Utilizando a Arquitetura Orientada a Serviços e o Desenvolvimento Baseado em Componente para Construir Aplicativos de Serviços da Web

Um White Paper da Rational Software

Alan Brown,
Simon Johnston,
Kevin Kelly

Rational®
the software development company

Conteúdo

Introdução

A construção de um sistema de software de escala corporativa é uma tarefa complexa. Apesar de décadas de avanços tecnológicos, as demandas impostas pelos atuais sistemas de informações muitas vezes ultrapassam os limites da capacidade de uma empresa para projetar, construir e evoluir suas soluções de software vitais. Em particular, alguns novos sistemas são projetados a partir da base. Mais propriamente, a tarefa de um arquiteto de software é normalmente estender a vida de uma solução, descrevendo a nova lógica de negócios que manipula um repositório de dados existente, apresentando dados e transações existentes por meio de novos canais, como um navegador de Internet ou dispositivos handheld, integrando sistemas anteriormente desconectados que suportam a sobreposição de atividades de negócios e assim por diante.

Para auxiliar os desenvolvedores de software, produtos de infra-estrutura de software comercial agora estão disponíveis em fornecedores como a Microsoft e a IBM. Eles formam a peça central das abordagens para desenvolvimento de software que eles defendem em suas linhas de produto .NET e WebSphere, respectivamente. Ambas as abordagens têm como foco a montagem de sistemas de serviços distribuídos. No entanto, há algo novo sobre a construção de soluções de escala corporativa de serviços? Como as lições de sistemas baseados em componentes aplicam-se à construção de arquiteturas baseadas em serviços (SOA)? Quais são as melhores abordagens para construir sistemas de alta qualidade para implementação nesta nova geração de produtos de infra-estrutura de software? Essas perguntas importantes são os tópicos deste documento.

Nos últimos anos, a maior parte da atenção na comunidade de engenharia de software se concentrou em abordagens de design, processos e ferramentas que suportam o conceito de que grandes sistemas de software podem ser montados a partir de coleções reutilizáveis independentes de funcionalidade. É possível que alguma funcionalidade já esteja disponível e implementada internamente ou tenha sido adquirida de terceiros, enquanto a funcionalidade restante pode precisar ser criada. Nesses casos, todo o sistema deve ser concebido e projetado para reunir todos esses elementos em um único conjunto coerente. Atualmente, isso é exemplificado em *CBD (Component-Based Development)*, um conceito que é compreendido em abordagens tecnológicas, como a plataforma Microsoft .NET e os padrões J2EE (Java 2 Enterprise Edition) suportados por produtos como o WebSphere da IBM e o iPlanet da Sun.

Uma consideração adicional é que os sistemas operacionais serão normalmente distribuídos em várias máquinas para aprimorar o desempenho, disponibilidade e escalabilidade. Uma solução corporativa precisa coordenar a funcionalidade que é executada em uma coleção de hardware. Uma solução corporativa precisa coordenar a funcionalidade que é executada em uma coleção de hardware. Cada serviço fornece acesso a uma coleção bem definida de funcionalidade. O sistema como um todo é projetado e implementado como um conjunto de interações entre esses serviços. A exposição da funcionalidade como serviços é a chave para a flexibilidade. Isso permite que outras partes da funcionalidade (talvez elas mesmas implementadas como

serviços), utilizem outros serviços de um modo natural, independentemente de seus locais físicos. Um sistema evolui com a inclusão de novos serviços. A *SOA (Service-Oriented Architecture)* resultante define os serviços dos quais o sistema é composto, descreve as interações que ocorrem entre os serviços para compreender determinado comportamento e mapeia os serviços para uma ou mais implementações em tecnologias específicas.

Embora os serviços encapsulem a funcionalidade de negócios, alguma forma de infra-estrutura entre serviços é necessária para facilitar as interações e a comunicação dos serviços. Diferentes formas dessa infra-estrutura são possíveis porque os serviços podem ser implementados em uma única máquina, distribuídos em um conjunto de computadores em uma rede local ou distribuídos mais amplamente nas redes de várias empresas. Um caso particularmente interessante é quando os serviços utilizam a Internet como o mecanismo de comunicação. Os *serviços da Web* resultantes compartilham as características de serviços mais gerais, mas requerem consideração especial como resultado da utilização de um mecanismo público, não seguro, de baixa fidelidade para interações entre serviços.

Até o momento, grande parte do foco do segmento de mercado de software tem sido a tecnologia subjacente para implementar serviços da Web e suas interações. Entretanto, outras preocupações surgem em torno da questão sobre o modo mais apropriado de projetar serviços da Web no intuito de facilitar a montagem em soluções de escala corporativa. Por outro lado, tem havido uma surpreendente falta de atenção sobre as práticas e ferramentas apropriadas para arquitetar as soluções de software de escala corporativa compostas de serviços da Web. Assim como ocorre com o design de qualquer estrutura complexa, as soluções de alta qualidade são o resultado de decisões arquiteturais antecipadas, suportadas por uma série de técnicas de design, padrões estruturais e estilos bem entendidos. Esses padrões tratam de problemas de serviços comuns, como escalabilidade, confiabilidade e segurança.

Este documento fornece o contexto para um entendimento mais profundo dos serviços e das arquiteturas orientadas a serviços para soluções de software de escala corporativa. Em particular, ele explora os serviços em relacionamento quanto ao conceito mais estabelecido dos componentes de software e descreve como as práticas atuais de desenvolvimento baseado em componente fornecem uma base experimentada e testada para a implementação de uma arquitetura orientada a serviços. O design baseado em interface é realçado como a chave para os designs de serviço e de componente e deduz-se que as interfaces expostas por ambos possuem determinadas restrições e critérios que as diferenciam. A UML (Unified Modeling Language)[1] é utilizada como uma ferramenta para descrever designs lógicas e de implementação, bem como padrões específicos para os designs de componente e de serviço.

Por último, o documento tem como foco os serviços da Web como um meio para explorar problemas relacionados à implementação de serviços em geral. Especificamente, o documento investiga como interpretar a orientação geral para arquitetura orientada a serviços como padrões de design específicos para serviços da Web. Todos os padrões descritos neste documento foram implementados utilizando os recursos exclusivos de gabarito de código e padrão do Rational® XDE™ e publicados por meio do Rational Developer Network[8].

O que é uma Arquitetura Orientada a Serviços?

Então o que é uma SOA (Service-Oriented Architecture)? Basicamente, é uma maneira de projetar um sistema de software para fornecer serviços para aplicativos de usuário final ou outros serviços por meio de interfaces publicadas e interfaces descobríveis. Em muitos casos, os serviços fornecem uma maneira melhor de expor funções de negócios distintas e, portanto, uma excelente maneira de desenvolver aplicativos que suportam processos de negócios.

A arquitetura orientada a serviços não é um noção recente; é importante no momento por causa da crescente tecnologia de serviços da Web. Por exemplo, aqui está uma citação de um livro publicado em 2000 que descreve o valor da arquitetura orientada a serviços:

> *Soluções Orientadas a Serviços… Os aplicativos devem ser desenvolvidos como conjuntos independentes de serviços de interação que oferecem interfaces bem definidas para seus potenciais*

usuários. De modo semelhante, a tecnologia de suporte deve estar disponível para permitir que os desenvolvedores de aplicativos procurem coleções de serviços, selecione aqueles de interesse e monte-os para criar a funcionalidade desejada. [2]

Para as finalidades deste documento, considere a seguinte definição de um serviço:

Um serviço é geralmente implementado como uma entidade de software grosseira descobrível que existe como uma única instância e interage com aplicativos e outros serviços por meio de um modelo de comunicação desconexo (geralmente assíncrono) baseado em mensagem.

De muitas maneiras, a terminologia de serviços é muito parecida com a terminologia utilizada para descrever o desenvolvimento baseado em componente; no entanto, há termos específicos utilizados para definir elementos nos serviços da Web, conforme mostrado na Figura 1 a seguir.

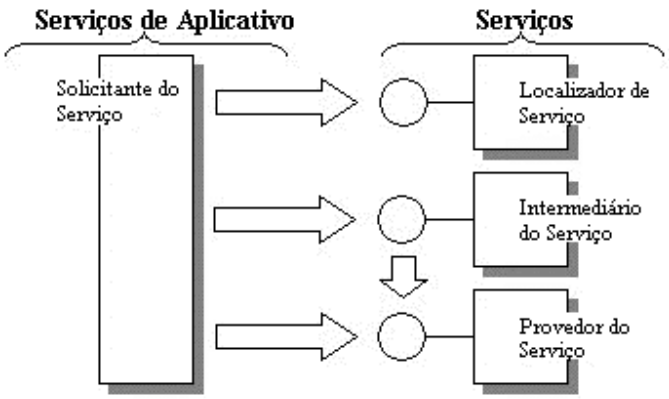

Figura 1 – Terminologia de Serviço

- **Serviço** — Uma entidade lógica; o contrato definido por uma ou mais interfaces publicadas.
- **Provedor de Serviços** — A entidade de software que implementa uma especificação de serviço.
- **Solicitante de Serviços** — A entidade de software que chama um fornecedor de serviços. Tradicionalmente, é chamado de "cliente"; entretanto, um solicitante de serviços pode ser um aplicativo de usuário final ou um outro serviço.
- **Locador de Serviços** — Um tipo específico de fornecedor de serviços que age como um registro e permite a consulta de interfaces de fornecedores de serviços e locais de serviços.
- **Intermediário de Serviços** — Um tipo específico de fornecedor de serviços que pode transmitir pedidos de serviço para um ou mais fornecedores de serviços adicionais.

Esta descrição de serviços e o contexto de seu uso impõem uma série de restrições. Além disso, o uso eficiente de serviços sugere algumas boas práticas de alto nível. Aqui estão algumas características-chave para o uso eficiente dos serviços:

- **Grosseiro** — As operações em serviços são muitas vezes implementadas para abranger mais funcionalidade e operar em conjuntos maiores de dados, em comparação com o design de interface de componente.
- **Design baseado em interface** — Os serviços implementam interfaces definidas separadamente. A vantagem disso é que vários serviços podem implementar uma interface comum e um serviço pode implementar várias interfaces.
- **Descobrível** — Os serviços precisam ser localizados no tempo de design e no tempo de execução, não apenas por identidade exclusiva, mas também por identidade de interface e por tipo de serviço.
- **Instância única** — Diferente do desenvolvimento baseado em componente, que instancia os componentes conforme necessário, cada serviço é uma instância única sempre em execução com a qual uma série de clientes se comunicam.

- **Desconexo** — Os serviços são conectados a outros serviços e clientes utilizando métodos padrão baseados em mensagem, desconexos e com redução de dependência, como trocas de documentos XML.
- **Assíncrono** — Em geral, os serviços utilizam uma abordagem de transmissão assíncrona de mensagens; no entanto, isso não é obrigatório. De fato, muitos serviços utilizarão às vezes a transmissão assíncrona de mensagens.

Alguns desses critérios, como design baseado em interface e recurso de descoberta, também são utilizados no desenvolvimento baseado em componente; no entanto, é a soma total desses atributos que diferenciam um aplicativo baseado em serviço de um aplicativo desenvolvido utilizando arquiteturas de componente, como J2EE ou .NET.

Design Baseado na Interface

No desenvolvimento de componente e de serviço, o design das interfaces é feito de modo que uma entidade de software implemente e exponha uma parte-chave de sua definição. Portanto, a noção e o conceito de "interface" é chave para um design bem-sucedido em sistemas baseados em componente e orientados a serviços. A seguir estão algumas definições relacionadas à interface-chave:

- **Interface** — Define um conjunto de assinaturas de método público, logicamente agrupadas, mas sem fornecer implementação. Uma interface define um contrato entre o solicitante e o fornecedor de um serviço. Qualquer implementação de uma interface deve fornecer todos os métodos.
- **Interface Publicada** — Uma interface que é exclusivamente identificável e disponibilizada por meio de um registro para os clientes descobrirem dinamicamente.[3]
- **Interface Pública** — Uma interface que é disponibilizada para os clientes utilizarem, mas não é publicada, portanto requer conhecimento estático na parte do cliente.
- **Interface Dual** — Muitas vezes, as interfaces são desenvolvidas como pares, de modo que uma interface dependa da outra; por exemplo, um cliente deve implementar uma interface para chamar um solicitante porque a interface de cliente fornece algum mecanismo de retorno de chamada. Esse conceito foi introduzido por serviços da Web.

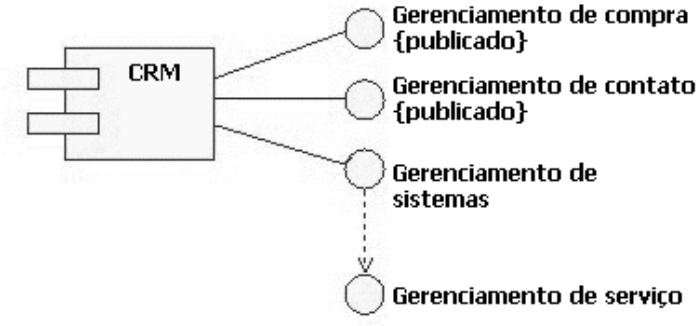

Figura 2 – Serviços implementados

A Figura 2 mostra a definição UML de um serviço CRM (Customer Relationship Management), representado como um componente UML, que implementa as interfaces AccountManagement, ContactManagement e SystemsManagement. Apenas as duas primeiras são interfaces publicadas, embora a última seja uma interface pública. Observe que as interfaces SystemsManagement e ManagementService formam uma interface dual. O serviço CRM pode implementar qualquer número dessas interfaces e é essa capacidade de um serviço (ou componente) se comportar de várias maneiras, dependendo do cliente, que permite grande flexibilidade na implementação do comportamento. Também é possível fornecer serviços diferentes ou adicionais para classes específicas de clientes. Em alguns ambientes de tempo de execução, esse recurso também é utilizado para suportar diferentes versões da mesma interface em um único componente ou serviço.

Comportamento da Interface

Uma definição de interface em linguagens como Java ou C# ou em linguagens como IDL fornece apenas um conjunto de assinaturas de método. A definição fornece o "what" sem qualquer orientação sobre o "how." Por exemplo, considere a interface de Segurança na Figura 3. Está claro que os clientes que chamam uma implementação desta interface também podem chamar um dos três métodos públicos — não é?

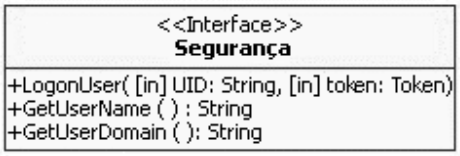

Figura 3 – Interface na UML

Simplesmente definindo o "what" não podemos retratar o fato de que o cliente não pode chamar GetUserName( ) ou GetUserDomain( ) até que o usuário tenha efetuado logon. A máquina de estados a seguir demonstra essa dependência, ou comportamento. Esse tipo de restrição está geralmente incluído na literatura sobre o design baseado em interface; entretanto, ele não é suportado em linguagens de programação e, por isso, não se pode assegurar que o implementador de uma interface seja compatível com qualquer especificação comportamental.

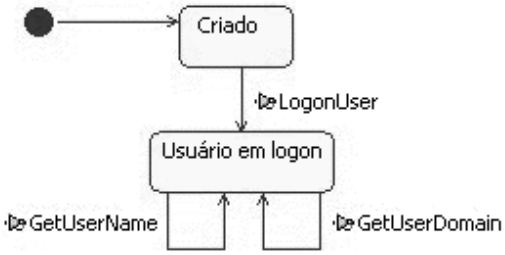

Figura 4 – Comportamento da interface

No entanto, os negócios estão mudando cada vez mais para sistemas orientados a serviços na esperança de que possam ser mais facilmente integrados e coordenados para concretizar os processos de negócios por meio de colaborações de serviços. Como resultado, a noção de definição do comportamento de uma interface e, o mais importante, do comportamento dos conjuntos de interfaces relacionadas está recebendo mais atenção do segmento de mercado. Infelizmente, atualmente há poucas abordagens sobre isso.

Uma abordagem poderia ser o uso de modelos de design, como aqueles introduzidos por meio deste documento, definidos em uma linguagem padronizada, como a UML, para documentar as interdependências entre as interfaces de serviço. Esses modelos podem ser compartilhados, socializados e utilizados para lançar padrões específicos quando eles surgem.

Além disso, a Rational patrocinou a RAS (Reusable Asset Specification), que fornece um mecanismo para empacotar e compartilhar recursos que poderiam ser aplicados a esse problema. Por exemplo, ao utilizar o mecanismo RAS para distribuir os detalhes de um serviço, alguém poderia também empacotar o modelo que descreve seu comportamento. Em um modelo como esse, pode-se utilizar um diagrama de seqüência para mostrar a interação necessária entre as chamadas na interface.

Arquitetando Sistemas Orientados a Serviços

Em qualquer novo desenvolvimento na engenharia de software, é muito fácil pressupor que possam ser aplicadas as mesmas técnicas e ferramentas que funcionaram em projetos anteriores. Conforme já mencionado, os componentes e serviços, embora semelhantes , não são iguais; eles possuem diferentes

critérios e padrões de design. Esta seção descreverá uma importante conseqüência prática. *O fator crucial é que nem todo bom componente transformado em um serviço faz um bom serviço.*

Dividindo o Design do Aplicativo em Camadas

Esta tendência para resolver novos problemas com soluções obsoletas não é nova. De modo semelhante, à medida que os desenvolvedores começaram a criar sistemas baseados em componente, eles tentaram aplicar suas experiências com o desenvolvimento orientado a objetos com problemas semelhantes. Com mais experiência, entendeu-se que a tecnologia orientada a objetos e as linguagens são excelentes maneiras de implementar componentes, embora se deva entender os dilemas criados por decisões e implementação. Dilemas relativos à herança vs. agregação para implementar um comportamento polimórfico ou o redesign de bibliotecas de classe para poder utilizar em conjuntos de componentes e não como a base para um aplicativo C++ monolítico.

De modo semelhante, vemos os *componentes como a melhor maneira de implementar serviços*, embora se deva entender que um aplicativo baseado em componente exemplar não torna necessariamente um aplicativo orientado a serviços exemplar. Vemos uma excelente oportunidade para alavancar os desenvolvedores de componentes e os componentes existentes de sua empresa, uma vez que a função desempenhada pelos serviços na arquitetura de aplicativo seja entendida. A chave para fazer essa transição é compreender que uma abordagem orientada a serviços implica em uma camada de arquitetura de aplicativo adicional. A Figura 5 a seguir demonstra como as camadas de tecnologia podem ser aplicadas à arquitetura de aplicativo para fornecer implementações mais grosseiras à medida que se aproxima dos consumidores do aplicativo. O termo para se referir a esta parte do sistema é " linha de aplicativo," refletindo o fato de que um serviço é uma excelente maneira de expor uma visão interna de um sistema, com reutilização interna e composição utilizando o design de componente tradicional.

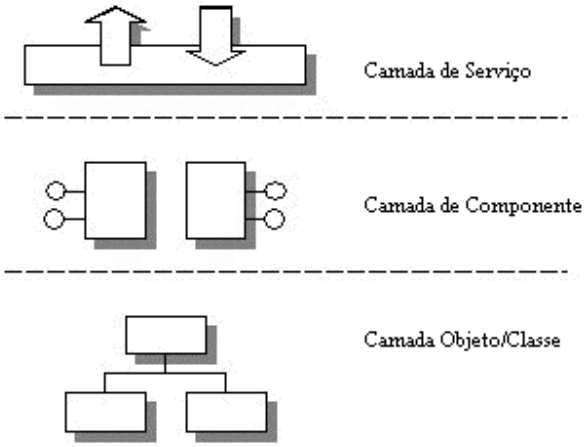

Figura 5 – Camadas de implementação do aplicativo

Em geral, a mudança do conceito "orientado a objetos" para "baseado em componente" levou entre 6 e 18 meses enquanto os desenvolvedores aprendiam sobre esta nova tecnologia e os requisitos estabelecidos. Felizmente, a mudança para sistemas orientados a serviços pode ocorrer mais rapidamente. Para essa finalidade, os desenvolvedores precisarão entender os desafios, os dilemas, e as decisões de design que permitem o desenvolvimento e a reutilização de componentes no suporte de aplicativos orientados a serviços.

Exemplo do Modelo de Cliente

Para ilustrar como os componentes e serviços interagem na compreensão de um aplicativo, considere um exemplo que gerencia algumas informações de relacionamento do cliente, conforme definido no diagrama de classe UML na Figura 6 a seguir.

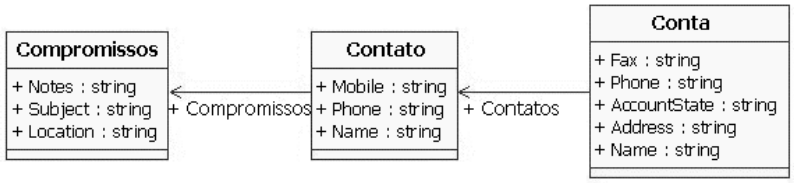

Figura 6 – Modelo lógico de cliente

Nas seções a seguir, descobriremos como os desenvolvedores transcrevem esse modelo lógico em um modelo de implementação para aplicativos baseados em componente e, por conseguinte, para aplicativos baseados em serviço (tendo em mente que esse modelo não descreve o comportamento do sistema de nenhuma maneira). Ficará claro que muitas dessas etapas de conversão podem ser geradas automaticamente. A Rational Software possui ferramentas para a modelagem da arquitetura de aplicativos, o resultado e a aplicação dos padrões e o gerenciamento desses artefatos de modelo/código no ciclo de vida completo do desenvolvimento.

Um Design Baseado em Componente

Como nosso exemplo de modelo lógico seria convertido em um design de componente (por exemplo, para COM ou J2EE)? Um design baseado em componente para o modelo é apresentado a seguir, na Figura 7.

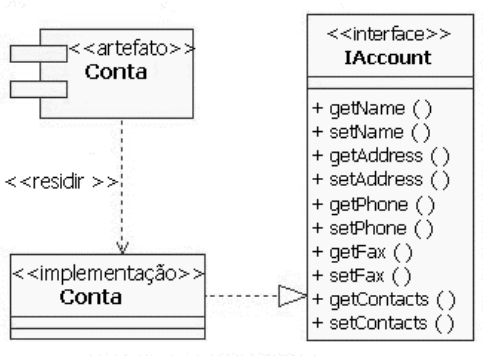

Figura 7 – Diagrama de componente genérico

Há um recurso de design *chave* da interface acima. A chave é que, para plataformas de componentes existentes, há um padrão comum; para cada atributo na classe de análise, temos que fornecer duas operações — uma que defina o valor e outra que retorne o valor. Para componentes locais, o código extra de uma chamada de método é insignificante e, para objetos remotos, os mecanismos RPC (Remote Procedure Call) são otimizados para minimizar o código extra. Em muitos casos, em um aplicativo, o cliente precisa somente de um subconjunto das propriedades e, desse modo, pode acessá-las conforme necessário.

Um Design Orientado a Serviços

O modelo de design acima demonstra a maneira correta de pensar sobre implementações de componentes, em que cada instância de componente representa um único objeto; por exemplo, para cada contato individual em nosso CRM, o banco de dados torna-se, logicamente, um componente separado. Portanto, a identidade do componente está vinculada à identidade do contato.

Entretanto, para um serviço, há uma única instância que gerencia um conjunto de recursos, portanto eles são na maior parte sem estado. Isso significa que precisamos visualizar um serviço como um objeto *gerenciador* que possa criar e gerenciar instâncias de um tipo ou conjunto de tipos. Isso produz um padrão de design que utiliza *objetos de valores* (um padrão comum em sistemas distribuídos em que o estado persiste para transferência entre componentes) que representam o estado da instância, objetos que são na realidade um estado simplesmente serializado. Um resultado interessante disso é que, se pudermos definir as regras para obter uma definição de componente, como o modelo na Figura 7, e transformá-la em um serviço, podemos implementar essa serialização como um padrão. É possível a criação e reutilização desses padrões utilizando o Rational XDE.

Essa passagem de estado do fornecedor para o solicitante indica que, em vez de um número grande de pequenas operações para recuperar o estado do componente, utiliza-se uma única operação grande. Isso tem determinadas implicações quanto ao uso da rede para serviços remotos (e a maioria dos serviços são remotos), bem como quanto ao comportamento dos solicitantes ao tratar de objetos de valores grandes. Também há uma outra implicação; o solicitante recebe uma cópia do estado de alguma entidade, mas esta cópia está stale? Quando recuperamos uma cotação de ações ou previsão de tempo, sabemos que existe a possibilidade de que estejam desatualizados, mas estamos condicionados a aceitar isso. Também estamos condicionados pelo tipo de dados; os dados de cotação de ações tornam-se stale mais rápidos que os dados de tempo. Em uma arquitetura tal como a descrita aqui, os solicitantes devem ser condicionados a aceitar cópias de estado. Para obter detalhes, consulte a seção “Impacto de Interações Grosseiras” posteriormente neste documento.

Uma vez estabelecidos esses requisitos, como você espera que seja um serviço? O fragmento de modelo na Figura 8 mostra como esse padrão pode ser descrito em um nível de design, demonstrando as interfaces publicadas pelo componente e os objetos de valor manipulados pela interface.

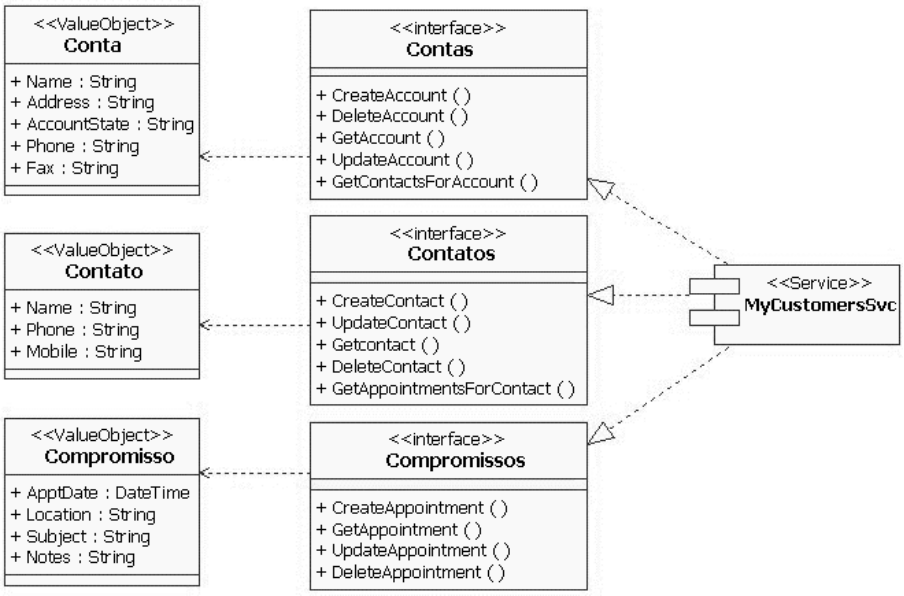

Figura 8 – Design genérico orientado a serviços

Agora que sabemos como os serviços podem ser projetados, vamos examinar alguns atributos deste design específico. Para começar, é evidente que há muito mais informações transmitidas no objetos de valores do que um simples “get” ou “set” do fornecedor, MyCustomerSvc, a um solicitante para uma determinada interação e que isso pode afetar a largura da banda da rede. No entanto, uma vez determinada a natureza dos serviços da Web, fica claro que os protocolos utilizados na implementação dos serviços são muito diferentes daqueles utilizados nas implementações dos componentes. Uma plataforma como essa atribui uma responsabilidade adicional ao arquiteto ou engenheiro de informações para escolher com cautela os objetos de valores e suas composições para maximizar o conteúdo de cada objeto de valor ao mesmo tempo em que não força a rede.

Armazenamento em Cache no Design Orientado a Serviços

Vamos reconsiderar a noção introduzida na seção anterior sobre a transmissão de cópias “stale” de informações de um fornecedor para um solicitante. Por exemplo, se estou desenvolvendo um aplicativo de gerenciamento de portfólio de ações, não desejo solicitar a um serviço da Web o preço atual de uma segurança repetidas vezes para cada segurança; transmitindo de 3 a 5 caracteres de dados para a segurança e de 5 a 7 caracteres para o retorno do preço. Isso pode resultar em um carregamento inaceitável na rede e no fornecedor de serviços. O que o solicitante deve fazer é solicitar o conteúdo do portfólio inteiro, transmitindo a lista de símbolos ou transmitindo o identificador do portfólio para o serviço e recuperando todas as informações de cada segurança. Então, se o usuário simplesmente solicitasse uma atualização para um único símbolo, isso daria a impressão de uma escala muito maior; entretanto, o solicitante agora pode armazenar em cache os resultados e, se o usuário solicitar uma atualização para outro símbolo, o pedido poderá ser atendido a partir do cache. O trabalho no solicitante é agora para identificar a duração da “locação” dos dados. Portanto, para um portfólio, se eu souber que o serviço de cotação de ações tem um retardo de 20 minutos, eu posso trabalhar em uma margem de 25% e armazenar os resultados em cache durante 5 minutos.

Esse padrão é visto repetidas vezes em sistemas de informações. Toda vez que um usuário recupera um pedido de um sistema de gerenciamento de pedidos, esse usuário recebe efetivamente uma cópia do pedido porque um outro usuário pode estar atualizando-o ao mesmo tempo (a menos que o sistema trave o acesso adicional ao pedido). Agora, seria ideal se um fornecedor de serviços da Web pudesse realmente, como parte de sua interação com o solicitante, identificar o cache ou a duração da locação. Esses problemas são bem entendidos nos sistemas de mensagens como o MSMQ (Microsoft Message Queue Server) e o IBM MQSeries, em que os tempos de limite de mensagens e os tempos de expiração são gerenciados de modo habitual.

Nós revisitaremos o problema posteriormente neste documento e forneceremos alguma orientação específica sobre o desenvolvimento de solicitantes e fornecedores para lidar com esse problema.

Design do Aplicativo de Serviços da Web XML

Os serviços da Web XML agora fazem parte de nosso vocabulário e há um amplo suporte a fornecedores, bem como um número crescente de plataformas e ferramentas para implementar e desenvolver serviços da Web. Os elementos da pilha de serviços da Web foram adequadamente descritos em alguma parte, portanto não entraremos em uma revisão exaustiva neste documento. A definição a seguir é proveniente do Grupo de Trabalho de Arquitetura de Serviços da Web W3C (World Wide Web Consortium):

> *Um serviço da Web é um aplicativo de software identificado por um URI, cujas interfaces e ligações podem ser definidas, descritas e descobertas por artefatos XML, e suporta interações diretas com outros aplicativos de software utilizando mensagens baseadas XML por meio de protocolos baseados em Internet.*[4]

Vamos considerar resumidamente como algumas das tecnologias no uso atual do serviço da Web são aplicadas à figura representada na Figura 1 quando descrevemos a terminologia de arquitetura orientada a serviços.

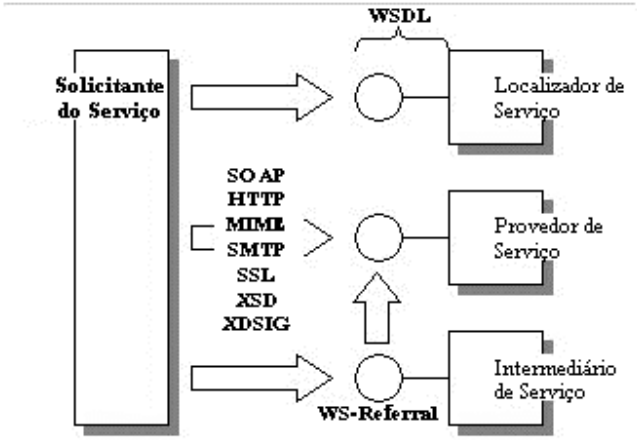

Figura 9 – Padrões de serviço da Web XML

Para iniciar, há um concepção errônea de que todos os serviços da Web utilizam mensagens XML, com SOAP (Simple Object Access Protocol) sobre HTTP — o protocolo, de fato, da Web. Isso não é totalmente verdadeiro. Uma mensagem de serviço da Web pode utilizar XML, mas poderia transportar dados binários; ela geralmente utiliza cabeçalhos SOAP, mas não é obrigatório utilizar codificação SOAP para o corpo da mensagem; ela pode utilizar HTTP como um transporte, mas poderia utilizar perfeitamente SMTP ou outros meios. Para a descrição e descoberta de serviços da Web, há dois padrões bem definidos: WSDL (Web Services Definition Language) e UDDI (Universal Discovery, Description and Integration).

Esta flexibilidade em formatos e protocolos de transporte é um dos atuais problemas com a interoperabilidade — dos serviços da Web. Consideradas as opções em formatos SOAP, envelopes, protocolos de transporte e assim por diante, como duas implementações podem transferir informações? O grupo WS-I (Web Services Interoperability) está avançado nisto, o qual está tentando fornecer ao segmento de mercado a orientação sobre o uso dos padrões atuais e emergentes. Os fornecedores também estão ajudando aqui, fornecendo ambientes de desenvolvimento de serviços da Web flexíveis, como o IBM WebSphere Studio Application Developer Integration Edition, que cria serviços da Web com vários formatos e protocolos de transporte para que o conjunto mais rápido ou correto possa ser utilizado, conforme necessário.

Design do Serviço da Web e Padrões de Implementação

Os serviços da Web realmente não alteram o processo de análise e design em torno dos requisitos funcionais de um aplicativo — um aplicativo de processamento de requerimentos de seguros ainda precisa processar os requerimentos de seguros! O que estamos introduzindo é um conjunto de restrições e possíveis problemas na área de requisitos não funcionais. As seções a seguir descrevem algumas dessas possíveis ciladas.

Desempenho e Confiabilidade

A pergunta feita com freqüência é se os recursos necessários para o desempenho, a confiabilidade e a escalabilidade dos serviços da Web podem ser fornecidos por uma arquitetura com base em HTTP e SOAP, que são propriamente lentos e não confiáveis. Primeiro, deve-se definir “lentos e não confiáveis”, depois se deve compreender que mesmos os transportes confiáveis no final do dia contam com meios não confiáveis. Ao arquitetar e projetar soluções de grande escala, devemos sempre ter em mente os requisitos funcionais e não funcionais e assegurar que os dilemas e decisões corretos sejam executados para suportar as metas de negócios.

Por exemplo, ao utilizar SOAP através HTTP, é sempre possível construir protocolos e interações no nível do aplicativo que forneçam recursos adicionais para confirmações de mensagens e segurança. Entretanto, se considerássemos que determinados serviços se comunicam dentro do mesmo contexto de segurança ou de

aplicativo, poderíamos considerar a utilização de meios diferentes do HTTP em qualquer caso. Considere o exemplo na Figura 10.

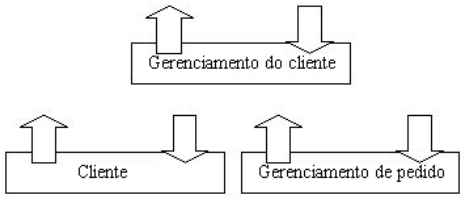

Figura 10 – Dependências de serviço

Basicamente, todos os clientes externos interagem com o serviço de Gerenciamento de Clientes; entretanto, ele interage com dois serviços internos. A decisão aqui é: Por que precisaríamos da flexibilidade do HTTP e SOAP para essas comunicações de serviço interno? Vamos supor que o desempenho fosse nosso requisito-chave para a interação entre o Gerenciamento de Clientes e os Clientes. Neste caso, poderíamos optar por utilizar uma comunicação RPC de componentes (como Microsoft .NET Remoting ou RMI do Java) que forneça formatos de codificação binária e características de desempenho superior. Por outro lado, o requisito-chave para fazer um pedido do Gerenciamento de Clientes para o Gerenciamento de Pedidos é para garantia de entrega, assim poderíamos utilizar alguma tecnologia de enfileiramento (como o IBM MQSeries ou MSMQ) para entregar a mensagem, em que o desempenho é negociado para um nível mais alto de confiabilidade.

É muito importante compreender que, mesmo que os serviços da Web apresentem um modelo simples e um conjunto de protocolos simples e flexíveis, você não está restrito a essas opções. Exatamente porque o WSDL já possui ligações para SOAP e HTTP GET/PUT, é importante fornecer opções adicionais para os solicitantes. Por exemplo, um único serviço pode expor uma mensagem utilizando uma ligação de fila de mensagem *e* uma ligação de SOAP, para que o solicitante possa escolher a ligação mais apropriada a ser utilizada. Neste caso, o fornecedor também pode fornecer incentivos, como um nível de garantia de serviço se a fila de mensagens for utilizada mas nenhuma garantia de serviço para uma conversação HTTP.

Uma outra decisão de design que se deve tomar antes é se as trocas de mensagens serão inalteradas, comutativas ou ambas. Ser *inalterada* significa que se a mesma mensagem chegar mais de uma vez, e *produzir efeito*, não haverá efeito desfavorável em nenhum caso. Ser *comutativa* significa que duas mensagens relacionadas podem chegar a qualquer ordem sem efeitos desfavoráveis. Se o design de um serviço puder ser de tal maneira que as trocas de mensagens sejam identificadas como pelo menos inalteradas, então os transportes menos confiáveis serão uma opção mais interessante (e econômica).

Da mesma maneira que essa segurança (não discutida neste documento) é, na verdade, um conjunto de opções, que variam de simples e econômicas a complexas e caras, as metas de desempenho, confiabilidade e escalabilidade para o design resultam em um conjunto de decisões: De quanto você precisa? De quanto você pode dispor? Os serviços fornecem tantas soluções, e também tantas opções, quantas abordagens de desenvolvimento existirem.

Escalabilidade por meio de Comportamento e Enfileiramento Assíncronos

Conforme mencionado na introdução para a arquitetura orientada a serviços, é vantajoso tornar seus serviços da Web, em sua essência, assíncronos. Em razão do código extra de transporte associado aos serviços da Web e da expectativa de que os serviços, por natureza, serão remotos, é importante reduzir o tempo que um solicitante gasta aguardando respostas. Tornando uma chamada de serviço assíncrona, com uma mensagem de retorno separada, permitimos que o solicitante continue a execução enquanto o fornecedor tem uma chance de responder. Isso não quer dizer que o comportamento de serviço síncrono está incorreto, apenas que a experiência tem demonstrado que o comportamento de serviço assíncrono é recomendável, especialmente onde os custos de comunicação são altos ou a latência da rede é imprevisível.

Considere o exemplo na Figura 11.

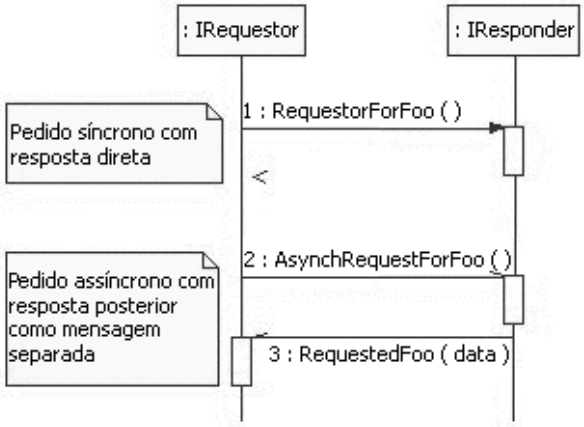

Figura 11 – Síncrono vs. assíncrono

O comportamento descrito na Figura 11 é uma grande passo para implementar serviços da Web altamente escaláveis. Tornando uma chamada de serviço assíncrona, isso permite que o fornecedor utilize vários encadeamentos em funcionamento para manipular pedidos de vários clientes. Há muito a ser feito para suportar um modo assíncrono de operação do que apenas retornar para o cliente rapidamente. Primeiramente, é necessário especificar interfaces duais; o solicitante precisará transmitir um endereço de retorno para um serviço que implemente uma interface que possa aceitar a mensagem retornada. Isso indica uma necessidade de gerenciar no estado na conversação entre as partes. Para aprender sobre os diversos métodos para fazer isso, consulte o design de sessões da Web que não se baseiam em serviços da Web.

No entanto, isso é escalável apenas até um determinado grau. Para os serviços que esperam uma carga muito alta, precisaríamos separar a parte que atende ao solicitante e a parte que atende ao próprio pedido. Isso já um padrão reconhecido, em que uma fila de mensagens é utilizada para separar uma face do serviço da implementação do serviço. O diagrama na Figura 12 mostra como o fornecedor está implementado utilizando uma fila para separar o pedido da implementação.

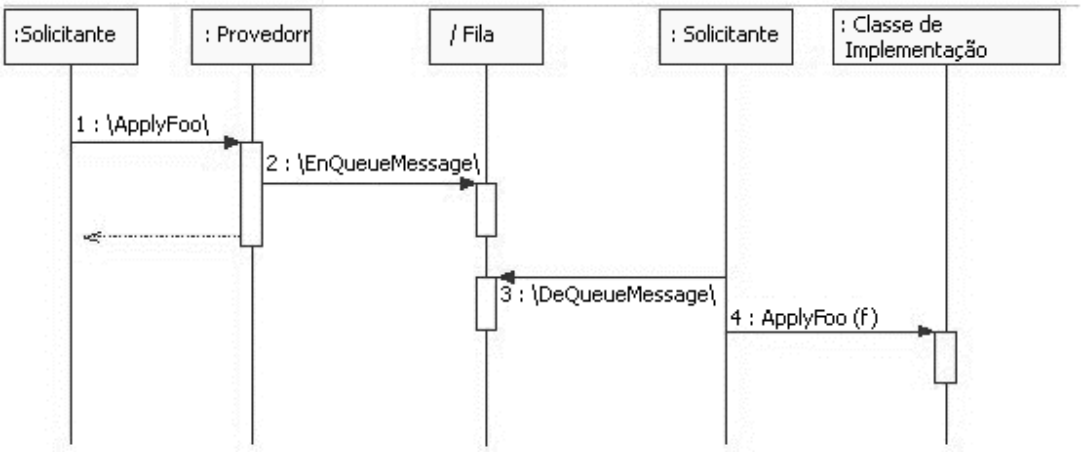

Figura 12 – Implementação enfileirada

Este padrão pode ser facilmente implementado no .NET e J2EE utilizando os serviços fornecidos por estas plataformas: MSMQ para .NET e JMS (Java Message Queue Service) ou beans orientados a mensagens para J2EE. Isso fornece ao desenvolvedor um modelo de escalabilidade mais simples; em vez de manipular um

conjunto de encadeamentos com sincronização para os pedidos, a implementação pode simplesmente incluir listeners de filas adicionais para selecionar mensagens da fila, mesmo em várias máquinas.

Locação de Informações Revisitada

Quando consideramos as informações de locação, vemos isso mais em termos do empréstimo de um livro de uma biblioteca do que da locação de uma propriedade, como casa ou carro. Implicitamente, sempre que um solicitante faz um pedido de um serviço, ele está solicitando uma cópia de algumas informações; isso é sempre fornecido com apenas uma captura instantânea do estado em um determinado momento. Agora isso pode ser um problema, a menos que seja explicitamente entendido e considerado. Uma estratégia é deixar que o fornecedor forneça o tempo de *expiração* juntamente com as informações. Alternativamente, o solicitante pode obter um "registro" com a locação (como um livro de biblioteca) que permitiria possivelmente estender a locação, perguntando se as informações ainda são válidas e, em seguida, deixar o servidor reconfigurar a locação sem ter que recuperar os dados novamente.

Assim sendo, esperar que o HTTP, SOAP ou um dos protocolos de transporte manipulem isso para nós parece ser uma questão fundamental. Poderíamos reutilizar a semântica de cache HTTP, que permite que os navegadores e firewalls armazenem as páginas em cache, mas na realidade isso não está sob controle do fornecedor e o solicitante pode não estar utilizando HTTP como um transporte. Uma opção é construir esse suporte para sua troca de documentos, de modo que as mensagens entre o solicitante e o fornecedor codifiquem as informações de locação para o cliente, conforme mostrado na Figura 13.

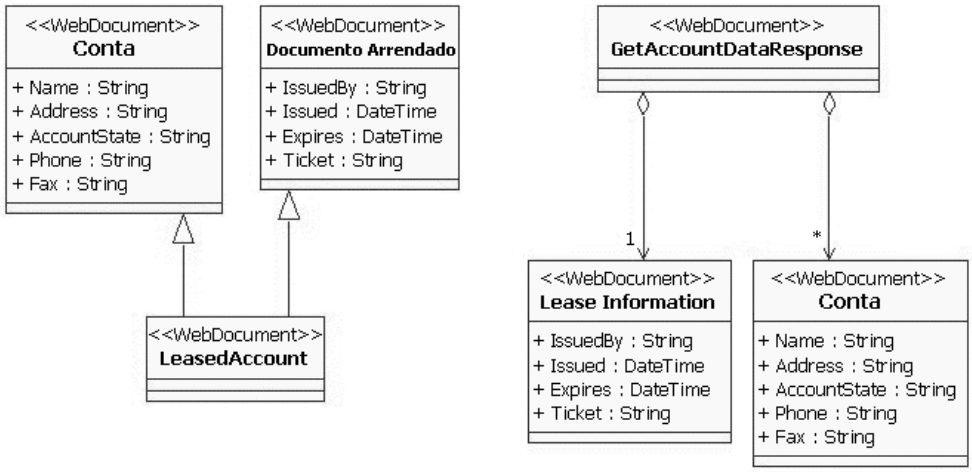

Figura 13 – Duas implementações de locação de informações

A Figura 13 demonstra duas implementações alternativas do padrão de locação de informações. A primeira demonstra o uso da herança para adaptar o documento XML Conta a um formulário especial que não seja apenas uma Conta, mas também um Documento Locado e, portanto, inclui as informações adicionais. Na segunda alternativa, as informações de locação são retornadas ao lado da conta como uma parte separada da mensagem de resposta. Ambas as abordagens são igualmente válidas, resultando em dados estruturados de modo diferente, e a opção depende muito do estilo, herança versus agregação.

Modelo de Design de Serviço da Web Resultante

A Figura 14 demonstra como o design de serviço genérico na Figura 7 pode ser modelado utilizando um perfil do UML específico para o desenvolvimento e design do serviço da Web. O perfil é muito simples e apresenta apenas dois novos *estereótipos* (extensões para a linguagem UML existente) para «WebService» e «WebDocument». Reutilizando a semântica de interface já existente na UML, podemos facilmente visualizar os aspectos publicados de um serviço, conforme definido no WSDL.

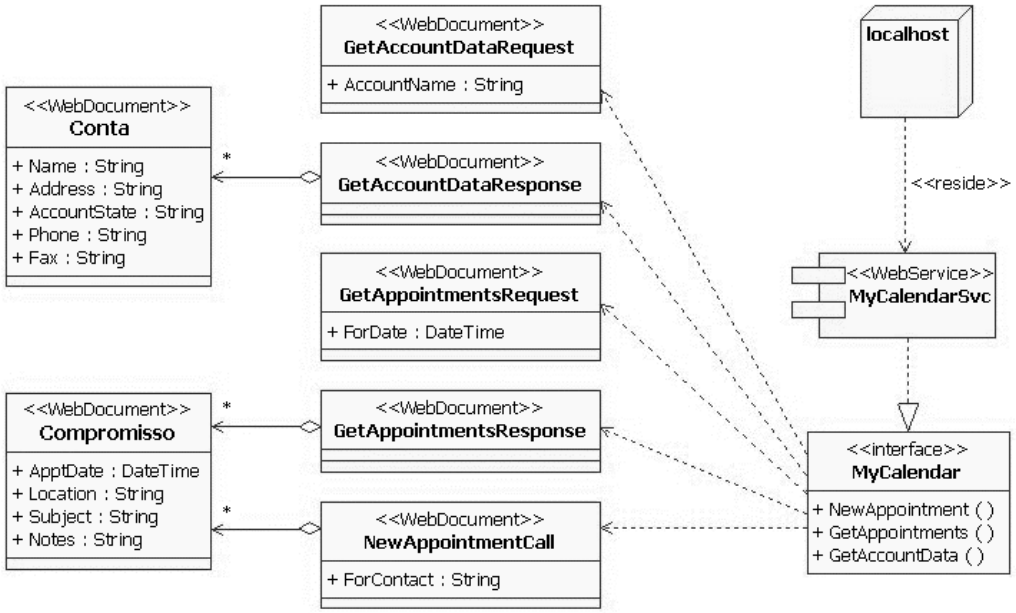

Figura 14 – Design do serviço da Web XML

A tabela a seguir demonstra como os elementos do perfil na Figura 14 estão relacionados aos artefatos no WSDL para o serviço MyCalendar.

Artefato do WSDL	Elemento da UML	Comentário
serviço	«WebService»	O serviço é representado como um componente UML que compreende uma ou mais interfaces e reside em um local específico. O relacionamento «reside» capturará as informações reais do local de URL.
portType	Interface	Cada portType é representado como uma interface UML compreendida por um ou mais serviços. O relacionamento de compreensão capturará as informações sobre ligação.
mensagem	«WebDocument»	Cada mensagem é representada como uma classe UML. Um mapeamento do Esquema XML para/da UML é necessário para modelar a mensagem e a estrutura de partes.
partir	Fim de Atributo ou Associação	Cada parte da mensagem pode ser representada como um atributo UML no «WebDocument» ou como uma associação para outro «WebDocument».
local de endereço	Nó	O nó representa o servidor no qual o serviço reside. O nó pode identificar um conjunto de serviços residentes e um serviço pode residir em mais de um nó.

É importante observar que esse método para projetar serviços da Web favorece a reutilização de ambos os documentos que definem a troca de dados e as interfaces suportadas pelos serviços. Este é um recurso-chave ao projetar soluções de escala corporativa. Preferivelmente, todos os serviços que interagem com o documento Conta fazem isso com base na mesma definição de documento. Também observamos que interfaces operacionais não publicadas, como SystemsManagement na Figura 2, podem ser definidas por especialistas dessa área e, em seguida, disponibilizadas de modo que sejam implementadas em comum em toda a solução.

Conclusão

Enquanto alguns estão esperando que os serviços da Web sejam uma revolução no segmento de mercado de software, outros dirão que os serviços da Web, embora estejam em alta, não representam nada. Como é de costume em uma nova tecnologia, a verdade está entre as duas coisas. Conforme vimos, há algumas diferenças fundamentais na arquitetura de aplicativos orientados a serviços; no entanto, as técnicas existentes de desenvolvimento baseado em componente ainda são válidas na implementação desses serviços.

É possível derivar ambas as implementações, de componente e de serviço, de um único modelo de domínio, conforme mostrado no modelo de Contato, Contato, Compromisso na Figura 6. Outras conclusões-chave são:

- Os modelos de serviços podem ser derivados dos modelos baseados em componente *existentes*.
- Padrões e designs comuns podem ser aplicados para auxiliar na transformação para os modelos orientados a serviços.

Esses padrões e designs comuns podem ser generalizados, automatizados e, por conseguinte, instanciados com uma ferramenta de transformação de modelo-para-modelo e modelo-para-código, como Rational XDE (e demonstrados na figura a seguir).

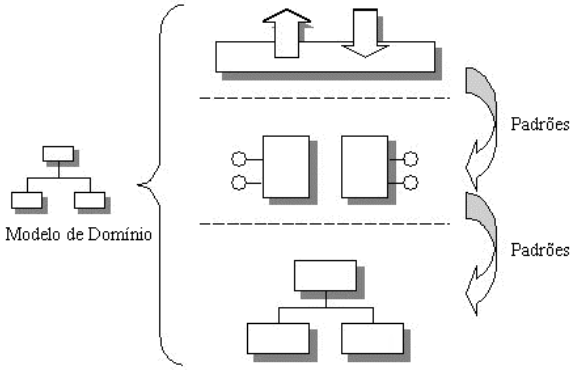

Figura 15 – Abordagem de implementação de serviço

A tomada desse caminho tem muitos benefícios: reduz a sobrecarga do desenvolvedor, aumenta a capacidade de previsão da implementação e aprimora a qualidade de reutilização de padrões comprovados. A inclusão de uma boa prática, como "Utilizar arquiteturas orientadas a serviço", em seu conhecimento e a codificação de algumas dessas boas práticas em conjuntos de ferramentas de automatização, como a Rational XDE, permitirão que você faça escolhas e tome decisões sobre dilemas apropriadas, de modo mais rápido e confiável.

Referências

[1] Para obter informações adicionais, consulte http://www.rational.com/uml/.
[2] Large-Scale, Component-Based Development de Alan W. Brown, Prentice Hall, 2000
[3] Public versus Published Interfaces de Martin Fowler, IEEE Software, março/abril de 2002 (Vol. 19, No. 2)
[4] W3C Web Services Architecture Requirements; http://www.w3.org/TR/2002/WD-wsa-reqs-20020429
[5] Web Services Security, Versão 1.0; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp
[6] Web Services Referral Protocol, Draft; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[7] XML Web Services Basics; http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-referral.asp
[8] Rational Developer Network http://www.rational.net/