Web Services

SOAP (**Simple Object Access Protocol**, em português **Protocolo Simples de Acesso a Objetos**) é um protocolo para troca de informações estruturadas em uma plataforma descentralizada e distribuída. Ele se baseia na Linguagem de Marcação Extensível (XML) para seu formato de mensagem, e normalmente baseia-se em outros protocolos da camada de aplicação, mais notavelmente em chamada de procedimento remoto (RPC) e Protocolo de transferência de hipertexto (HTTP), para negociação e transmissão de mensagens.

SOAP pode formar a camada base de uma pilha de protocolos de serviços Web, fornecendo um arcabouço básico de mensagens sob o qual se podem construir os serviços Web.

Este protocolo baseado em XML consiste de três partes: um envelope, que define o que está na mensagem e como processá-la, um conjunto de regras codificadas para expressar instâncias do tipos de dados definidos na aplicação e uma convenção para representar chamadas de procedimentos e respostas.

Sua especificação define um arcabouço que provê maneiras para se construir mensagens que podem trafegar através de diversos protocolos e que foi especificado de forma a ser independente de qualquer modelo de programação ou outra implementação específica. Por não se tratar de um protocolo de acesso a objetos, o acrônimo não é mais utilizado.

Geralmente servidores SOAP são implementados utilizando-se servidores HTTP, embora isto não seja uma restrição para funcionamento do protocolo. As mensagens SOAP são documentos XML que aderem a uma especificação W3C.

O primeiro esforço do desenvolvimento do SOAP foi implementar RPCs sobre XML.

Definição

Envelope das mensagens, regras de codificação, convenção RPC, ligação com protocolos subjacentes.

O SOAP tem:

Mecanismo para definir a unidade de comunicação,

Mecanismo para lidar com erros,

Mecanismo de extensão que permite evolução,

Mecanismo entre as mensagens SOAP e o HTTP, que permite representar tipos de dados em XML.

Concepção

Simplicidade, independente de vendedor, independente da linguagem, independente do modelo de objetos, independente do transporte.

Introdução às tecnologias Web Services: SOA, SOAP, WSDL e UDDI

Antes de nos aprofundarmos nos conceitos e tecnologia de web services, vejamos um pouco sua evolução. No ano de 2000, a W3C (World Wide Web Consortium) aceitou a submissão do Simple Object Access Protocol (SOAP).

Este formato de mensagens baseado em XML estabeleceu uma estrutura de transmissão para comunicação entre aplicações (ou entre serviços) via HTTP. Sendo uma tecnologia não amarrada a fornecedor, o SOAP disponibilizou uma alternativa atrativa em relação aos protocolos proprietários tradicionais, tais como CORBA e DCOM.

No decorrer do ano seguinte, o W3C publicou a especificação WSDL. Uma nova implementação do XML, este padrão forneceu uma linguagem para descrever a interface dos web services. Posteriormente suplementada pela especificação UDDI (Universal Description, Discovery and Integration), que proporcionou um mecanismo padrão para a descoberta dinâmica (dynamic discovering) de descrições

de serviço, a primeira geração da plataforma de Web services foi estabelecida. A **Figura 1** ilustra em alto nível o relacionamento entre estes padrões.

Figura 1: O relacionamento entre especificações de primeira geração

Is accessed using: é acessado utilizando;

Enables discovery of: permite a descoberta de;

Describes: descreve;

Enables communication between: permite a comunicação entre;

Binds to: ligação para.

Desde então, os web services foram adotados por vendedores e fabricantes num ritmo considerável. Suporte amplo da indústria seguiu-se à popularidade e importância desta plataforma e de princípios de projeto orientados a serviço. Isto levou à criação de uma segunda geração de especificação de Web services.

Web services e a arquitetura orientada a serviços (SOA)

Entendendo serviços

O conceito de serviços em uma aplicação existe faz algum tempo. Serviços, assim como componentes, são considerados blocos de construção independentes, os quais coletivamente representam um ambiente de aplicação.

No entanto, diferente de componentes tradicionais, serviços têm algumas características únicas que lhes permitem participar como parte de uma arquitetura orientada a serviços.

Uma destas características é a completa autonomia em relação a outros serviços. Isto significa que cada serviço é responsável por seu próprio domínio, o que tipicamente significa limitar seu alcance para uma função de negócio específica (ou um grupo de funções relacionadas).

Este enfoque de projeto resulta na criação de unidades isoladas de funcionalidades de negócio ligadas fracamente entre si. Isto é possível por causa da definição de uma estrutura padrão de comunicação.

Devido à independência que esses serviços desfrutam dentro desta estrutura, a lógica de programação que encapsulam não tem necessidade de obedecer a nenhuma outra plataforma ou conjunto de tecnologias.

XML Web services

O tipo de serviço mais largamente aceito e bem-sucedido é o XML Web service, que será daqui em diante chamado apenas de web service, ou simplesmente service. Este tipo de serviço possui dois requisitos fundamentais:

Comunica-se via protocolos internet (normalmente HTTP);

Envia e recebe dados formatados como documentos XML.

A ampla aceitação do web service resultou no surgimento de um conjunto de tecnologias suplementares que se tornaram um padrão de fato. Assim ao desenvolver nossa web services devemos considerar o uso de tecnologias que:

Forneça uma descrição de serviço que, no mínimo, consista de um documento WSDL;

Seja capaz de transportar documentos XML utilizando SOAP sobre HTTP.

Estas tecnologias não modificam a funcionalidade do núcleo de um serviço web, tanto como o faz sua habilidade para se representar e comunicar num modo padrão.

Muitas das convenções de arquitetura expressadas neste artigo assumem que SOAP e WSDL fazem parte da estrutura de web services descrita.

Além disto, é normal que um Web service seja:

Capaz de agir como o solicitante e o provedor de um serviço;

Registrado com um discovery agent através do qual possam ser localizados.

Numa conversação típica com um web service, o cliente iniciador do pedido é um web service também. Como mostrado na **Figura 2**, qualquer interface exposta por este client service também o qualifica como um serviço a partir do qual outros serviços podem solicitar informação. Posto isto, web services não se encaixam no modelo clássico de cliente-servidor. Na verdade, eles tendem a estabelecer um sistema ponto-a-ponto, onde cada serviço pode atuar como cliente ou servidor.

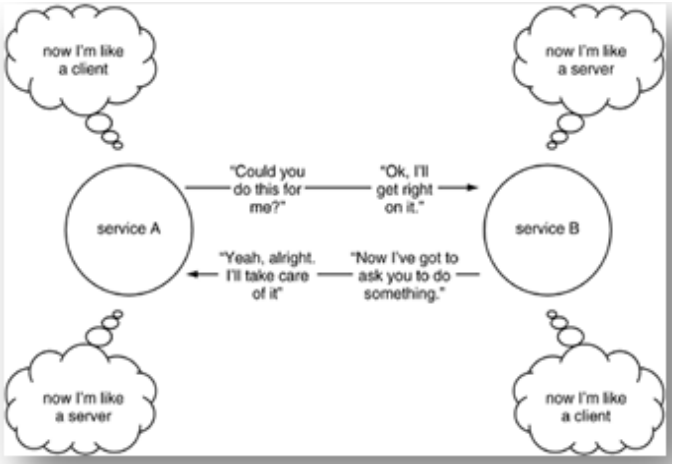

Figura 2: Troca de papéis do Web services durante uma conversação

Now I´m like a client: agora sou um cliente;

Now I´m like a server: agora sou um servidor;

Service: serviços;

Could you do this for me: poderia fazer isso por mim?;

Ok, i´ll get right on it: Ok;

Now I´ve got to ask you to do something: agora eu gostaria de lhe pedir algo;

Yeah, alright. I´ll take care of if: Tudo bem, é só falar.

Service-oriented architecture (SOA)

Como mencionado anteriormente, adicionar uma aplicação com uns poucos web services não é nenhum problema.

Esta integração limitada pode ser apropriada para uma experiência de aprendizado, ou para complementar a arquitetura de uma aplicação existente com uma peça de funcionalidade baseada em serviços que atende a um requisito específico do projeto. No entanto, isto não estabelece uma arquitetura orientada a serviço. Existe uma clara diferença entre:

Uma aplicação que usa web service;

Uma aplicação baseada numa arquitetura orientada a serviços.

Uma SOA é um modelo de projeto com um conceito profundamente amarrado à questão do encapsulamento de aplicação.

A arquitetura resultante estabelece essencialmente um paradigma de projeto, no qual web services são os blocos de construção chave. Isto quer dizer que ao migrar a arquitetura da sua aplicação para uma SOA, estabelece-se um compromisso com os princípios de projeto de web services e a tecnologia correspondente, como partes fundamentais do seu ambiente técnico.

Uma SOA baseada em XML web service é construída sobre camadas de tecnologia XML estabelecidas, focada em expor a lógica de aplicação existente como um serviço fracamente acoplado. Para apoiar este modelo, uma SOA promove o uso de um mecanismo de discovery por serviços via um service broker ou discovery agent.

A **Figura 3** mostra como uma SOA altera a arquitetura multicamada existente, ao introduzir uma camada lógica que, através do uso de interfaces programáticas padrão (providas pelo web services), estabelece um ponto comum de integração.

Esta camada de integração de serviços constitui a base para um novo modelo que pode se estender além do escopo de uma única aplicação, unificando plataformas legadas díspares em um ambiente aberto. Quando web services são utilizados para integração cruzada de aplicações (ver **Figura 4**), elas se estabelecem como parte da infra-estrutura do sistema.

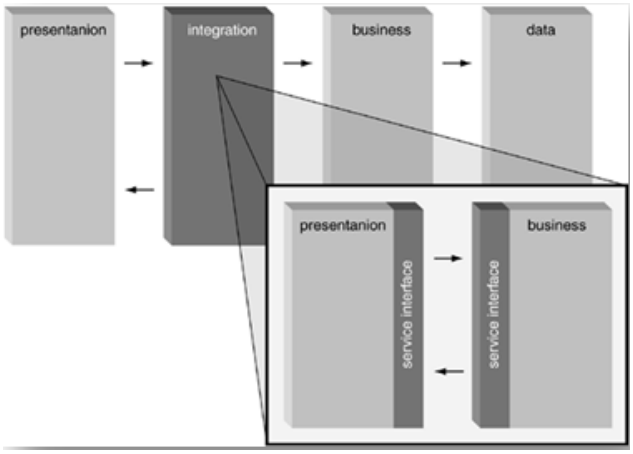

Figura 3: Uma representação lógica de uma arquitetura orientada a serviços

Presentation: apresentação;

Integration: integração;

Business: negócio;

Data: dado;

Service interface: interface de serviço.

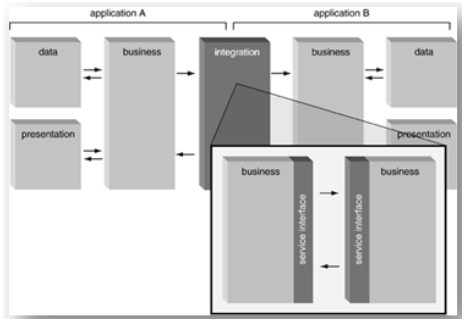

Figura 4: Uma representação lógica de uma arquitetura de integração orientada a serviço

É importante se conscientizar quanto ao acréscimo de complexidade de projeto introduzido pelo SOA. Mais ainda do que em um ambiente n-camada, projetistas de aplicação devem considerar de uma forma completa como a introdução de serviços vai afetar dados existentes e modelos de negócio.

Na medida em que a utilização de serviços se diversifica, o significado dos requisitos de segurança e escalabilidade são amplificados. Ambientes orientados a serviço bem projetados tentarão vencer estes desafios com infra-estrutura adequada, ao invés de utilizar soluções sob medida, específicas de aplicação

Os papéis e cenários ilustrados nas próximas duas seções estão limitados somente ao assunto web service. A estrutura de mensagens SOAP subjacente será explicada em separado, no próximo artigo desta série.

Papéis Web Service

Serviços podem assumir diferentes papéis quando envolvidos em diversos cenários de interação. Dependendo do contexto pelo qual é visualizado, assim como o estado da tarefa rodando no momento, o mesmo web service pode trocar de papéis ou ser designado para múltiplos papéis simultâneos:

Provedor de Serviços

Agindo como um provedor de serviços, um web service expõe uma interface pública através da qual pode ser chamado por solicitantes do serviço. Um provedor de serviços disponibiliza esta interface publicando uma descrição do serviço. Num modelo cliente-servidor, o provedor de serviço pode ser comparado ao servidor.

O termo “provedor de serviço” pode também ser usado para descrever a organização ou ambiente que hospeda (provê) o web service.

Um provedor de serviço pode também agir como um solicitante de serviço. Por exemplo, um web service pode atuar como um provedor de serviço quando um solicitante de serviço lhe pede para executar uma função.

Pode então atuar como um solicitante de serviço quando mais tarde contata o solicitante de serviço original (agora agindo como um provedor de serviço) para solicitar informação de status.

Solicitante de Serviço

Um solicitante de serviço é o remetente de uma mensagem web service ou o programa de software solicitando uma web service específico. O solicitante de serviço é comparável ao cliente dentro de um modelo cliente-servidor padrão. Solicitantes de serviços são às vezes chamados de consumidores de serviços.

Um solicitante de serviço pode também ser um provedor de serviço. Por exemplo, num modelo de solicitação e resposta, a web service iniciador primeiro age como um solicitante de serviço ao requerer informações do provedor de serviço.

A mesma web service então, faz o papel de um provedor de serviço ao responder à solicitação original.

Intermediário

O papel de intermediário é assumido pelo web service quando ele recebe a mensagem de um solicitante de serviço e a passa adiante para o provedor de serviço. Neste caso, ele pode também agir como um provedor de serviço (recebendo a mensagem) e como um solicitante de serviço (passando adiante a mensagem).

Intermediários podem existir em muitas formas diferentes. Alguns são passivos e simplesmente retransmitem ou roteam as mensagens, enquanto outros processam ativamente uma mensagem antes de repassá-la.

Tipicamente, aos intermediários só é permitido o processamento e modificação do cabeçalho da mensagem. Para preservar a integridade da mensagem, seus dados não devem ser alterados.

Remetente Inicial

Como o web service responsável por iniciar a transmissão da mensagem, remetentes iniciais também podem ser considerados solicitantes de serviço. Este termo existe para ajudar a diferenciar o primeiro web service que envia uma mensagem, dos intermediários também qualificados como solicitantes de serviço.

Receptor Final

O último Web service a receber uma mensagem é o receptor final. Estes serviços representam o destino final de uma mensagem e também podem ser considerados provedores de serviço.

Interação Web Service

Quando mensagens são passadas entre dois ou mais web services, uma variedade de cenários de interação pode acontecer. A seguir, termos comuns utilizados para identificar e etiquetar estes cenários serão apresentados.

Caminho da Mensagem

A rota pela qual a mensagem viaja é o caminho da mensagem. Deve consistir de um remetente inicial e um receptor final e pode conter nenhum, um, ou mais de um intermediários. A **Figura 5** ilustra um caminho de mensagem simples.

O caminho de transmissão atual percorrido por uma mensagem pode ser dinamicamente determinado por roteadores intermediários.

A lógica de roteamento pode ser ativada em resposta à carga de requisitos de balanceamento, ou pode ser baseada nas características da mensagem e outras variáveis lidas e processadas pelo intermediário em tempo de execução. A **Figura 6** descreve como uma mensagem é enviada via um ou dois caminhos de mensagem possíveis, tal como é determinado pelo roteador intermediário.

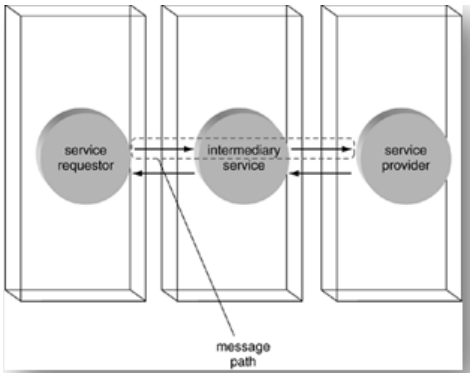

Figura 5: Um caminho de mensagem formado por três Web services

Message path: caminho da mensagem.

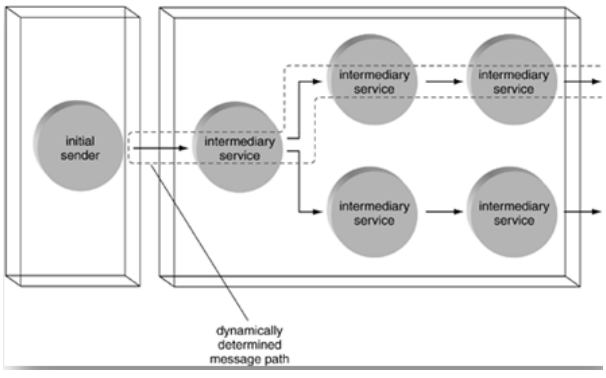

Figura 6: Uma mensagem dinamicamente determinada por um roteador intermediário

Dynamically determined message path: caminho da mensagem determinada dinamicamente.

Padrão de troca de mensagens

Serviços que interagem em um ambiente orientado a serviços tipicamente se enquadram em determinados padrões de troca de mensagens. Padrões típicos incluem:

Solicite e responda (request and response);

Publique e subscreva (publish and subscribe);

fire and forget - um para um;

fire and forget - um para muitos ou difusão;

O padrão pedido e resposta é o mais comum quando se está simulando intercâmbio de dados síncronizados. Os demais padrões são usados principalmente para facilitar transferência de dados assíncrona.

Correlação

Correlação (Correlation) é a técnica utilizada para casar mensagens enviadas através de caminhos de mensagem diferentes. É comumente empregada num padrão de intercâmbio de pedido e resposta de mensagem, onde a mensagem de resposta deve estar associada à mensagem original que iniciou a solicitação. Embutir valores de ID sincronizados dentro de mensagens relacionadas é uma técnica frequentemente utilizada para conseguir correlação.

Coreografia

Regras que governam características de comportamento relacionadas à forma como um grupo de web services interagem podem ser aplicadas como uma coreografia (choreography). Estas regras incluem a seqüência na qual web services podem ser chamados, condições que se aplicam à seqüência que está sendo transportada e o padrão de uso que irá definir os cenários de interação permitidos. O escopo de uma coreografia está tipicamente amarrado ao de uma atividade ou tarefa (ver exemplo na **Figura 7**).

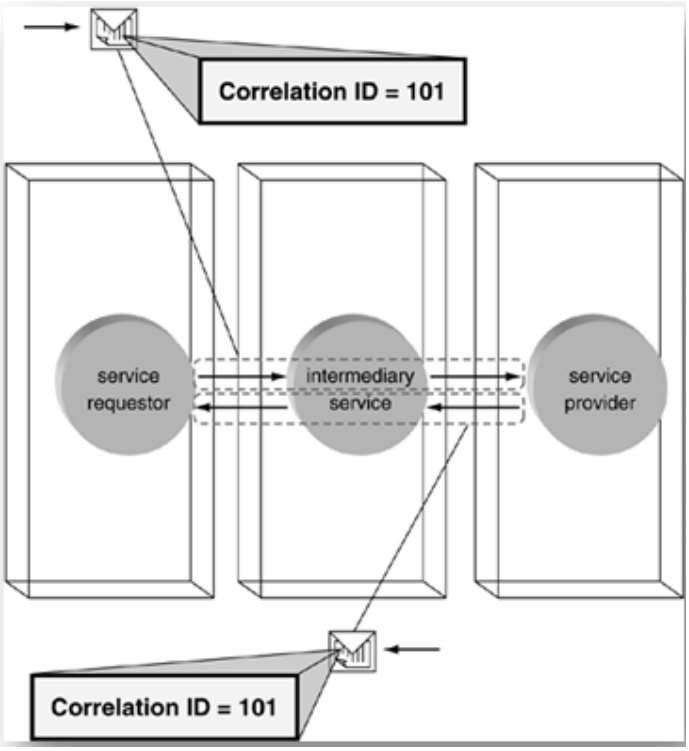

Figura 7: A seqüência de interação de um grupo de serviços sendo comandados por uma coreografia

Correlation: correlação.

Atividade

Padrões de intercâmbio de mensagens formam a base para atividades de serviços (também conhecidos como tarefas). Uma atividade consiste de um grupo de web services que interagem e colaboram para realizar uma função ou um grupo lógico de funções.

A **Figura 8** mostra uma atividade de serviço simples. A diferença entre uma coreografia e uma atividade está no fato de que a atividade é geralmente associada com uma função de aplicação específica, tal como a execução de uma tarefa de negócio.

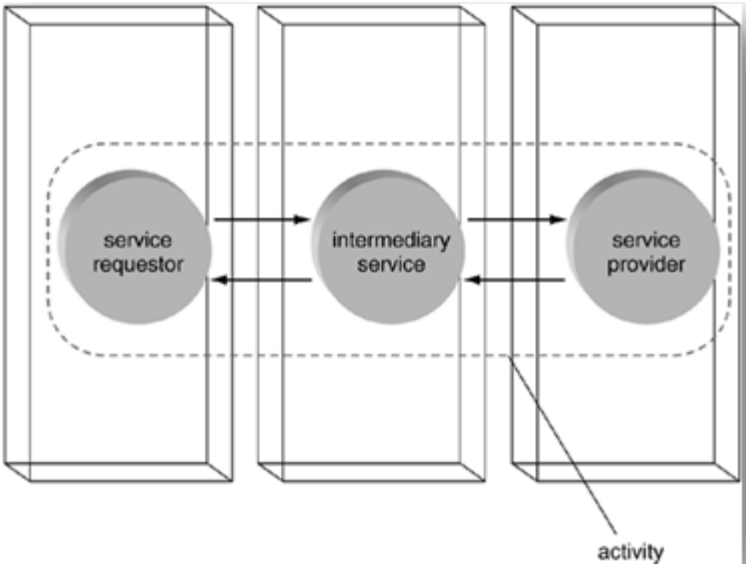

Figura 8: Uma atividade de serviço envolvendo três serviços

Activity: atividade.

Descrição da estrutura de web services

Um web service é descrito através de uma coleção de documentos de definição. Estes atuam como blocos de construção para uma descrição de serviço:

Abstrato + Concreto = Definição de Serviço;

Definição de Serviço + Definições Complementares = Descrição de Serviço.

A **Figura 9** ilustra a relação entre esses documentos.

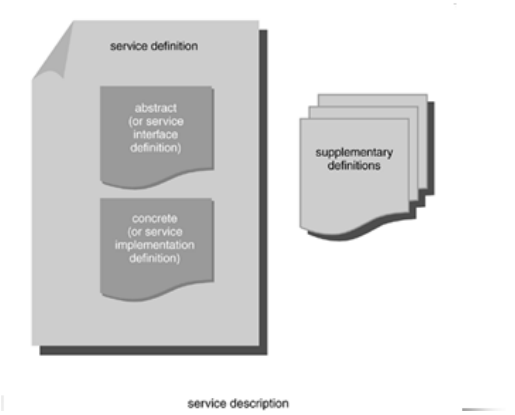

Figura 9: Conteúdo de uma descrição de serviço

Service description: descrição do serviço;

Service definifion: definição do serviço;

Supplementary definitions: definições suplementares;

Abstract: abstrato;

Concret: concreto;

service interface definition: definição da interface do serviço;

service implementation definition: definição da implementação do serviço.

Abstrato

A descrição de uma interface web service, independente dos detalhes de implementação, é chamada de abstrato (abstract). Descrições deste elemento são fornecidas adiante neste artigo, como parte do tutorial do WSDL.

Concreto

Localização e informação de implementação específicas sobre um web service constituem as partes concretas (concrete) de um documento WSDL. Elas são representadas pelos elementos de ligação (binding), serviço (service) e ponto-de-término (endpoint ou port).

Definição de serviço

Geralmente, o conteúdo de um documento WSDL constitui uma definição de serviço (service definition) que inclui as definições da interface (abstrato) e da implementação (concreto).

Descrição de Serviço

Frequentemente, uma descrição de serviço é um único documento WSDL que fornece uma definição de serviço. No entanto, pode também incluir vários documentos de definição adicionais que irão fornecer informações complementares.

Introdução Aos Web Services de Primeira Geração

A estrutura W3C para web services está fundamentada em três especificações XML fundamentais:

Linguagem para definição de web service (Web Services Definition Language - WSDL);

Simple Object Access Protocol (SOAP);

Universal Description, Discovery, and Integration (UDDI).

Estes padrões de tecnologia, acoplados aos princípios de projeto orientado a serviço, formam um SOA fundamentado na tecnologia XML. Esta arquitetura de web services de primeira geração permite a criação de web services independentes capazes de encapsular unidades isoladas de funcionalidades de negócio.

Esta tecnologia tem também algumas limitações, que tem sido contempladas numa segunda geração de especificações. A seção a seguir, fornece um tutorial introdutório para a tecnologia WSDL. SOAP e UDDI serão vistos no próximo artigo da série.

Linguagem para definição de Web Services (WSDL)

Web services devem ser definidos numa forma consistente para que possam ser descobertos e “interfaceados” com outros serviços e aplicações. A WSDL é uma especificação W3C que fornece a linguagem mais avançada para a descrição de definições de web services.

A camada de integração introduzida pela estrutura de web services estabelece um padrão, universalmente reconhecido e com interface programática suportada. Tal como mostrado na **Figura 10**, WSDL permite a comunicação entre essas camadas ao fornecer descrições padronizadas.

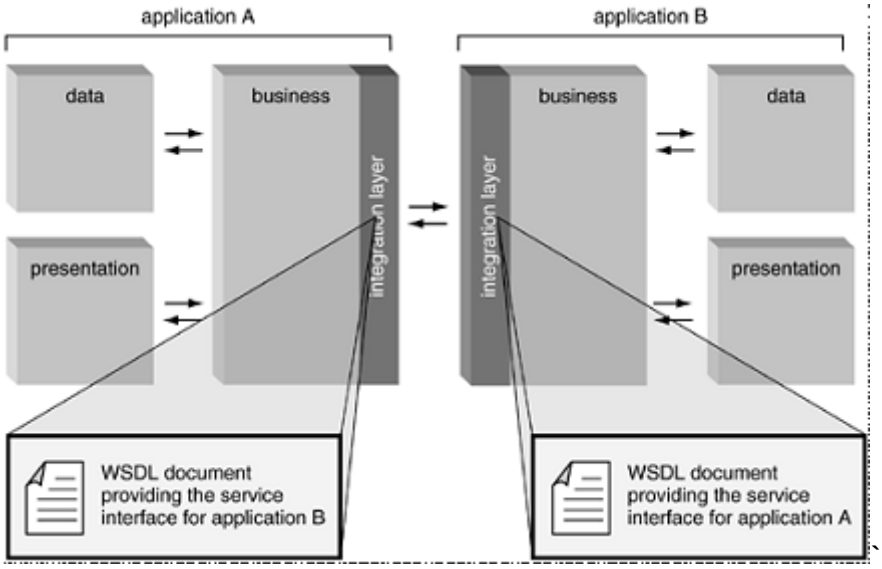

Figura 10: Documentos WSDL representando aplicações web services

Application: aplicação;

Integration layer: camada de integração;

WSDL document providing the service interface for application b: documento WSDLl provendo a interface para o serviço da aplicação b;

WSDL document providing the service interface for application a: documento WSDL provendo a interface para o serviço da aplicação a.

A melhor forma de entender como é definido um web service e como ele é expresso por um documento WSDL, é caminhar através de cada construtor que coletivamente representa essa definição. Comecemos com o elemento definitions raiz, o qual age como o conteiner para a definição do serviço (ver **Listagem 1**).

Listagem 1: Uma definição de serviço, tal como é expressa pelo construtor definitions

```
<definitions>
        <interface name="Catalog">
        ...
        </interface>
        <message name="BookInfo">
        ...
        </message>
        <service>
        ...
        </service>
```

```
        <binding name="Binding1">
        ...
        </binding>
</definitions>
```

Uma definição WSDL pode conter coleções dos seguintes construtores primários:

Interface;

Message;

Service;

Binding.

A **Figura 11** ilustra como os primeiros dois construtores representam a definição da interface de serviço e os últimos dois fornecem os detalhes de implementação do serviço.

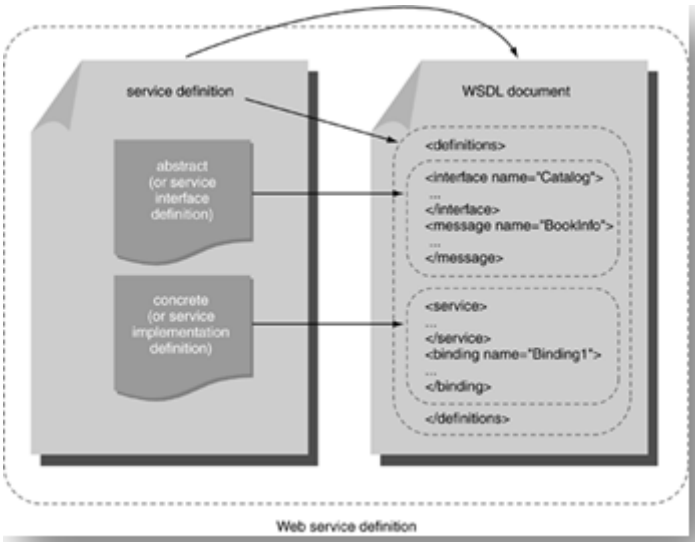

Figura 11: O conteúdo de um documento WSDL, tal como se relaciona com uma definição de serviço

Web service definition: definição do web service;

WSDL document: documento WSDL.

Definição de interface abstrata

Interfaces de web services individuais são representadas por elementos interface WSDL. Estes construtores contêm um grupo de operações lógicas correlatas. Numa arquitetura baseada em componentes, um elemento interface WSDL é análogo à interface do componente. Uma operação, portanto, é equivalente a um método de componente, por representar uma única ação ou função. A **Listagem 2** apresenta um exemplo de interface.

Listagem 2: Uma interface representada pelo elemento interface

```
<definitions>
        <interface name="Catalog">
                <operation name="GetBook">
                ...
```

```
            </operation>
        </interface>
</definitions>
```

Um elemento operation típico consiste de um grupo de mensagens de entrada e saída correlatas. A execução de uma operation requer a transmissão ou intercâmbio destas mensagens entre o serviço solicitante e o provedor de serviço.

Mensagens operation são representadas por construtores message que são declarados sob os elementos definitions. Os nomes das mensagens são então referenciados nos elementos filho das operation input ou output (ver **Listagem 3**).

Listagem 3: O elemento input dentro do construtor operation referenciando um bloco de mensagem

```
<definitions>
        <message name="BookInfo">
        ...
        </message>
        <interface name="Catalog">
            <operation name="GetBook">
                <input name="Msg1" message="BookInfo"/>
            </operation>
        </interface>
</definitions>
```

Um elemento message pode conter um ou mais parâmetros input ou output que pertencem a uma operation. Cada elemento part define um destes parâmetros. Ele fornece um conjunto nome/valor (name/value), junto com o tipo de dado associado. Em uma arquitetura baseada em componentes, um part WSDL equivale a um parâmetro de input ou output (ou um valor de retorno) de um método de componente (ver **Listagem 4**).

Listagem 4: Um bloco de mensagem com um construtor part representando parâmetros operation

```
<definitions>
        <message name="BookInfo">
            <part name="title" type="xs:string">
                Field Guide
            </part>
            <part name="author" type="xs:string">
                Mr. T
            </part>
        </message>
</definitions>
```

A seguir, um breve resumo dos construtores fundamentais que podem ser montados para estabelecer uma definição de interface abstrata:

Interfaces: representam interfaces de serviço e podem conter múltiplos operations;

Operations: representam uma função web service e podem referenciar múltiplas messages;

Messages: representam uma coleção de parâmetros input ou output e podem conter múltiplas parts;

Parts: representam parâmetros de dados operation tanto de chegada como de partida.

Definição concrete (implementação)

Sobre os detalhes de implementação, usando os elementos descritos nesta seção, um documento WSDL pode estabelecer detalhes “concrete” de ligação para protocolos, tais como SOAP e HTTP.

Dentro de um documento WSDL, o elemento service representa um ou mais pontos-de-término nos quais o web service pode ser acessado.

Estes pontos-de-têrmino consistem de informações de localização e protocolo e são armazenados numa coleção de elementos endpoint (ver **Listagem 5**).

Listagem 5: O elemento ponto-de-término

```
<definitions>
        <service name="Service1">
                <endpoint name="EndPoint1" binding="Binding1">
                ...concrete implementation details...
                </endpoint>
        </service>
</definitions>
```

Agora que descrevemos como um web service pode ser acessado, precisamos definir os requisitos de chamada para cada uma das suas operations.

Os elementos binding associam às operations informações de formato de protocolo e mensagem. O construtor operation que reside dentro do bloco binding é semelhante a sua contrapartida na seção interface (ver **Listagem 6**).

Listagem 6: O elemento binding representando uma operation existente

```
<definitions>
        <service name="Service1">
                <binding name="Binding1">
                        <operation>
                                <input name="Msg1" message="book"/>
                        </operation>
                </binding>
        </service>
</definitions>
```

A descrição de informação concrete dentro de um documento WSDL pode ser resumida assim:

elementos service: hospedam coleções de ponto-de-término representados individualmente por elementos endpoint;

elementos endpoint: contem dados endpoint, incluindo endereço físico e informação de protocolo;

elementos binding: se auto-associam a construtores operation;

cada endpoint pode referenciar um elemento binding e, portanto, fornecer informação endpoint para a operation subordinada.

Vimos nesse primeiro artigo o princípio de funcionamento dos web services bem como duas de suas tecnologias de base, SOA e WSDL. No próximo artigo, complementaremos o assunto apresentado aqui analisando as tecnologias SOAP e UDDI. Até lá

REST e SOAP: Usar um dos dois ou ambos?

Desenvolvedores web têm uma grande quantidade de tecnologias que podem escolher, de ferramentas para acesso simples a bancos de dados, integração com serviços em middleware, a softwares do lado do cliente.

A quantidade de opções em si já é um desafio, e escolher uma abordagem específica para construir partes de uma aplicativa web exacerba o problema.

Neste breve artigo, vamos nos concentrar em uma dessas escolhas: SOAP ou REST. Ambas possuem vantagens e desvantagens e fica na mão do desenvolvedor determinar a melhor abordagem para cada caso em particular.

A maioria dos desenvolvedores tem exposto seus serviços utilizando REST, que faz uso de um padrão de URI (Uniform Resource Identifier), fazendo uma chamada para um serviço web como em:

O REST é simples de entender e pode ser adotado em praticamente qualquer cliente ou servidor com suporte a HTTP/HTTPS. Os desenvolvedores que o utilizam citam, como principais vantagens a facilidade no desenvolvimento, o aproveitamento da infraestrutura web existente e um esforço de aprendizado pequeno.

Por outro lado, o SOAP, avô das interfaces de serviços web, não deixará de ser usado tão cedo. Com o SOAP v 1.2, muitas das deficiências percebidas nessa tecnologia foram corrigidas e aumentou a facilidade de uso.

Além disso, a sigla SOAP deixou de representar "Simple Object Access Protocol". Na especificação 1.2 da W3C, SOAP é apenas o nome da especificação.

Utilizar o SOAP 1.2 traz uma carga adicional não encontrada ao usar REST, mas há também vantagens.

Primeiramente o SOAP é baseado em XML, de três formas: o envelope, que define o conteúdo da mensagem e informa como processá-la; um conjunto de regras de codificação para os tipos de dados; e o layout para os procedimentos de chamadas e respostas.

Esse "envelope" é enviado por meio de (por exemplo) HTTP/HTTPS. E uma RPC (Remote Procedure Call) é executada, e o envelope retorna com as informações do documento XML formatado.

Uma das vantagens do SOAP é o uso de um método de transporte "genérico". Enquanto que o REST faz uso de HTTP/HTTPS, o SOAP pode usar qualquer meio de transporte existente para enviar sua requisição, desde SMTP até mesmo JMS (Java Messaging Service).

No entanto, uma desvantagem percebida no uso de XML é a sua natureza prolixa e o tempo necessário para analisar o resultado apresentado.

A boa notícia para os desenvolvedores web é que ambas as tecnologias são muito viáveis no mercado atual. Ambos REST e o SOAP conseguem resolver um grande número de problemas e desafios

na web, e em muitos casos tanto um como o outro podem ser utilizados para fazer o que querem os desenvolvedores.

Mas uma história não contada é que ambas as tecnologias podem ser misturadas e combinadas. O REST é fácil de entender e extremamente acessível, porém faltam padrões, e a tecnologia é considerada apenas uma abordagem arquitetural. Em comparação, o SOAP é um padrão da indústria, com protocolos bem definidos e um conjunto de regras bem estabelecidas.

Pode-se afirmar, então, que casos onde o REST funciona bem são:

Situações em que há limitação de recursos e de largura de banda: A estrutura de retorno é em qualquer formato definido pelo desenvolvedor e qualquer navegador pode ser usado. Isso porque a abordagem REST usa o padrão de chamadas GET, PUT, POST e DELETE. O REST também pode usar objetos XMLHttpRequest (a base do velho AJAX) que a maioria dos navegadores modernos suporta.

Operações totalmente sem-estado: se uma operação precisa ser continuada, o REST não será a melhor opção. No entanto, se forem necessárias operações de CRUD stateless (Criar, Ler, Atualizar e Excluir), o REST seria a melhor alternativa.

Situações que exigem cache: se a informação pode ser armazenada em cache, devido à natureza da operação stateless do REST, esse seria um cenário adequado para a tecnologia.

Essas três situações abrangem muitas soluções. Então por que ainda precisamos considerar o uso do SOAP? Mais uma vez, o SOAP é bastante maduro e bem definido e vem com uma especificação completa. Já a abordagem REST é apenas isso: uma abordagem. Está totalmente aberta. Por isso ao se encontrar uma das situações abaixo, o SOAP pode ser uma ótima solução:

Processamento e chamada assíncronos: se o aplicativo precisa de um nível garantido de confiabilidade e segurança para a troca de mensagens, então o SOAP 1.2 oferece padrões adicionais para esse tipo de operação como por exemplo o WSRM (WS-Reliable Messaging).

Contratos formais: se ambos os lados (fornecedor e consumidor) têm que concordar com o formato de intercâmbio de dados, então o SOAP 1.2 fornece especificações rígidas para esse tipo de interação.

Operações stateful: para o caso de o aplicativo precisar de informação contextual e gerenciamento de estado com coordenação e segurança, o SOAP 1.2 possui uma especificação adicional em sua estrutura que apoia essa necessidade (segurança, transações, coordenação etc.). Comparativamente, usar o REST exigiria que os desenvolvedores construíssem uma solução personalizada.

Como se vê, cada uma das abordagens tem sua utilidade. Ambas têm problemas nos quesitos de segurança, camadas de transporte etc.; mas ambas podem realizar o trabalho necessário e trazem sua contribuição para o desenvolvimento de aplicações web.

Portanto, a melhor abordagem é a flexibilidade, pois não importa qual seja o problema, no mundo de hoje do desenvolvimento web, conta-se com excelentes resultados ao fazer uso de um desses padrões.

Qual a diferença entre REST e SOAP?

Bom, vamos lá.. Quem é que nunca leu algum artigo, presenciou ou de fato iniciou uma discussão sobre REST e SOAP? Acho que muitos... certo?!

Então… Logo após de ler o título deste post você deve estar se perguntando “Esse cara vai ser mais um desses que fica defendendo um lado e detonando o outro com argumentos esquerdistas e radicais?” De ante mão te respondo:

“Não!”. Minha intenção com este post é apenas explicar um pouco sobre o assunto e tentar desfazer algumas confusões que as pessoas fazem com REST, SOAP e outros assuntos adjacentes.

Antes de mais nada, vamos a definição básica destes dois:

REST – Representational State Transfer é um estilo arquitetural usado no projeto de aplicações da Web que contam com recursos nomeados (URL,URI,URN) e engenhosamente utiliza mais profundamente o protocolo HTTP, seu cabeçalho , seus métodos (GET, POST, PUT, DELETE, HEAD) e toda a infraestrutura web já bem estabelecida, reconhecida e utilizada por todos.

SOAP – Simple Object Access Protocol é um protocolo para troca de informações estruturadas geralmente em uma plataforma descentralizada e distribuída. Ele se baseia em XML para seu formato de mensagem, ou seja, uma mensagem SOAP encapsula o conteúdo e pode ser trafegada via HTTP, JMS ou outro protocolo.

Definição de Interfaces

Primeiramente, pensando em serviços que utilizam mensagens no protocolo SOAP, o mais comum é descrevermos a interface do mesmo com WSDL (Web Services Description Language). Basicamente, cada serviço terá um arquivo .wsdl que terá a definição de suas operações, estrutura de dados que são usadas nas requisições e respostas, Endpoints (endereços de rede do serviço).

Uma associação que ajuda o entendimento deste ponto é pensarmos nas operações como métodos de uma classe, a assinatura do método como as mensagens definidas, e o tipo de cada campo da assinatura a definição da estrutura de dados que será usada nas mensagens.

Agora, nos serviços RESTful, se seguirmos de fato o estilo, parte da descrição da interface é desnecessária já que a forma de interagir com os serviços é sempre a mesma, por meio dos métodos http.

E ainda mais com o uso do Método Option podemos saber quais outros métodos são permitidos naquele serviço. Agora com relação aos dados continua a ser interessante a definição de que dados vão trafegar e suas restrições.

Agora, tem pessoas que mesmo assim preferem de uma definição tanto dados quanto de operações, e estas podem fazer o seguinte:

a) Criar um documento para ser lido por humanos que pode conter, por exemplo, a estrutura de dados que seu serviço recebe e responde, lista de quais operações HTTP estão sendo suportadas naquele serviço, qual formato de mensagem é suportado, etc.

Na minha opinião, esta abordagem faz mais sentido quando a estrutura de dados da mensagem é simples e/ou utiliza notações simples como JSON por exemplo, devido a facilidade da implementação do código que irá acessar ao serviço.

b) A segunda forma, é utilizar o WADL(Web Application Description Language), ele é similar ao WSDL só que mais simples, com artibutos de configuração que facilitam a compreensão, por exemplo, de qual notação/formato é utilizada nas mensagens de request e response, e o mais importante, é todo orientado aos recursos e ao protocolo HTTP.

Obs:

WSDL e WADL serve para que aplicações clientes/consumidoras possam gerar o código automaticamente a partir destas definições. No entanto, nada impede que trasnformações .XSL sejam aplicadas a estes arquivos e eles fiquem mais “legíveis” para leigos, afinal os dois são XML com schema e com vocabulário bem definido.

SOAP e SOA

Preciso começar este tópico com uma frase clássica, “uma coisa é uma coisa outra coisa é outra coisa”. Nem adianta falar que a diferença entre SOAP e SOA é a letra “P” essa já está velha.

Primeiro, SOAP é um protocolo para troca de informações estruturada, como vimos no início deste post. SOA, por sua vez, é uma arquitetura que se baseia no paradigma de orientação a serviço que molda o desenvolvimento de funcionalidades de negócio, buscando desacoplamento, reutilização, produtividade e alinhamento entre objetivos de negócio e as estratégias de TI. (SOA Manifesto)

Além do nome ser parecido, o que faz com que as pessoas troquem um pelo outro em conversas, e leigos que estão escutando multiplicarem esta confusão por aí…, existe também uma questão relacionada tanto aos big vendors de ferramentas, quanto aos consultores tradicionais atrelados à SOA, que focam muito em Web services, WSDL, WS-* Extensions e SOAP.

Ainda tem o fato adicional de que a maioria deles estão engatinhando em REST. Ou seja…, tem muitos profissionais de TI, inclusive que se dizem entendidos de SOA, que acham que para se ter uma Arquitetura Orientada a serviço precisamos de Webservices, WSDLs e SOAP, o que está 100% errado.

Algumas questões que podem desfazer algumas amarras entre estes conceitos

a) Podemos ter uma empresa com uma maturidade elevada em SOA, apenas com serviços RESTful? SIM!

b) Podemos ter serviços RESTful com a interface descrita com WSDL.? SIM! É! com WSDL.. WSDL foi criado para descrever interfaces e tem uma boa flexibilidade, podemos usar SOAP sobre HTTP, ou simplesmente HTTP com mensagens em formato JSON, ou XML. (obs: Apartir do WSDL 2.0 isto ficou mais fácil.)

c) Podemos ter serviços disponíveis em HTTP e não serem serviços REstful? SIM! Formas de utilizar o cabeçalho HTTP, como modelar as URLS, e usar operações GET, POST, PUT, DELETE entre outras coisas vão determinar se o serviço é RESTful ou não.

SOAP ou REST ?

Podemos elencar alguns pontos interessantes:

Rest
– É mais elegante, pois utiliza ao máximo o protocolo HTTP, evitando a construção de protocolos adicionais
– Tem o potencial de ser bem mais simples que uma implementação com WSDL/SOAP
– Tende a ser mais performático
– ~ 80% das integrações utilizam o protocolo HTTP.
– A possibilidade de ter difersças representações de um mesmo recurso, por exemplo, uma dada entidade pode ser representada em diferentes formatos como Json, xml, html e text/plain, dependendo da requisição feita pelo cliente(Content-Negotiation)
– Possibilidade de navegar entre relacionamentos (Links web) de vários recursos de forma dinamica. seguindo a usabilidade de qualquer sistema web. HATEOAS (Hypermedia as the Engine of Application State).

SOAP
– É um padrão que combinado a as especificações WS-* podem garantir questões de QoS(Quality of Service), Segurança, transação e outras questões presentes em integrações mais complexas.
– Uma mensagem SOAP pode ser propagada por diferentes protocolos, o que flexibiliza bastante várias integrações.
– É um padrão que está muito maduro no mercado, qualquer ferramenta de integração e Framework tem várias funcionalidades para manipular as mensagens que seguem este padrão.

Com certeza existem muitos outros pontos a serem levantados, mas de qualquer forma.. Minha resposta para a pergunta tão popular “Qual o melhor?” seria “Depende”.. depende de qual problema você quer resolver.

Particularmente, acredito que cada demanda tem que ser analisada, com calma, por pessoas que conheçam as 2 abordagens e possam chegar a uma solução adequada ao problema em questão. Isto me faz lembrar de um artigo que o Marcelo Fernandes (companheiro de equipe), me encaminhou há algum tempo que falava sobre uma metodolia ágil muito extremista mas que tinha uma frase interessante.. algo como “Tudo aquilo que você considera verdade, está errado em algum contexto” ou seja, achar uma solução antes de entender o problema é algo que pode funcionar algumas vezes, mas tem um risco, e com certeza em algum, ou vários contextos aquilo não irá se aplicar.. então.. analise cada caso e tire proveito de cada um dos dois.

SOAP e WebServices

O SOAP é um protocolo elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante à invocação de páginas Web.

O SOAP é um protocolo elaborado para facilitar a chamada remota de funções via Internet, permitindo que dois programas se comuniquem de uma maneira tecnicamente muito semelhante à invocação de páginas Web.

O protocolo SOAP tem diversas vantagens sobre outras maneiras de chamar funções remotamente como DCOM, CORBA ou diretamente no TCP/IP:

É simples de implementar, testar e usar.

É um padrão da indústria, criado por um consórcio da qual a Microsoft é parte, adotado pela W3C (http://www.w3.org/TR/SOAP/) e por várias outras empresas.

Usa os mesmos padrões da Web para quase tudo: a comunicação é feita via HTTP com pacotes virtualmente idênticos; os protocolos de autenticação e encriptação são os mesmos; a manutenção de estado é feita da mesma forma; é normalmente implementado pelo próprio servidor Web.

Atravessa “firewalls” e roteadores, que “pensam” que é uma comunicação HTTP.

Tanto os dados como as funções são descritas em XML, o que torna o protocolo não apenas fácil de usar como também muito robusto.

É independente do sistema operacional e CPU.

Pode ser usado tanto de forma anônima como com autenticação (nome/senha).

Os pedidos SOAP podem ser feitos em três padrões: GET, POST e SOAP. Os padrões GET e POST são idênticos aos pedidos feitos por navegadores Internet. O SOAP é um padrão semelhante ao POST, mas os pedidos são feitos em XML e permitem recursos mais sofisticados como passar estruturas e arrays. Independente de como seja feito o pedido, as respostas são sempre em XML.

O XML descreve perfeitamente os dados em tempo de execução e evita problemas causados por inadvertidas mudanças nas funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo muito robusto.

O SOAP define também um padrão chamado WSDL, que descreve perfeitamente os objetos e métodos disponíveis, através de páginas XML acessíveis através da Web. A idéia é a seguinte: quem publicar um serviço, cria também estas páginas.

Quem quiser chamar o serviço, pode usar estas páginas como “documentação” de chamada e também usadas antes de chamar as funções para verificar se alguma coisa mudou.

O SOAP pode ser facilmente implementado em virtualmente qualquer ambiente de programação. Existem atualmente diversos “kits” de desenvolvimento SOAP para vários sistemas operacionais e linguagens de alto nível. A própria Microsoft tem um “kit” para o Visual Studio 6 em http://msdn.microsoft.com/soap/default.asp.

O SOAP é uma parte importante da arquitetura .NET da Microsoft e tem um extenso suporte no Visual Studio.NET. Um WebService é um conjunto de métodos WebMethods logicamente associados e chamados através de SOAP.

Os WebMethods são funções chamadas remotamente através de SOAP. Cada WebService tem dois arquivos associados: um com extensão “asmx” e outro com extensão “cs” se você estiver usando a linguagem C#.

Na arquitetura .NET os WebServices são implementados sempre em uma classe derivada de “System.Web.Services.WebService”. Nesta classe adicionamos as funções (métodos) que serão chamados via SOAP. A diferença entre um WebMethod e um método comum é a presença de um “atributo

WebMethod", uma espécie de diretiva de compilação. A página SDL é gerada automaticamente pelas ferramentas de programação.

Do ponto de vista do programador, um WebService é uma página ASP.NET "glorificada", que mapeia automaticamente pedidos via Web a métodos de uma linguagem de alto-nível.

Criando um Web Service

Veja a seguir uma seqüência de criação de um WebService que faz algumas contas simples. Inicialmente criamos um aplicativo WebService no Visual Studio.NET Beta 2:

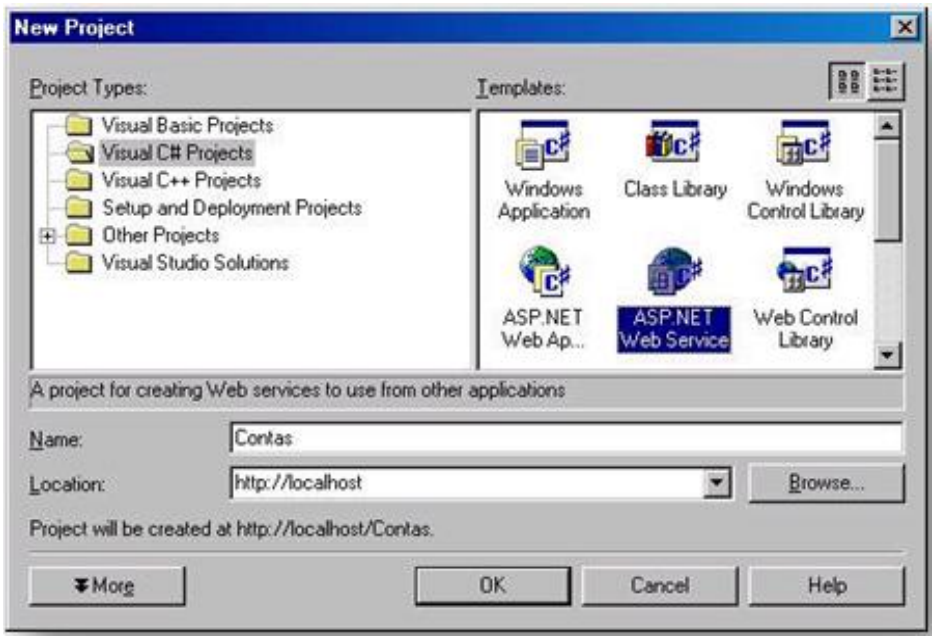

Figura 1: Criando novo web service

Este é um projeto ASP.NET comum. Na prática você provavelmente irá também acrescentar páginas ASP.NET comuns.

A seguir escrevemos o código do serviço:

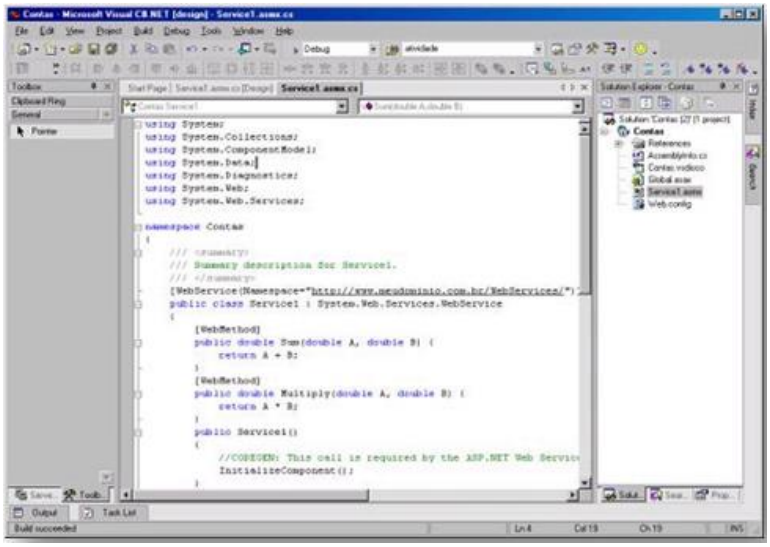

Figura 2: Código no editor

Veja o código digitado:

Listagem 1: Código do serviço

```
1[WebMethod]

2public double Sum(double A, double B) {

3return A + B;

4}

5[WebMethod]

6public double Multiply(double A, double B) {

7return A * B;

8}
```

Note o atributo [WebMethod], que sinaliza ao sistema de runtime que este é um método chamado via HTTP. Nem todos os métodos precisam ser WebMethods.

Todo WebService deve ser identificado de forma única no Universo. A maneira de fazer isto é fornecer uma URI baseada em um domínio Internet registrado por você ou pela sua empresa. Esta URI deve ser fornecida em um atributo antes da declaração da classe:

```
[WebService(Namespace="http://www.meudominio.com.br/WebServices/")]
```

Após pedirmos “Build”, podemos imediatamente acessar o serviço através de um navegador Web. Peça “Debug | Start Without Debugging”. As rotinas de suporte a WebServices irão criar automaticamente uma página para testar o WebService:

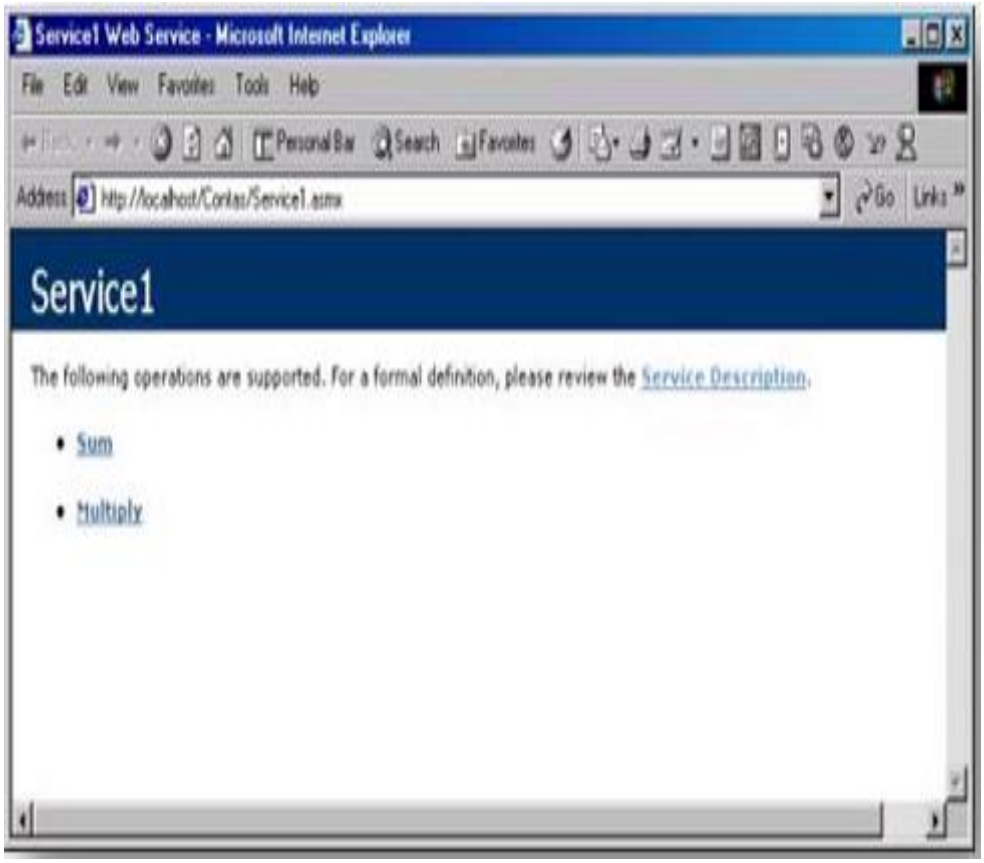

Figura 3: Página de teste do webservice

Selecione algum método, Sum, por exemplo:

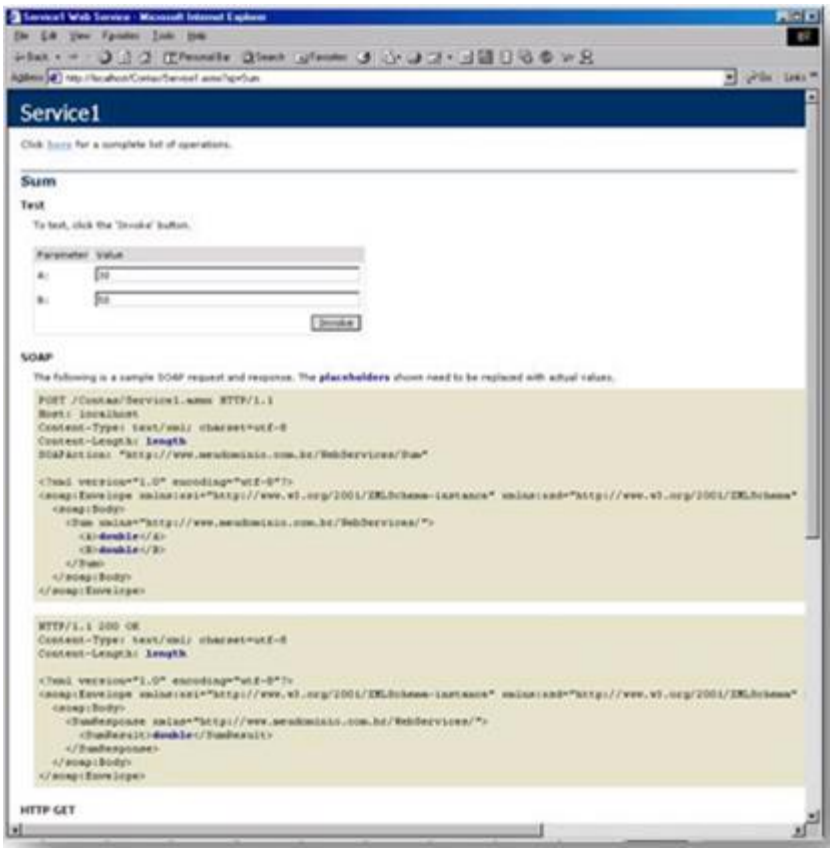

Figura 4: Página de execução do método Sum

Preencha alguns valores e veja a saída do teste:

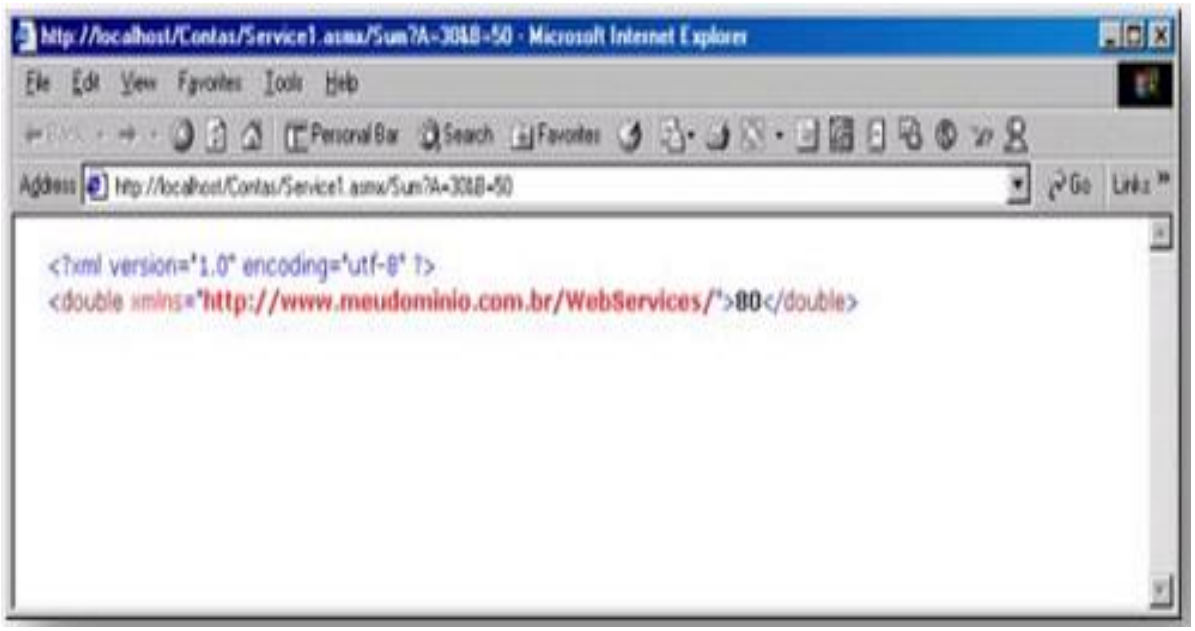

Figura 5: Resultado do método Sum

Podemos interrogar pegar a descrição do serviço como XML através do padrão “WSDL”. Esta é a maneira que será usada pelo Visual Studio.NET para criar automaticamente uma classe “proxy” que chamará o WebService:

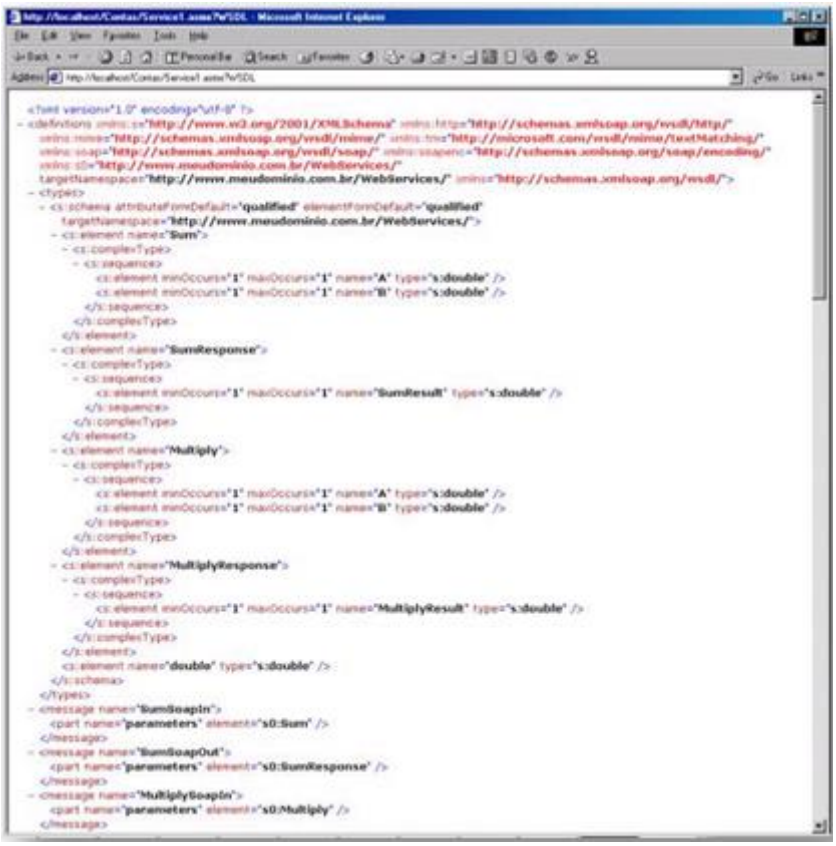

Figura 6: Descrição do serviço como XML

Observe que o VS.NET Beta 1 usava o padrão “SDL” correspondente ao “SOAP Toolkit 1.0”, baseado em uma especificação temporária do protocolo. O Beta 2 utiliza o padrão definitivo e ligeiramente diferente chamado “WSDL”, correspondente ao SOAP Toolkit 2.0.

Consumindo um Web Service

Para consumir um WebService, o Visual Studio.NET pode criar uma classe “proxy” a partir de informação obtida interrogando o SDL. Vamos criar um novo projeto “WinForms” para chamar o WebService:

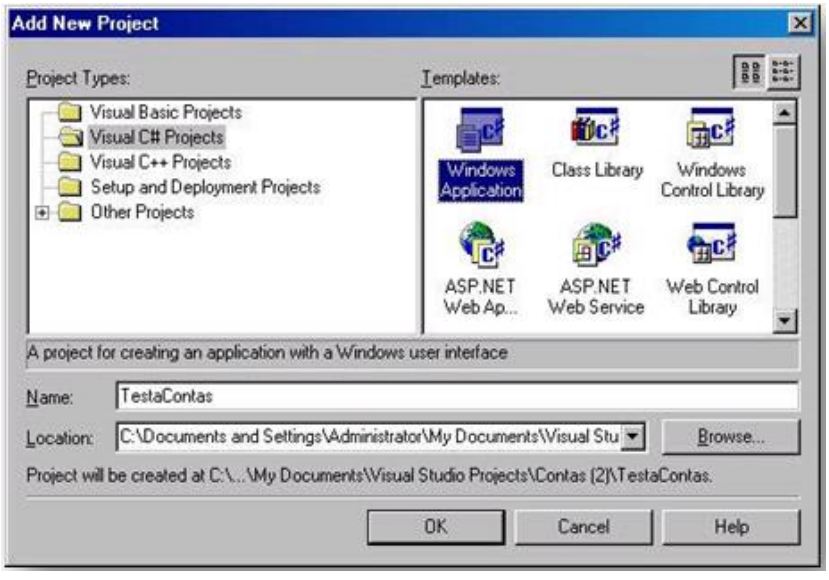

Figura 7: Criando nova aplicação windows

Adicione dois TextBox, um Button e um ListBox:

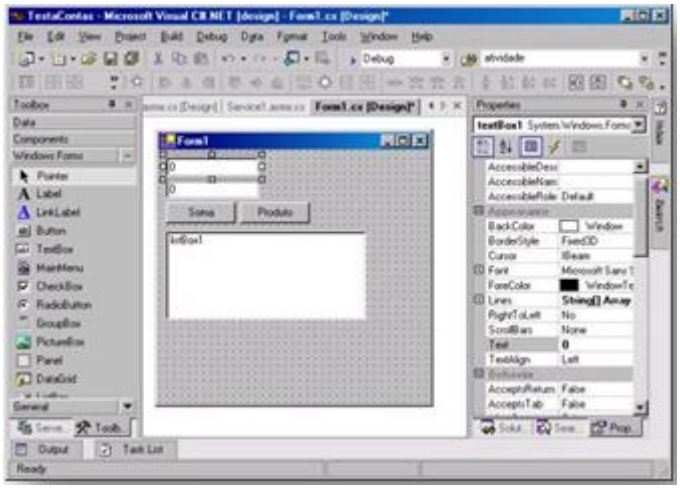

Figura 8: Layout do form

Clique com o botão direito sobre o projeto e peça “Add Web Reference...”:

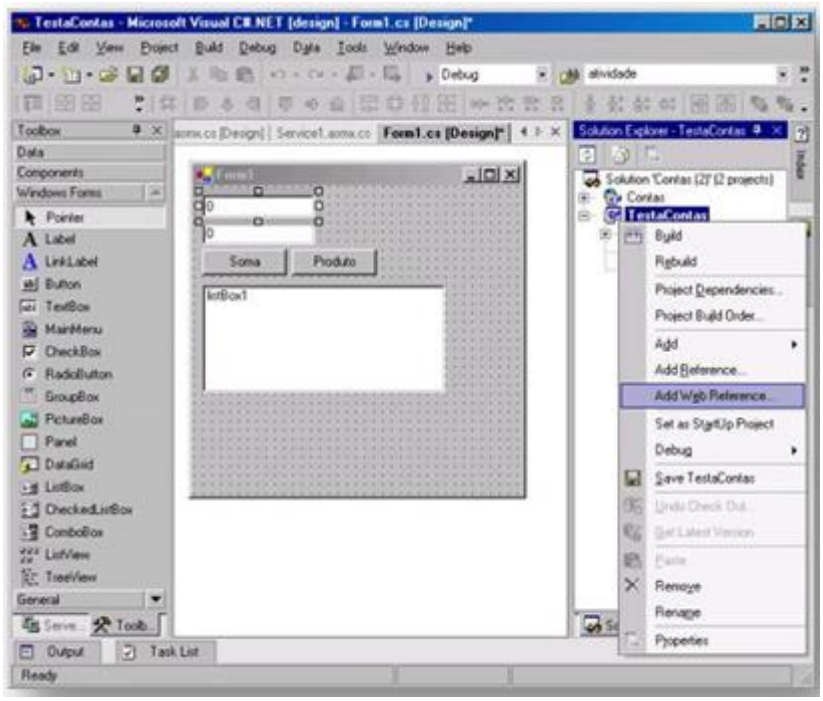

Figura 9: Referenciando o web service

Entre com a URL do WebService, http://localhost/Contas/Service1.asmx, no caso. A janela que aparece à esquerda é um navegador Web que pode ser utilizado normalmente para navegação e localização dos WebServices, listados à direita:

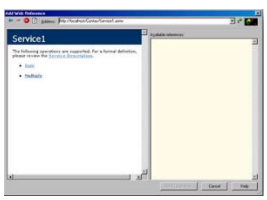

Figura 10: Informando o caminho do web service

Clique “Add Reference”. O Visual Studio.NET criará uma classe “proxy” no projeto. Esta classe tem a mesma sintaxe de uma classe .NET, mas na verdade está invocando um WebService.

Veja o projeto com a “Web Reference”:

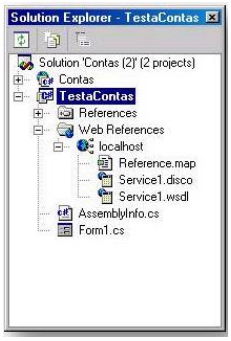

Figura 11: Estrutura do projeto após a adição da referência

Se você tiver curiosidade, esta é a classe criada:

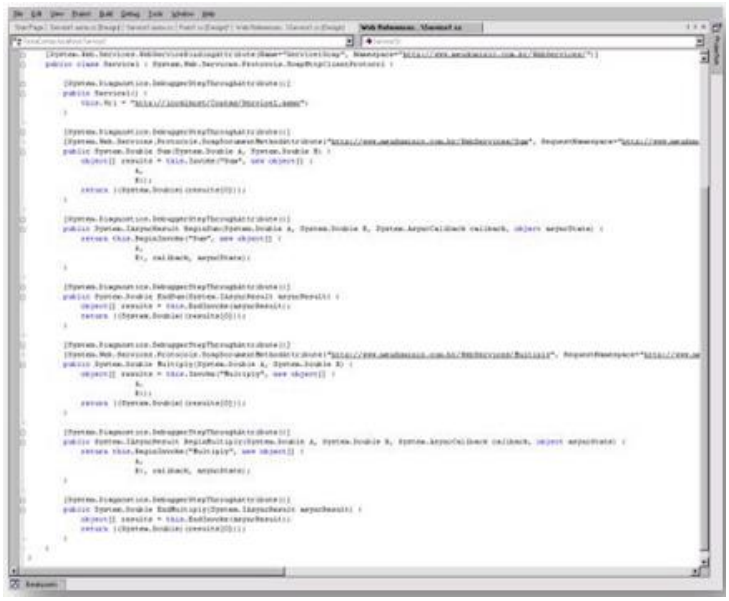

Figura 12: Código da classe criada

O código que chama o WebService é o seguinte:

Listagem 2: Código para utilizar o serviço

```
1  // Soma
2  private void button1_Click(object sender, System.EventArgs e) {
3  double N1 = Convert.ToDouble(textBox1.Text);
4  double N2 = Convert.ToDouble(textBox2.Text);
5  localhost.Service1 Contas = new localhost.Service1();
6  double R = Contas.Sum(N1, N2);
7  listBox1.Items.Add(R.ToString());
```

```
8  }
9  // Produto
10private void button2_Click(object sender, System.EventArgs e) {
11double N1 = Convert.ToDouble(textBox1.Text);
12double N2 = Convert.ToDouble(textBox2.Text);
13localhost.Service1 Contas = new localhost.Service1();
14double R = Contas.Multiply(N1, N2);
15listBox1.Items.Add(R.ToString());
16}
```

Veja o programa rodando e chamando o WebService:

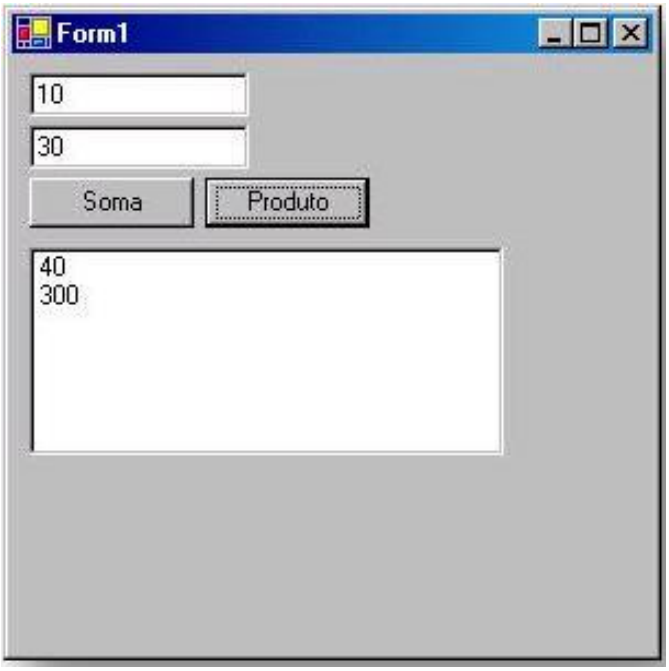

Figura 13: Programa em execução

Note que criamos um objeto da classe proxy e chamamos seus métodos para usá-la. O objeto Cota-susado para chamar o WebService funciona de forma muito semelhante a um navegador Internet, mas sem interpretar documentos HTML. Ele pode armazenar cookies, fazer autenticação e encripta-ção, ter um nome como “user agent” e usar um servidor de proxy.

Na segunda parte, mostrarei o uso de WebServices “de verdade” na Internet e também como criar-WebServices mais complexos.

REST

Origem: Wikipédia, a enciclopédia livre.

A **Representational State Transfer** (**REST**), em português **Transferência de Estado Representa-cional**, é uma abstração da arquitetura da World Wide Web (Web), um estilo arquitetural que consiste de um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e ele-mentos de dados dentro de um sistema de hipermídia distribuído.

O REST ignora os detalhes da implementação de componente e a sintaxe de protocolo com o objetivo de focar nos papéis dos componentes, nas restrições sobre sua interação com outros componentes e na sua interpretação de elementos de dados significantes.

O termo transferência de estado representacional foi apresentado e definido no ano de 2000 por Roy Fielding, um dos principais autores da especificação do protocolo HTTPque é utilizado por sites da Internet, em uma tese de doutorado (PHD) na UC Irvine.

A REST tem sido aplicada para descrever a arquitetura web desejada, identificar problemas existentes, comparar soluções alternativas e garantir que extensões de protocolo não violem as principais restrições que fazem da Web um sucesso. Fielding desenvolveu a REST em colaboração com seus colegas enquanto trabalhava no HTTP 1.1 e nos Identificadores de Recursos Uniformes.

O estilo arquitetural de REST também é aplicado no desenvolvimento de serviços Web. Pode-se caracterizar os web services como "RESTful" se eles estiverem em conformidade com as restrições descritas na seção restrições arquiteturais.

“A REST (Transferência do Estado Representativo) é pensada como uma imagem do design da aplicação se comportará: uma rede de sítios da Teia (um estado virtual), onde o utilizador progride com uma aplicação clicando em vínculos (transições do estado), tendo como resultado a página seguinte (que representa o estado seguinte da aplicação) que está sendo transferida ao utilizador e apresentada para seu uso. ”

O termo REST se referia, originalmente, a um conjunto de princípios de arquitetura (descritos mais abaixo), na atualidade se usa no sentido mais amplo para descrever qualquer interface web simples que utiliza XML (ou YAML, JSON, ou texto puro) e HTTP, sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem como o protocolo de serviços Web SOAP.

É possível projetar sistemas de serviços Web de acordo com o estilo arquitetural REST descrito por Fielding, e também é possível projetar interfaces XMLHTTP de acordo com o estilo de RPC mas sem utilizar SOAP. Estes usos diferentes do termo REST causam certa confusão em discussões técnicas, onde RPC não é um exemplo de REST.

Os sistemas que seguem os princípios REST são freqüentemente chamados de RESTful'.

Princípios

REST afirma que a Web já desfrutou de escalabilidade como resultado de uma série de conceitos de projeto fundamentais:

Um protocolo cliente/servidor sem estado: cada mensagem HTTP contém toda a informação necessária para compreender o pedido. Como resultado, nem o cliente e nem o servidor necessitam gravar nenhum estado das comunicações entre mensagens.

Na prática, muitas aplicações baseadas em HTTP utilizam cookies e outros mecanismos para manter o estado da sessão (algumas destas práticas, como a reescrita de URLs, não são permitidas pela regra do REST).

Um conjunto de operações bem definidas que se aplicam a todos os recursos de informação: HTTP em si define um pequeno conjunto de operações, as mais importantes são **POST**, **GET**, **PUT** e **DELETE**.

Com frequência estas operações são combinadas com operações CRUD para a persistência de dados, onde POST não se encaixa exatamente neste esquema.

Uma sintaxe universal para identificar os recursos. No sistema REST, cada recurso é unicamente direcionado através da sua URI.

O uso de hipermídia, tanto para a informação da aplicação como para as transições de estado da aplicação: a representação deste estado em um sistema REST são tipicamente HTML ou XML.

Como resultado disto, é possível navegar com um recurso REST a muitos outros, simplesmente seguindo ligações sem requerer o uso de registros ou outra infraestrutura adicional.

Recursos

Um conceito importante em REST é a existência de recursos (elementos de informação), que podem ser usados utilizando um identificador global (um Identificador Uniforme de Recurso) para manipular estes recursos, os componentes da rede (clientes e servidores) se comunicam através de uma interface padrão (HTTP) e trocam representações de recursos (os arquivos ou ficheiros são recebidos e enviados) – é uma questão polêmica e gera grande discussão, sem a distinção entre recursos e suas representações é demasiado utópico o seu uso prático na rede, onde é popular na comunidade RDF.

O pedido pode ser transmitido por qualquer número de conectores (por exemplo clientes, servidores, caches, etc) mas não poderá ver mais nada do seu próprio pedido (conhecido com separação de capas, outra restrição do REST, que é um princípio comum com muitas outras partes da arquitetura de redes e da informação).

Assim, uma aplicação pode interagir com um recurso conhecendo o identificador do recurso e a ação requerida, não necessitando conhecer se existem caches, proxys, ou outra, entre ela e o servidor que guarda a informação.

A aplicação deve compreender o formato da informação de volta (a representação), que é geralmente um documento em formato HTML ou XML, onde também pode ser uma imagem ou qualquer outro conteúdo.

REST e RPC

Uma aplicação web REST requer um enfoque de desenho diferente a uma aplicação baseada em RPCs. No RPC, dá-se ênfase à diversidade de operações do protocolo, ou verbos; por exemplo uma aplicação RPC poderia definir operações como:

`getUser()`

`addUser()`

`removeUser()`

`updateUser()`

`getLocation()`

`addLocation()`

`removeLocation()`

`updateLocation()`

`listUsers()`

`listLocations()`

`findLocation()`

`findUser()`

Em REST, ao contrário, a ênfase está na diversidade de recursos, nos nomes; por exemplo, uma aplicação REST poderia definir os seguintes tipos de recursos:

Usuario {}

Localizacao {}

Cada recurso teria seu próprio identificador, como http://www.example.org/locations/us/ny/new_york_city.

Os clientes trabalhariam com estes recursos através das operações padrão de HTTP, como o GET para chamar uma cópia do recurso.

Observa-se como cada objeto tem sua própria URL e pode ser facilmente "cacheado", copiado e guardado como marcador. POST utiliza-se geralmente para ações com efeitos colaterais, como enviar uma ordem de compra.

Por exemplo, o registo para um Utilizador poderia ter o seguinte aspecto:

<usuario>

<nome>Maria Juana</nome>

<genero>feminino</genero>

<localizacao href="http://www.example.org/locations/us/ny/new_york_city"

>Nova York, NY, US</localizacao>

</usuario>

Para atualizar a localização do utilizador, um cliente REST poderia primeiro chamar um registo XML anterior usando GET. O cliente depois modificaria o arquivo para mudar a localização e submetê-la-ia para o servidor utilizando HTTP PUT.

Note-se que o HTTP não proporciona nenhum recurso padrão para descobrir recursos – não há nenhuma operação LIST ou FIND em HTTP, que se corresponda às operações list*() e find*() como por exemplo em RPC.

No seu lugar, as aplicações baseadas em dados REST resolvem o problema, tratando uma coleção de resultados de busca como outro tipo de recurso, o que requer que os engenheiros de software desenhem na aplicação colocando URLs adicionais para mostrar ou encontrar cada tipo de recurso.

Por exemplo, um pedido GET HTTP sobre a URL http://www.example.org/locations/us/ny/ poderia devolver uma lista de arquivos em XML com todas as localizações possíveis em Nova Iorque, e outra, um pedido GET para URL http://www.example.org/users?nome=Michaels poderia devolver uma lista de todos os usuários com o apelido "Michaels".

REST proporciona algumas indicações sobre como realizar este tipo de ação como parte de sua restrição "hipermídia como o meio de estado da aplicação", o que sugere o uso de uma linguagem de marcação (como um formulário HTML) para especificar consultas parametrizadas.

A iniciativa OpenSearch da A9.com tenta padronizar as buscas usando REST estabelecendo especificações para descobrir recursos e um formato genérico para utilizar sistemas baseados em REST, incluindo o RDF, XTM, Atom, RSS (e suas várias formas) e XML com XLink para gerir as ligações.

Implementações Públicas

Dado que a definição de REST é muito ampla, é possível afirmar que existe um enorme número de aplicações REST na rede (praticamente qualquer coisa acessível mediante um pedido HTTP GET). De forma mais restritiva, em contraposição aos serviços web e ao RPC, REST se pode encontrar em diferentes áreas da web:

Na blogosfera -o universo dos blogs- está, em sua maior parte, baseado em REST, dado que implica chamar arquivos/ficheros XML (em formato RSS ou Atom) que contém listas de ligações a outros recursos.

Amazon.com oferece sua interface tanto em formato REST como em formato SOAP (sendo a versão REST a que recebe maior tráfego)

eBay oferece uma interface REST.

O Projeto "Seniores Canada On-line" do Governo do Canadá oferece um interface REST .

Bloglines oferece uma API baseada em REST.

Yahoo! oferece uma API em REST.

O framework Ruby on Rails suporta aplicações REST utilizando o padrão MVC.

O publicador de objetos do Zope.

Os Frameworks Python possuem pleno suporte a REST, por exemplo: Django, Flask.

Os Frameworks para desenvolvimento em PHP, Laravel ,Zend Framework e CakePHP possuem componentes para criação de aplicações REST.

Os Frameworks REST NodeJS: Gugamarket - ver mais em Nodeframework.

Os Frameworks para desenvolvimento Web em Groovy,Grails e Spring-boot suportam aplicações REST utilizando padrão MVC.

De qualquer forma, deve-se ter em mente que as aplicações descritas acima não são totalmente escritas puramente em REST, isto é, não respeitam todas as restrições que impõe a arquitetura REST.

E sim, todas são inspiradas em REST e respeitam os aspectos mais significativos e restritivos da sua arquitetura, em particular a restrição de "interface uniforme". Estes serviços são denominados "Acidentalmente RESTful".-