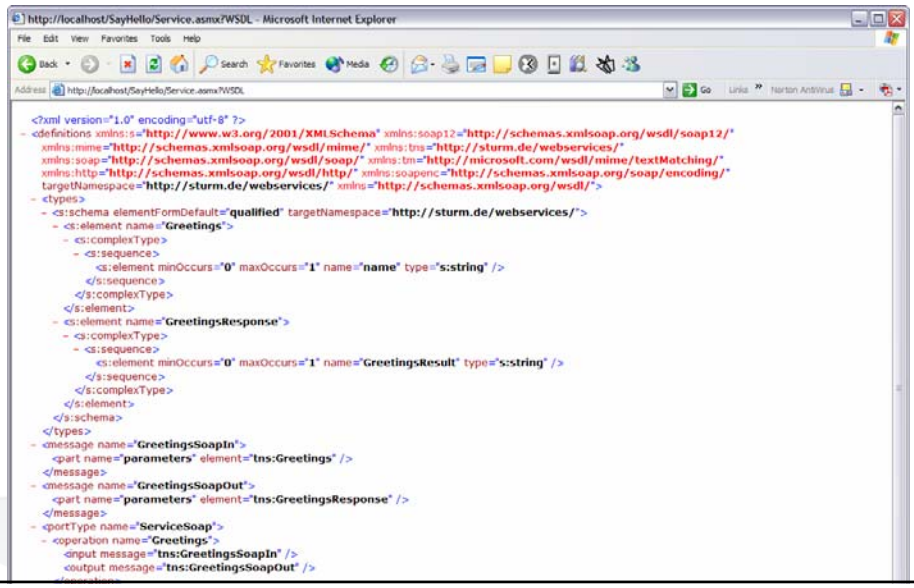
Aufruf einer Web-Methode



… und die Antwort



Die WSDL-Beschreibung



Software Reuse

Beispiel-Client nutzt Web Service

Der Client


```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace WindowsClient
{
    /// <summary>
    /// Summary description for Form1.
    /// </summary>
    public class Form1 : System.Windows.Forms.Form
    {
        private System.Windows.Forms.TextBox textBox_Input;
        private System.Windows.Forms.Button button_Los;
        private System.Windows.Forms.TextBox textBox_Output;
        /**/
        private System.ComponentModel.Container components = null;

        public Form1()...
        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )...
        Windows Form Designer generated code

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new Form1());
        }

        private void button_Los_Click(object sender, System.EventArgs e)
        {
            localhost.Service s = new localhost.Service();
            textBox_Output.Text = s.Greetings(textBox_Input.Text);
        }
    }
}
```