Workshop on Web Services
SE Day 2001 – Munich and Nuremberg

Michael Stal, Senior Principal Engineer
Corporate Technology
Dept. CT SE 2 – Architecture
Phone: +49 89 636 49380
E-Mail: Michael.Stal@mchp.siemens.de

Content

• Problems today
• The Web – Today and Tomorrow
• Web Services - The Web is The Computer
• Web Frameworks
• Case Study: Web Services in .NET
• Limitations
• Summary
Problems to be tackled for tomorrow's products

- Web- and Net-based Integration of Applications, Services, and Systems
- Quality of Service issues such as availability, resource and time constraints, ...
- Deployment of Automated and Autonomous Systems
- Smart Services that share and distribute context
- Flexible Data Exchange
- Software as Service (aka ASP)

Consequence: Agility does matter

- Developers should not make too many assumptions about usage contexts and environments \textit{a priori}.
- Developers don’t live on an island and must integrate legacy code and other peer systems.
- Programmers need to develop and deploy their software on different heterogeneous machines and devices.
- Running applications are reconfigured instead of recompiled.
- Software Engineering must cope with change requests in Internet time.
Consequence: Architecture does matter

- "Software entropy" should be maximized: loosely coupling between peers, decentralized information access, reflective approaches (Just-in-Time Integration).
- Software should be partitioned into composable services that can be spontaneously connected and orchestrated by business/technical processes (component-based software).
- Software must be e-nabled.
- Legacy code must be integrated in order to protect investments.

• Can today's Web technologies offer an appropriate solution?

The Web today: Component Age

- Dynamic context-aware content
- Good scalability possible due to component-based approach
- Easy to implement
- Separation of concerns: Component can access/implement business logic and workflow while Server Pages handle layout issues and user interaction
But there are some limitations ...

• Some observations:
  • While the backend has always been subject to evolutionary change, the frontend remained the same:
    • Only human users can access services and information.
    • Backend and frontend denote different worlds connected by HTTP but separated by the Web Server barrier.
  • Using Java applets and middleware doesn’t work because of
    • security restrictions imposed by firewalls,
    • huge number of different middleware technologies available.

Solution: The Web is the Computer

• Establish an architecture that
  • supports web-based inter-application communication paradigms,
  • provides location transparency,
  • allows to add, exchange, or remove services dynamically,
  • hides system details from the developer.
Architectural Solution: The Web as Middleware Technology

Dynamics
How to implement this vision ...

Step 1: Define a Transport Protocol

- A protocol defines syntax, semantics and order of messages exchanged between peers.
- For a Web-based transport protocol:
  - Use HTTP and other Internet Protocols as transport layer and
  - introduce a self-describing data representation format using XML.
  - In detail: represent each request and each response as a XML message and send it over the wire.
  - Welcome to the Simple Object Access Protocol (SOAP).
Step 1 – Example

- Example for a SOAP Message:

  ```xml
  <soap:Envelope>
    <soap:Header>
      <transaction>
        <soap:mustUnderstand stand="true" xmlns="http://tx.com">
          <id>12345678</id>
        </transaction>
    </soap:Header>
    <soap:Body>
      <m:getPhoneNumber>
        <theName>Bill Gates</theName>
      </m:getPhoneNumber>
    </soap:Body>
  </soap:Envelope>
  ```

  - SOAP messages are bracketed by envelopes
  - headers are optional
  - this feature must be supported by the receiver
  - message body

Step 2: Define a Service Description Language

- A description language must be available to define Web service interfaces and how to invoke them.
- We can use XML to
  - specify deployment information
  - and structural information.
- That is what the Web Service Description Language (WSDL) is all about.
- Note: WSDL addresses syntax, not semantics!
WSDL Elements

- **Port**
  - Concrete address of WebService (e.g., URL and port)
- **Service**
  - Collection of ports
  - Physical location of end point
- **Message**
  - Format for single transfer
  - Request and Response are separate messages
- **PortType**
  - Logical grouping of messages to operations
- **Binding**
  - Maps PortType to implementation (e.g., SOAP)
  - Concrete interface of service
- **Types**
  - Type definitions used in Messages (XML Schema Notation)

WSDL Example: Message and PortType

```xml
<message name="CelsiusToFahrenheitRequest">
  <part name="Celsius" type="xsd:double" />
</message>
<message name="CelsiusToFahrenheitResponse">
  <part name="Return" type="xsd:double" />
</message>
<portType name="Temperature.ctemperaturePortType">
  <operation name="CelsiusToFahrenheit">
    <input message="tns:CelsiusToFahrenheitRequest" />
    <output message="tns:CelsiusToFahrenheitResponse" />
  </operation>
</portType>
```
WSDL Example: Binding and Service

```xml
<binding name="Temperature.ctemperaturebinding"
   type="tns:Temperature.ctemperaturePortType">
   <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http" />
   <operation name="CelsiusToFahrenheit">
      <input>
         <soap:body … />
      </input>
      <output>
         <soap:body … />
      </output>
   </operation>
</binding>

<service name="Temperature.ctemperatureService">
   <port name="Temperature.ctemperaturePort"
      binding="tns:Temperature.ctemperaturebinding">
      <soap:address location="http://localhost/converter.asmx" />
   </port>
</service>
```

Step 3: Define a global Service Broker

- UDDI is a global registry
- Registration possible at any node
- Registrations replicated at daily basis
- Common SOAP protocol used
What is UDDI?

White Pages
- Directory of names
- Provider info
- Contact info

Yellow Pages
- Directory of domains
- Specific search using context such as location, service type, ...
- Points to White Pages

Green Pages
- Directory of business info
- Information about business model
- Technical details of provided services
- Information on business processes

Step 4: All you need are Generators

- Introduce tools that generate glue for connecting you with the Web ORB, i.e.,
  - Client Proxies for connecting the client with services.
  - Server Proxies for seamless service deployment.
Services alone are not enough...

- Information and services must be consumable from any device and from any place:
  - We need a platform that is device independent (virtual machine).
- New services must be composable from existing services and transparently accessible by consumers:
  - We need a middleware approach that provides code AND data interoperability (SOAP, UDDI, WSDL, XML).
- Support of „old-style“ Web content AND Web services is required:
  - We need advanced Web servers as gateways to Web pages and services.

Further Ingredients

- Web services should be smart and context-aware:
  - We need means to let services exchange and share contexts. Contexts denote out-of-the-band information (e.g., information about location, sessions, transactions, preferences, or callers) used to adapt and control service behavior.
- Mega Services should be available:
  - We need to leverage core services such as transaction monitors, database systems, calendars, authentication services.
- Business processes and workflows should be automated:
  - We need workflow engines.
- Existing legacy code needs to be integrated:
  - We need connectors and standard middleware (J2EE, CORBA 3, COM+).
Further Ingredients (cont’d)

- **New flexible communication paradigms are needed:**
  - We require Peer-to-Peer (P2P) solutions to let applications and users share information within virtual environments.
  - Mobile solutions help to access services from any place.

- **Decentralized, heterogeneous systems are difficult to manage:**
  - We need sophisticated solutions to prevent administration hell.

- **Non-functional requirements must be guaranteed:**
  - We need support for fault-tolerance, load-balancing, multimedia streams.

- **Last but not least, we need tools:**
  - We need programming environments, content management tools, ...

---

Services alone are not enough: Web Frameworks

**Core Elements of Web Frameworks**

- Web Service User/Provider
- Integration Layer
- Mainframe
- Backend Server
- Legacy
- Micro/Macro Services
- Virtual Machine
- Workflow Engine
- Frontend Layer (Web Server)
- Service Context (Who, Where, When, Why, …)
- Service Description (WSDL)
- Service Description, Discovery, Integration (UDDI)
- Web-based and -related Protocols (HTTP, XML, SMTP, …)
What we finally get

- Out-of-the-box interoperability and integration through standardized communication protocols (XML, HTTP, ...).
- Dynamic discovery, integration, and binding of services (WSDL, UDDI).
- Service discovery and trading using standardized business interfaces.
- Orchestration of services using workflow engines.
- Advanced context-aware services.
- Integration of legacy code through standard middleware (CORBA, EJB, COM+).
- Device independance through virtual machines (Java’s JRE, .NET’s CLR).

Advantages of Web Services

- Infrastructure services from 3rd parties such as authentication, backup, ... can be seamlessly integrated.
- IT experts can focus on business logic instead of needing to integrate the infrastructure.
- Easy maintenance of networked systems:
  - Remote Administration and Control
  - Dynamic Software Updates
- Connecting internal and external services enables new types of applications.
- Interoperability of heterogeneous platforms highly improved.
  - Common accepted standards such as SOAP, WSDL, UDDI
  - Web technology can be integrated much easier in existing IT in contrast to proprietary protocols.
- Web Service enabled development tools increase productivity and reduce time-to-market.
Technologies and Products

- There are many products available, even open source solutions:
  - Apache SOAP
  - Pocket SOAP for mobile devices
  - IBM Webservices Toolkit
  - Sun ONE (Open Net Environment); J2EE support expected in 2002
  - Silverstream JBrokerWeb
  - Borland Delphi Webservices (support for JBuilder to come soon)
  - Microsoft .NET and .NET Compact Framework (Beta 1 in October)
  - ... Many, many more ...

- Basically the main competitors are:
  - Webservices based upon Java
  - Webservices based upon Microsoft .NET

Mobile Web Services

- Mobile devices may also leverage Web Services
  - but they add new requirements such as
    - Network connectivity
    - Context awareness
    - Session management

- Current toolkits available (PocketSOAP, kSOAP) do support Web Services at a very basic level:
  - Insufficient WSDL integration
  - No automatic proxy generation

- Almost no support for SOAP servers on devices and appliances:
  - Support not available in current programming environments
  - However, high potential for enterprise integration and P2P
Case Study: Web Services in .NET

Runtime (.NET)

- The runtime of .NET (CLR = Common Language Runtime) offers compilation and language interoperability.

Diagram showing the compilation and execution processes with languages such as C#, VB.NET, C++/CLI, and Perl, among others. The diagram also highlights the components like MSIL + Metadata, Compiler, Loader/Verifier, JIT, Execution, Managed Code, Garbage Collection, Security, Multithreading, etc.
Using Web Services in .NET

- .NET offers a comfortable implementation:

```csharp
namespace WebService1 {
        // lot of stuff omitted
        [WebMethod]
        public double DM_to_Euro(double value) {
            return value / 1.95583;
        }
        [WebMethod]
        public double Euro_to_DM(double value) {
            return value * 1.95583;
        }
    }
}
```

Using Web Services in .NET (cont'd)

- Usage of Web services is simple as well:

```csharp
localhost.Service1 s1 = new localhost.Service1();
double result = s1.Euro_to_DM(200);
```

- You might also call the service asynchronously:

```csharp
localhost.Service1 s1 = new localhost.Service1();
IAsyncResult ar = s1.BeginEuro_to_DM(200, null, null);
while (!ar.IsCompleted) {
    // enjoy your coffee
}
double result = s1.EndEuro_to_DM(ar);
```

- Callback and cancel/timeout are also possible.
Using Web Services in .NET (cont’d)

- Web Services in .NET supplement ASP.NET and facilitate programmatic access.
- But:
  - Microsoft doesn’t offer ebXML-conformance => Vendor-Lock-in
  - Same for Hailstorm (Building Block Services)

Demo: Stock Info Server

1. SOAP-CALL: GetStockInfo(<COND>)
2. ADO.NET ACCESS to Database
3. [optional] SOAP CALL: Euro_to_DM
There is no free lunch

• Today, we face several limitations:
  • Unsufficient integration of embedded and mobile devices.
  • Limited availability of mature solutions.
  • Interoperability not guaranteed.
  • Sharing/transmitting context not fully supported. For example: How can transactions be shared between services?
  • Integration of multi-media streaming still an open issue.
  • Limited set of communication semantics.
  • WSDL specifies syntax and not semantics.
  • Quality of Services issues such as availability unresolved.
  • Web services resort to flat procedure calls. Identity must be provided by programmers. What about Web objects?

Summary

• Web services are based on XML, Object-Oriented Middleware, and Internet protocols.
• We are living in an agile world. Web services enable agility by:
  • relying on standardized protocols and hiding other middleware details behind the Web server (loosely coupling),
  • using globally available services for discovery, description, and integration such as UDDI, WSDL (decentralized, reflective information),
  • supporting advanced (context-aware) services,
  • leveraging the Web browser for client-side integration,
  • (hopefully) introducing standardized domain-specific interfaces.
Summary (cont’d)

• Some problems still unresolved such as the handling of non-functional requirements.
• Interoperability between different technologies possible in theory (SOAP, UDDI, WSDL). In practice, vendor still matters.
• Nonetheless, get ready for the Agile Web.
• Before you start: Mind the architectural and engineering issues. Read our pattern books 😊

Our Offer

• We are experienced architects and developers developing solutions for networked systems using standard object-oriented middleware such as
  • CORBA 3, COM/COM+, Microsoft .NET, Java 2
  • Relational and object-oriented databases
  • XML
  • Web Services
• We can offer professional services such as:
  • Development Support
  • Consulting, Mentoring
  • Architecture Reviews
  • Training
• We invite you to visit our booth for further information

We empower your Network
Management Summary

- **According to Sun (White Paper on Sun Open Net Environment Software Architecture):**
  - A **Web service** represents a unit of business, application, or system functionality that can be accessed over the Web (Internet, Intranet, or Extranet)
  - Web services focus on business-to-consumer, business-to-business, department-to-department, peer-to-peer interactions.
  - Consumers of these services may be humans or application programs or other Web services.
  - Web services reveal the following characteristics:
    - they are accessible over the Web
    - they expose an XML interface
    - they can be registered and located through a Web service registry
    - they communicate using XML messages over standard Web protocols
    - they support loosely-coupled connections between systems
  - In theory, it doesn’t matter what technologies are used to build Web services, because all Web service environments can interoperate using XML protocols.