# Utilizando SoaML para Modelagem e Geração de Código de Serviços em uma Abordagem SOA

Thaíssa Diirr, Leonardo Guerreiro Azevedo, Flávio Faria, Flávia Santoro, Fernanda Baião
*NP2Tec – Núcleo de Pesquisa e Prática em Tecnologia*
*Departamento de Informática Aplicada (DIA)*
*Universidade Federal do Estado do Rio de Janeiro (UNIRIO)*
*{thaissa.medeiros, azevedo, flavio.faria, flavia.santoro, fernanda.baiao}@uniriotec.br*

## Resumo

*Em um ciclo de vida de uma Arquitetura Orientada a Serviços, os serviços devem ser especificados utilizando uma linguagem de modelagem. A UML (Unified Modeling Language) é uma linguagem padrão para modelagem de sistemas de software amplamente utilizada; no entanto, ela não possui suporte nativo à modelagem de serviços. Por outro lado, a UML provê mecanismos de extensão, chamados de profiles, que correspondem a especializações e configurações da UML padrão para domínios de aplicação específicos. Este trabalho é um tutorial com o objetivo de apresentar o profile SoaML (Service Oriented Architecture Modeling Language) proposto pela OMG (Object Managment Group) para especificação de serviços. São apresentados os conceitos introduzidos à UML pelo profile e a geração de código automática a partir dos modelos produzidos.*

## Abstract

*In a SOA lifecycle, services must be specified using a modeling language. UML (Unified Modeling Language) is a widely used and standard language for modeling software systems; however it has no native support for modeling services. On the other hand, UML provides extension mechanisms, called profiles, which correspond to specializations and configurations of the standard UML for specific application domains. This work is a tutorial that presents the profile SoaML (Service Oriented Architecture Modeling Language) proposed by OMG (Object Management Group) for services specifications. The concepts introduced by the UML profile and the automatic code generation from models are demonstrated.*

## 1. Introdução

Arquitetura Orientada a Serviços (SOA) é um paradigma para a realização e a manutenção dos processos corporativos que se encontram em grandes sistemas distribuídos [8]. O objetivo de SOA é atender aos requisitos de baixo acoplamento, desenvolvimento baseado em padrões e computação distribuída independente de protocolo, mapeando os sistemas de informação da empresa para os fluxos de processos de negócios [18].

Serviços são tudo em SOA. Eles oferecem valor aos seus consumidores através de uma ou mais capacidades [16]. Com a definição e reutilização de serviços, SOA gera melhoria da produtividade e agilidade para o negócio e para a TI, possibilitando que as organizações reduzam os custos no desenvolvimento e manutenção dos sistemas envolvidos [8]. Devido a estes benefícios, SOA é amplamente utilizada.

Em uma abordagem SOA, as decisões tomadas durante a fase de análise de serviços devem ser especificadas utilizando uma linguagem de modelagem de artefatos de software, como, por exemplo, UML (Unified Modeling Language) como apresentado por [6].

A UML é uma linguagem de modelagem padrão de uso geral no domínio da engenharia de software, porém não possui suporte nativo para modelagem de serviços [5, 9, 12, 13]. Por outro lado, a especificação UML permite extensões da linguagem através de mecanismos chamados de *profiles* [15].

Os *profiles* possibilitam a personalização de modelos UML para domínios e plataformas específicos. Eles são definidos a partir um conjunto de estereótipos, propriedades e restrições que especializam e configuram a UML para um determinado domínio de aplicação, permitindo a representação de artefatos com semântica mais específica. Os *profiles* não introduzem conceitos de base e não contradizem a semântica padrão da UML.

Na literatura, *profiles* UML para SOA foram propostos, a fim de permitir a especificação de informações inerentes a serviços, seus provedores e consumidores [1, 4, 5, 7, 9, 11, 12, 13, 16]. Alguns destes *profiles* também possibilitam a geração de artefatos de implementação a partir dos modelos, como documentos XSD e WSDL [12, 13, 16].

Este trabalho tem o objetivo de apresentar o *profile* SoaML para modelagem de sistemas orientados a serviço, descrevendo seus conceitos e apresentando a geração de código a partir da modelagem. Este *profile* foi escolhido por ser a proposta de padrão da OMG, tendo formalismos e conceitos bem especificados e ferramental de apoio para seu uso.

Este tutorial está organizado da seguinte forma. A seção 1 é a presente introdução. Na seção 2, o *profile* SoaML é introduzido. Na seção 3, é apresentado o domínio do exemplo utilizado na modelagem com o *profile* e são descritos os conceitos em SoaML. Na seção 4, a geração automática de documentos a partir da modelagem é apresentada. Na seção 5, é apresentada a conclusão do trabalho e, na seção 6, as referências bibliográficas

## 2. O profile SoaML

A especificação SoaML (*Service Oriented Architecture Modeling Language*) foi criada pela OMG (*Object Management Group*) e descreve um *profile* e um metamodelo que estendem a UML 2.0 com o objetivo de apoiar as atividades de modelagem e projeto de serviços em SOA. O *profile* também visa possibilitar a geração automática de artefatos derivados em uma abordagem de desenvolvimento orientado por modelos (*Model-Driven Development* – MDD) [16]. SoaML suporta requisitos de modelagem de SOA possibilitando: especificação de serviços, incluindo suas capacidades funcionais providas, capacidades que os consumidores devem possuir, fluxo de interação para o serviço, informações de serviço trocadas entre consumidores e provedores; especificação de dependências entre serviços; especificação de regras para usar e prover serviços; especificação de consumidores e provedores, quais requisições e serviços eles consumem e provêem e como eles estão conectados.

Como, em uma abordagem SOA, os serviços modelados devem estar de acordo com o processo de negócio que apóia a organização, neste trabalho, adotamos a sequência de passos para a modelagem dos serviços apresentada a seguir. Serviços são parte de processos de negócio e, por isso, é importante que se conheça os processos para obter um alinhamento entre a Tecnologia da Informação e o negócio da organização [8].

1. Modelagem do processo de negócio que se deseja automatizar, a fim de definir as necessidades do negócio. A princípio, este passo pode ser modelado sem utilizar o *profile* SoaML. Ele tem o objetivo de explicitar os insumos para a modelagem de serviços. O modelo de processos de negócio pode ser elaborado utilizando diagrama de atividades da UML (Unified Modeling Language) [3], BPMN (Business Process Management Network) [17], EPC (Event-Driven Process Chain) [20].
2. Detalhamento das capacidades existentes no processo de negócio;
3. Detalhamento dos tipos de dados necessários e das mensagens trocadas entre os serviços;
4. Especificação das interfaces dos serviços;
5. Especificação dos contratos dos serviços;
6. Detalhamento do fluxo de interação entre consumidor e provedor em um contrato de serviço;
7. Especificação dos participantes com seus pontos de serviço e de requisição;
8. Especificação da arquitetura de serviços do processo.

Estes itens serão explicados na próxima seção.

## 3. Modelando serviços com SoaML

SoaML introduz conceitos necessários para modelagem de serviços em uma Arquitetura Orientada a Serviços ao padrão de modelagem UML. Estes conceitos são explicados a seguir de acordo com a especificação do *profile* SoaML[16]. Para facilitar o entendimento, os conceitos são apresentados sendo aplicados diretamente em um mini-mundo fictício. Dessa forma, a seção 3.1 apresenta o exemplo que será utilizado nas explicações do *profile*, enquanto que as seções seguintes apresentam a explicação de cada conceito aplicado ao exemplo.

### 3.1. Mini-mundo utilizado como exemplo

Neste tutorial foi considerado um mini-mundo do processo de ordem de compra retirado da especificação do padrão SoaML e modelado com a utilização do software Modelio Case Tool Enterprise Edition [14]. A Figura 1 ilustra este processo em um diagrama de atividades da UML padrão. O processo inicia quando uma ordem de compra é recebida. A partir daí, em paralelo, o cálculo do preço é iniciado e a remessa do produto e a programação da produção são requisitados. O preço inicial é calculado com base nos produtos encomendados, mas não pode ser concluído, pois a fatura total depende do custo de transporte. Assim, após a conclusão da requisição da remessa do produto e do cálculo inicial do preço, pode ser realizada a conclusão do cálculo do preço do produto e, em seguida, a fatura é processada. Ao concluir a requisição da remessa do produto, o processamento da programação da remessa é realizado. Após o processamento do agendamento da remessa e a requisição do agendamento da produção, o agendamento da remessa é enviado.

As figuras de exemplo apresentadas neste tutorial são baseadas neste processo.

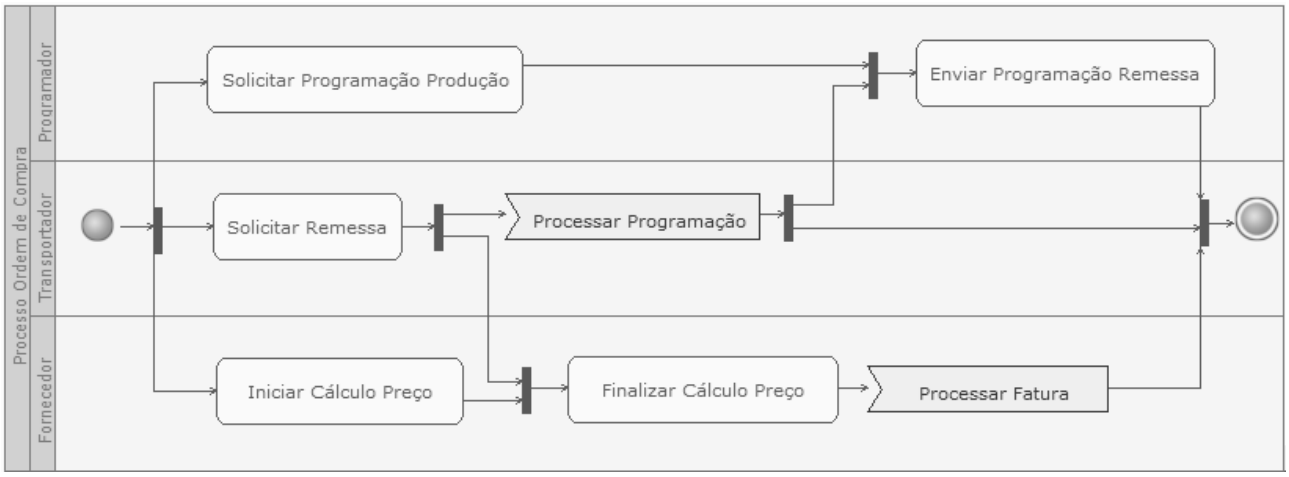


Figura 1. **Processo de negócio de ordens de compra**

## 3.2. Capacidade

Uma capacidade (*«Capability»*) é a habilidade de agir e produzir resultados que atingem objetivos. Assim, capacidades identificam ou especificam um conjunto coesivo de funções ou recursos que um serviço provido por um ou mais participantes pode oferecer. Logo, uma capacidade consiste em um conjunto de funcionalidades de um serviço. Elas podem ser vistas de duas perspectivas: capacidades que um participante possui e podem ser expostas como serviço, e capacidades que uma organização necessita e podem ser utilizadas para identificar serviços candidatos.

Uma capacidade pode depender de outras capacidades para ser fornecida.

As capacidades são especificadas sem levar em conta como um serviço particular deve ser implementado e oferecido posteriormente aos consumidores por um provedor. Assim, elas são utilizadas para identificar serviços necessários, organizá-los a fim de verificar as necessidades de uma área, analisar o modo como estão relacionados e como eles podem ser combinados para formar alguma capacidade maior, antes de alocá-los a provedores particulares.

A Figura 2 ilustra as capacidades identificadas a partir do processo de ordem de compra. As capacidades Fatura, Programação e Remessa representam as raias de mesmo nome no processo. As operações contidas nas capacidades não precisam ser necessariamente iguais às atividades presentes nas raias correspondentes. Neste exemplo, por simplificação e para focar no uso do *profile* SoaML, foi decidido definir a granularidade das capacidades com base nas raias do processo. Logo, novas operações podem ser adicionadas e operações existentes podem ser renomeadas e excluídas. A capacidade Compra representa o processo como um todo e, por isso, possui uma relação de uso com as outras capacidades.

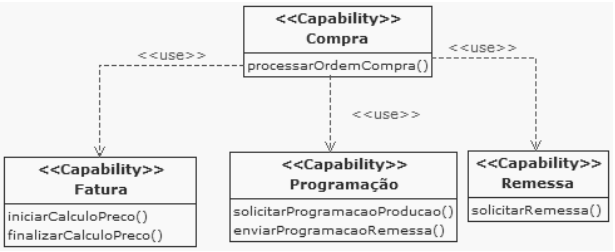


Figura 2. **Capacidades do processo de ordem de compra**

## 3.3. Tipos de dados e Mensagens

Após definir as capacidades, o modelo de dados do domínio pode ser especificado da mesma forma como em diagramas de classe da UML padrão. Assim, podem ser utilizados tipos simples disponíveis na UML ou

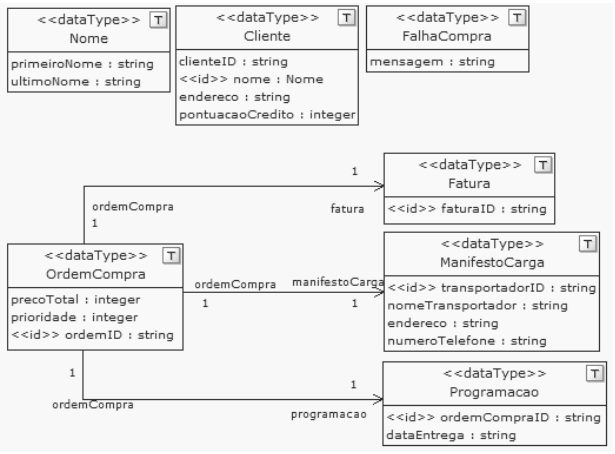


Figura 3. **Modelo de dados do processo de ordem de compra**

classes e tipos de dados (*«DataType»*) definidos pelo usuário. A Figura 3 apresenta o modelo de dados definido para o processo de ordem de compra. Por exemplo, o tipo de dados OrdemCompra é composto dos atributos preço total, prioridade e atributo ordemId (do

tipo (*«Id»*) que pode ser usado para distinguir as instâncias), além de estar associado a uma fatura, um manifesto de carga e um agendamento.

Além do modelo de dados, é possível definir também os tipos de mensagens (*«MessageType»*), que são os tipos de dados que representam a informação trocada entre requisições de consumidores e os serviços de provedores. Essa informação consiste de dados passados para e/ou retornados a partir de uma invocação de uma operação de um serviço. Assim, os tipos de mensagens podem ser usados para agregar as entradas, saídas e exceções das operações de serviços. A Figura 4 apresenta os tipos de mensagens definidos para o processo de ordem de compra.

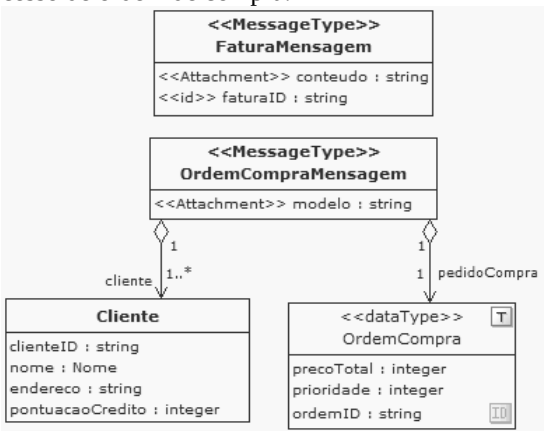


Figura 4. **Tipos de mensagens do processo de ordem de compra**

Os tipos de mensagem não possuem identidade e, por isso, deveriam sempre possuir tipos de dados como atributos. Porém, é permitido que as mensagens possuam atributos cujo tipo é uma classe. Neste caso, a identidade implícita pela classe não é considerada no tipo de mensagem e a classe é tratada como um tipo de dados.

Tipos de mensagens podem possuir anexos (*«Attachment»*) além de atributos. Os anexos não são parte direta da mensagem em si e podem ser usados para indicar parte dos dados do serviço que podem ser acessados separadamente, reduzindo os dados enviados entre consumidores e fornecedores, exceto quando necessário.

Existem dois estilos para definição dos parâmetros das operações de serviços: centrado a documento (ou mensagem) e RPC (*Remote Procedure Call*). No estilo centrado a documento são usados tipos de mensagens como parâmetros e cada operação só pode ter um parâmetro de entrada, um de saída e um de exceção. Já no estilo RPC, não existe um número máximo para quantidade de parâmetros, mas seus tipos estão restritos a tipos primitivos ou tipos de dados. Para o processo de ordem de compra utilizado como exemplo neste tutorial foi utilizado o estilo RPC para definir os parâmetros dos serviços.

## 3.4. Consumidor e Provedor

O conceito de provedores e consumidores é fundamental em uma arquitetura orientada a serviços. O consumidor solicita um serviço do provedor que usa suas capacidades para atender à solicitação do serviço.

A interação entre o provedor e o consumidor é governada por um contrato de serviço que os relaciona. O consumidor é normalmente aquele que inicia a interação de serviço. Já o provedor é normalmente aquele que responde a interação de serviço.

Os estereótipos de consumidor (*«Consumer»*) e provedor (*«Provider»*) são aplicados às interfaces dos serviços, de forma a modelar, respectivamente, interfaces disponibilizadas por consumidores de serviço e disponibilizadas por provedores de serviço. A seguir, será demonstrada a aplicação destes estereótipos nas interfaces de serviço.

## 3.5. Interfaces

Uma interface de um serviço define as operações que serão disponibilizadas pelos provedores deste serviço aos seus consumidores. Assim, a interface é a assinatura de um serviço que descreve suas operações, os parâmetros de entrada, parâmetros de saída e possíveis exceções [8].

As interfaces são informações importantes aos consumidores potenciais para determinar se um serviço responde suas necessidades e como usá-lo em caso positivo. Elas também fornecem a informação que os provedores precisam para implementar o serviço.

Em SoaML, as interfaces podem ser interfaces simples da UML padrão ou interfaces do tipo *ServiceInterface* adicionadas pelo SoaML. A partir das capacidades criadas, foram modeladas as interfaces dos serviços para o exemplo do processo de ordem de compra, as quais são apresentadas a seguir.

### 3.5.1. Interfaces simples

As interfaces simples são usadas para especificar serviços cuja interação entre consumidor e provedor é *one-way*, ou seja, o consumidor apenas invoca as operações definidas na interface. Neste tipo de interação não é requerido um protocolo (ou fluxo de interação) para a comunicação entre as partes consumidora e provedora, ou seja, o serviço é unidirecional onde o consumidor invoca o serviço provido e não há *callback* do servidor. Um *callback* é uma operação que o provedor invoca no consumidor para retornar a resposta da execução do serviço. Este tipo de interface pode ser usado com consumidores desconhecidos e o provedor não faz nenhuma suposição sobre eles.

Uma porta de provisão ou requisição de serviços pode ser representada através de uma interface UML, de forma que um participante pode prover ou consumir as operações definidas na interface através desta porta.

A Figura 5 ilustra a interface de serviço Programacao que possui as operações da capacidade de mesmo nome. Esta capacidade foi exposta como uma interface simples, pois nenhuma de suas operações exige *callback* do servidor. Isto é identificado a partir do processo de

negócio, onde as atividades que geraram a capacidade são representadas no diagrama de atividades como ações do tipo *action*. Além disso, a interface possui o estereótipo de provedor (*«Provider»*), já que as interfaces em SoaML são definidas do ponto de vista dos provedores de serviços.

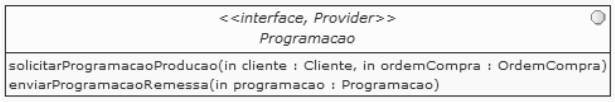


Figura 5. **Interface simples para o serviço Programacao**

A interface para a capacidade Compra também foi modelada através de uma interface simples. Ela representa o processo como um todo e é responsável apenas por invocar as outras capacidades necessárias para a automatização do processo.

### 3.5.2. Interfaces de serviço

Assim como uma interface UML, uma interface de serviço (*«ServiceInterface»*) pode ser o tipo de uma porta de provisão ou requisição de serviço. Porém, este tipo de interface tem como característica adicional permitir a especificação de um serviço bi-direcional, onde tanto o provedor quanto o consumidor têm responsabilidades no envio e recebimento de mensagens e eventos. Neste tipo de interface é esperada uma interação com o consumidor para que o serviço seja provido e, assim, existem *callbacks* do provedor ao consumidor como parte da comunicação entre eles.

Interfaces de serviço *ServiceInterface* definem as operações providas através do serviço e as necessidades a serem atendidas pelos consumidores ao invocar o serviço. Elas representam um acordo formal entre consumidores e provedores que deve ser atendido.

As interfaces *ServiceInterface* também são definidas da perspectiva do provedor do serviço, especificando a interface que o provedor oferece e a que ele espera do consumidor. As interfaces providas e requeridas por uma *ServiceInterface* são interfaces simples da UML padrão e são, respectivamente, realizadas ou utilizadas por ela. As interfaces realizadas especificam o valor provido e as mensagens a serem recebidas pelo provedor e correspondem às mensagens enviadas pelo consumidor. Já as interfaces utilizadas definem o valor requerido e as mensagens que serão recebidas pelo consumidor e enviadas pelo provedor.

Um relacionamento de exposição (*«Expose»*) pode ser definido entre uma interface *ServiceInterface* e uma capacidade. Isto significa que a interface provê as operações de forma consistente com a capacidade exposta. Alternativamente, capacidades podem realizar *ServiceInterfaces* usando o relacionamento de realização da UML padrão. Nesta abordagem, a interface é a especificação e a capacidade implementa essa especificação. Nesses dois tipos de relacionamento, a interface *ServiceInterface* não precisa ter as mesmas operações e propriedades que a capacidade.

A Figura 6 apresenta a interface *ServiceInterface* Fatura. Esta interface expõe a capacidade de mesmo nome e foi modelada como uma interface do tipo *ServiceInterface*, pois algumas operações são providas e o processamento da fatura é requerido. No diagrama de atividades do processo de negócio, podem ser indicadas as operações que os serviços precisam invocar dos seus consumidores a partir de atividades do tipo *Accept event action* (para chamadas assíncronas) ou *Accept call action* (para chamadas síncronas), desde que a linguagem permita incluir esta marcação. No caso deste tutorial, foi utilizado o diagrama de atividades para representar o processo empregando SoaML. Dessa forma, neste diagrama foi possível acrescentar estas informações. No modelo de processos apresentado na Figura 1, atividades deste tipo são representadas pelas atividades ProcessarFatura e ProcessarProgramacao. Como as interfaces de serviço *ServiceInterface* são definidas da perspectiva do provedor de serviços, é construída uma interface de serviço FaturaServico que provê (ou realiza) a interface Fatura com estereótipo de provedor (*«Provider»*) e utiliza a interface ProcessamentoFatura com estereótipo de consumidor (*«Consumer»*). Também foi construída a interface conjugada, iniciada por "~" (til), que representa a perspectiva do consumidor do serviço. Uma interface conjugada apresenta as mesmas operações da interface provida, porém suas interfaces simples utilizadas e realizadas são inversas.

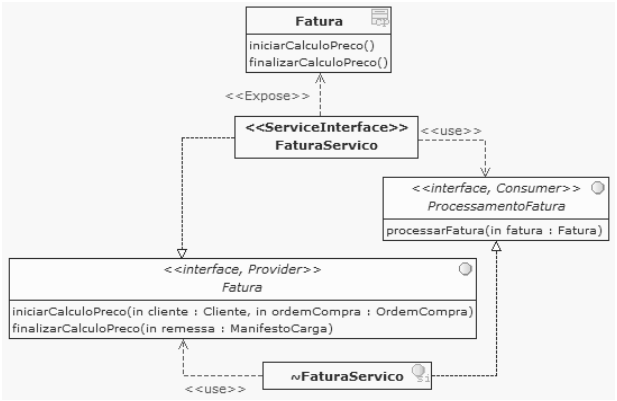


Figura 6. **Interface *ServiceInterface* para o serviço Fatura**

A interface de serviço RemessaServico, ilustrada na Figura 7, requer o processamento do agendamento do embarque e expõe a capacidade Remessa. Assim, ela foi construída de forma análoga à interface FaturaServiço, provendo a interface *Remessa* e utilizando a interface ProcessamentoProgramacao, além de apresentar também a interface conjugada.

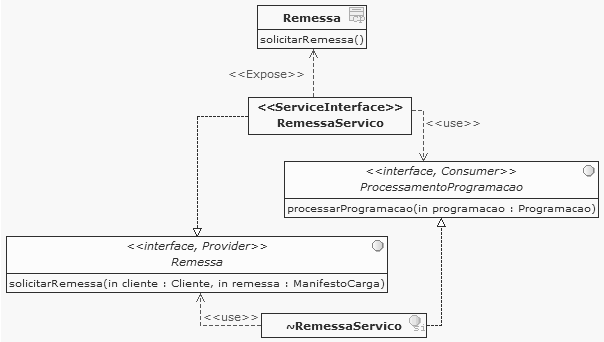


Figura 7. **Interface *ServiceInterface* para o serviço Remessa**

### 3.5.3. Contratos de serviço

Com as interfaces dos serviços criadas, o próximo passo é a criação dos contratos de serviço.

Uma parte chave de um serviço é o seu contrato, que especifica como consumidores e provedores trabalham juntos para atingir algum valor. O contrato representa um acordo entre consumidor e provedor, definindo como um serviço é prestado e consumido.

Um contrato de serviço define os termos, condições, interfaces e coreografia que os participantes que interagem devem concordar para que o serviço seja provido. No contrato são especificados os papéis que cada participante assume no serviço (provedor ou consumidor) e as interfaces que eles implementam para assumir estes papéis. Um papel define a função básica (ou conjunto de funções) que uma entidade pode executar em um determinado contexto. Assim, o contrato define uma ligação entre provedor e consumidor, especificando quais informações, produtos, bens de valor e obrigações irão fluir entre eles.

A especificação de um contrato não se preocupa com sua realização ou implementação. Dessa forma, os contratos descrevem como as partes concordam para prover ou usar o serviço, sem considerar como estas partes implementam seu papel no serviço através de seus processos internos.

A base para um contrato de serviço é uma colaboração da UML focada nas interações realizadas para a provisão de um serviço. Cada papel ou parte envolvida em um contrato possui um nome e um tipo, que pode ser uma interface de serviço simples ou uma interface *ServiceInterface*. As interfaces podem possuir estereótipos de provedores («*Provider*») ou consumidores («*Consumer*»). Caso possuam estereótipos de provedores, os papéis apresentaram a indicação de provedores ○. Já se possuírem estereótipos de consumidores, os papéis serão indicados com ◔. Estas interfaces são também os tipos das portas de serviço e de requisição em um participante, o que obriga o participante a estar habilitado a assumir o papel correspondente no contrato do serviço.

Na maioria dos casos, um contrato de serviços especifica dois papéis, sendo um consumidor e um provedor, mas existem casos onde mais papéis podem ser especificados. Este tipo de contrato é chamado de contrato multi-parte.

Um serviço pode ser construído para atender a um contrato de serviço já estabelecido antes ou o contrato pode ser definido após a construção do serviço correspondente. O importante é que no momento em que o serviço é invocado, um contrato de serviço deve ser seguido.

### 3.5.4. Contratos de serviço para interfaces simples

Para serviços que possuem interface definida como uma interface simples da UML padrão, não é requerido um fluxo de informação entre o consumidor e o provedor. Como o consumidor apenas invoca o serviço e o provedor não realiza *callback* para ele, os consumidores podem ser desconhecidos. Assim, ao definir um contrato para um serviço cuja interface é simples, o tipo do papel consumidor não é definido. Apenas será definido que existe um papel que é consumidor. Já para o papel provedor seu tipo é definido normalmente.

A Figura 8 apresenta o contrato definido para o serviço Programacao. Como esse serviço possui interface simples, seu contrato não especifica o consumidor. Já o papel provedor possui o nome programador e seu tipo é definido pela interface do serviço Programacao. Como esta interface foi definida como provedora, o papel aparece com a indicação de provedor.

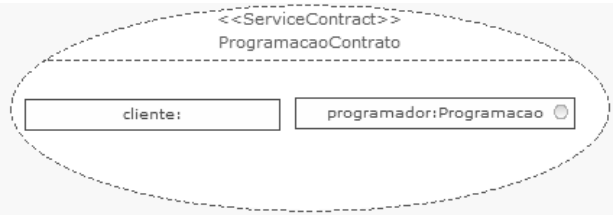


Figura 8. **Contrato de serviço para a interface Programacao**

### 3.5.5. Contratos de serviço para interfaces de serviço

Para serviços que possuem interface definida como *ServiceInterface*, é requerido um fluxo de informação entre o consumidor e o provedor. Assim, os consumidores devem ser conhecidos, pois uma interação com eles é esperada para que o serviço seja provido e existem *callbacks* do provedor como parte da comunicação entre eles. Dessa forma, ao definir um contrato para um serviço cuja interface é uma *ServiceInterface*, deve-se definir o tipo do papel provedor e o do papel consumidor.

A Figura 9 apresenta o contrato definido para o serviço Remessa. Como esse serviço possui interface *ServiceInterface*, seu contrato especifica o consumidor e o provedor. Os tipos dos papéis consumidor e provedor são definidos de acordo com a interface usada e a interface provida pela *ServiceInterface*. Assim, o papel provedor possui nome transportador, seu tipo é definido

pela interface utilizada Remessa e ele aparece com a indicação de provedor. Já o papel consumidor possui o nome solicitante, seu tipo é definido pela interface realizada ProcessamentoProgramacao e ele apresenta a indicação de consumidor.

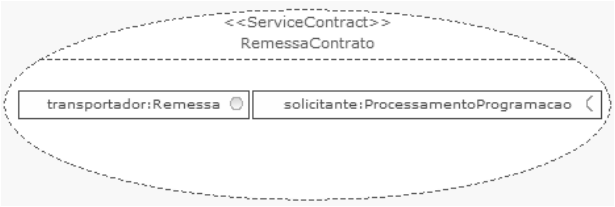


Figura 9. **Contrato de serviço para a interface RemessaServico**

O contrato de serviço para a interface FaturaServico foi construído da mesma forma, pois esta interface também é do tipo *ServiceInterface* e especifica seu consumidor. Assim, o contrato ContratoFatura possui dois papéis: o papel provedor fornecedor cujo tipo é a interface realizada Fatura e o papel consumidor solicitante cujo tipo é a interface utilizada ProcessamentoFatura.

### 3.5.6. Contratos de serviço multi-parte

Contratos de serviços multi-parte são aqueles com mais de dois participantes - embora este seja um caso menos comum, ele representa muitos negócios e interações da tecnologia.

No contexto do processo de ordem de compra utilizado neste tutorial não há nenhum caso deste tipo de contrato. Porém, como exemplo deste tipo de contrato pode-se pensar em um serviço de compras onde há um intermediário. Neste exemplo, existem três participantes: comprador, agente intermediário e vendedor. O serviço inicia com um depósito feito pelo comprador para um agente intermediário que notifica o vendedor. A entrega da compra é feita pelo vendedor e aceita pelo cliente, ou o cliente envia uma reclamação ao agente intermediário que a encaminha ao vendedor. O vendedor pode se justificar. Esse processo se repete até que o agente intermediário conclua a transação e faça o pagamento para o vendedor (se entrega foi feita), ou restituição ao comprador (se a entrega não foi feita). Este serviço é um contrato multi-parte porque o comprador também interage com o vendedor, quando o vendedor faz a entrega diretamente ao comprador; e exceto para a entrega, a interação entre o comprador e o vendedor é mediada pelo agente intermediário.

### 3.5.7. Fluxos de interação de contratos

Em contratos de serviço para interfaces de serviço do tipo *ServiceInterface*, além dos papéis que os participantes assumem nos contratos também é importante que o fluxo de interação necessário entre consumidores e provedores para provisão do serviço seja especificado. Este fluxo é chamado de protocolo da comunicação e representa a coreografia entre consumidor e provedor, ou seja, as interações válidas entre eles. Os fluxos de interação não são definidos para contratos de serviços de interface simples.

A coreografia é a especificação do que é transmitido e da sequência em que esta transmissão ocorre para que um serviço possa ser provido. Ela define o que acontece entre o provedor e o consumidor sem definir seus processos internos.

O fluxo de interação de um contrato de serviço é um comportamento UML que pode ser modelado como um diagrama de interação ou um diagrama de atividade. Este fluxo define o que deve acontecer entre os papéis do contrato, tal como definido pelas interfaces correspondentes, e como os participantes que assumem os papéis (sem especificar quem são) devem proceder.

Para o contrato de serviço RemessaContrato foi modelado um diagrama de sequência que especifica seu protocolo de comunicação e é ilustrado na Figura 10. Neste contrato, é especificado que o consumidor ProcessamentoProgramacao invoca o método solicitarRemessa do serviço Remessa o qual, ao concluir a execução, invoca o método processarProgramacao do consumidor. Pelo diagrama, sabe-se que os métodos solicitarRemessa e processarProgramacao são assíncronos porque a ponta da setas das mensagens transmitidas são abertas.

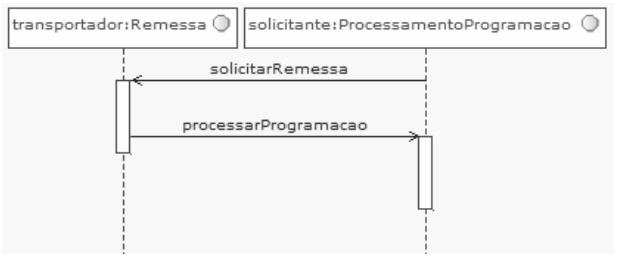


Figura 10. **Fluxo de interação do contrato RemessaContrato**

O protocolo de interação para o contrato FaturaContrato foi modelado utilizando um diagrama de atividades e é apresentado na Figura 11. Neste diagrama é apresentado que, inicialmente, o papel provedor fornecedor do tipo Fatura executa o método "iniciar calculo preco" e, em seguida, o método "finalizar calculo preco". Por fim, o papel consumidor cujo tipo é ProcessamentoFatura executa o método "processar fatura".

Para a interface simples Programacao não foi definido um protocolo de comunicação. Isto ocorre devido às interfaces simples não requererem fluxo de interação entre consumidor e provedor.

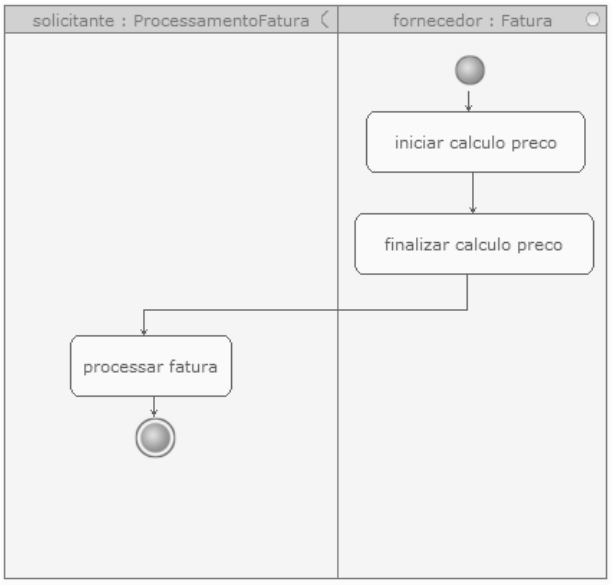


Figura 11. **Fluxo de interação do contrato FaturaContrato**

## 3.6. Portas

Uma porta representa um ponto de conexão em um participante, onde um serviço é provido ou consumido por ele. Ela é um ponto visível a partir do qual as solicitações dos consumidores são enviadas aos provedores e através do qual eles interagem para atingir um objetivo.

As portas pertencem aos participantes e podem ser portas de serviço ou de requisição conforme descrito a seguir.

### 3.6.1. Portas de serviço

Portas de serviço («*Service*») representam serviços que participantes provêem e oferecem para consumo de outros participantes. Através de uma porta de serviço, um participante com capacidade de prover um serviço se comunica com um consumidor que possui necessidades compatíveis.

Uma porta de serviço pode ser do tipo de uma interface simples ou de uma interface *ServiceInterface*. Dessa forma, as operações providas através de uma porta de serviço são aquelas definidas na interface correspondente.

Para que o serviço oferecido através de uma porta seja provido, seu contrato deve ser cumprido. Isto inclui as interfaces providas (capacidades do participante), as interfaces requeridas (o que os consumidores devem fornecer para utilizar as capacidades) e qualquer protocolo de comunicação para interação entre consumidor e provedor.

Se o tipo de uma porta de serviço for uma interface *ServiceInterface*, então a interface provida através da porta será aquela realizada pela *ServiceInterface*, enquanto a interface requerida será aquela usada. Já se o tipo da porta for uma interface simples, a interface provida será esta própria interface e não haverá interfaces requeridas e nem protocolos de comunicação a serem cumpridos.

### 3.6.2. Portas de requisição

Portas de requisição («*Request*») representam o consumo, por um participante, de um serviço provido por outros participantes. Através de uma porta de requisição, um participante consumidor satisfaz suas necessidades através da utilização do serviço disponibilizado pelos provedores.

Assim como as portas de serviço, uma porta de requisição também pode ser do tipo de uma interface simples ou de uma interface *ServiceInterface*, definindo a requisição das operações providas pela interface correspondente. Também é necessário que o contrato do serviço seja atendido para que o serviço possa ser consumido através da porta.

Se o tipo da porta for uma interface simples, a interface requerida será esta própria interface e não haverá interfaces providas e nem protocolos de comunicação a serem cumpridos.

Porém, se o tipo de uma porta de requisição for uma interface *ServiceInterface*, ela deverá corresponder à interface conjugada àquela que provê o serviço requisitado. Isto porque as portas de requisição definem o consumo do serviço. Dessa forma, a interface provida e requerida através da porta serão, respectivamente, aquelas realizada e usada pela interface conjugada.

## 3.7. Participantes

Após a especificação dos contratos de serviço, os participantes foram definidos. Os participantes são entidades que podem representar pessoas, organizações, unidades organizacionais ou sistemas de informação que provêem e/ou utilizam serviços. Os participantes correspondem às raias do processo de negócio utilizado.

Um participante é um provedor se ele provê um serviço. De forma análoga, um participante é consumidor se ele utiliza um serviço. Não há limite para quantidade de serviços que um participante pode prover e/ou consumir.

Consumidor e provedor de serviço são papéis que os participantes podem assumir. Assim, um participante pode ser provedor em alguns serviços e consumidor em outros, dependendo das capacidades providas e das necessidades que possuem para realizar suas capacidades.

Um participante não pode realizar ou utilizar interfaces diretamente, mas sim através de portas de serviço ou portas de requisição onde os serviços serão oferecidos ou consumidos respectivamente.

Para o nosso exemplo do processo de ordem de compra foram definidos os participantes Programador, Transportador, Fornecedor e ProcessadorOrdem. O participante Programador é apresentado na Figura 12 e possui uma porta de serviço («*Service*») chamada de "programacao". Como o tipo desta porta é a interface

simples Programacao, esta mesma interface aparece como interface provida através da porta e não existem interfaces requeridas.

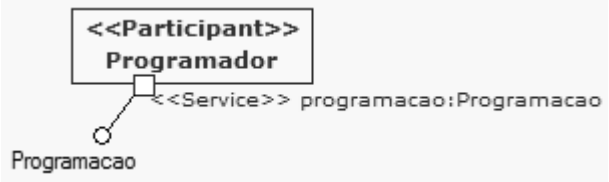


Figura 12. **Participante Programador**

O participante Transportador também possui uma porta de serviço chamada de "remessa". Porém, o tipo desta porta de serviço é a *ServiceInterface* RemessaServico. Assim, a porta de serviço possui a interface provida Remessa e a interface requerida ProcessamentoProgramacao.

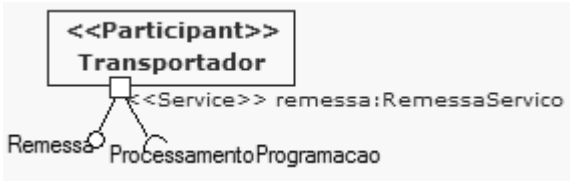


Figura 13. **Participante Transportador**

O participante Fornecedor é definido de forma análoga ao Transportador. Isto porque ele possuiu a porta de serviço fatura cujo tipo é a *ServiceInterface* FaturaServico. Dessa forma, a porta de serviço possui a interface provida Fatura e a interface requerida ProcessamentoFatura. Já o participante ProcessadorOrdem, ilustrado na Figura 14 , também possui portas de requisição. Este participante provê a interface Compra, que representa o processo de ordem como um todo, e para isso precisa invocar as outras interfaces definidas para atender as capacidades do processo. Este participante possui:

- Ponto de provisão de serviço chamado compra cujo tipo é a interface simples Compra;
- Ponto de requisição de serviço chamado programacao cujo tipo é a interface simples Programacao;
- Ponto de requisição de serviço chamado fatura cujo tipo é a interface *ServiceInterface* conjugada de FaturaServico;
- Ponto de requisição de serviço chamado remessa cujo tipo é a interface *ServiceInterface* conjugada de RemessaServico.

Pode-se perceber através deste participante que no caso de *ServiceInterfaces*, as interfaces providas e requeridas por um ponto de requisição são inversas àquelas providas e requeridas por um ponto de serviço. Por exemplo, o participante Transportador possui um ponto de serviço que usa a interface ProcessamentoProgramacao e provê Remessa. Já o participante ProcessadorOrdem possui um ponto de requisição da interface conjugada, provendo a interface ProcessamentoProgramacao e utilizando Remessa.

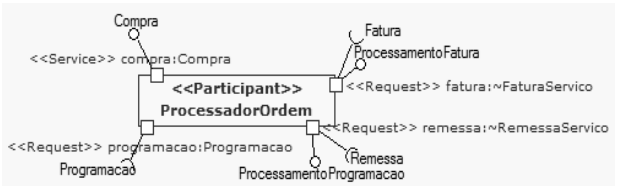


Figura 14. **Participante ProcessamentoProgramacao**

## 3.8. Arquitetura de serviço

Para implantação de uma arquitetura orientada a serviços é necessário o entendimento de como as pessoas, organizações e sistemas trabalham juntos ou colaboram para um propósito. Uma arquitetura de serviços («*ServicesArchitecture*») mostra um conjunto de serviços em um contexto e apresenta como os participantes trabalham juntos para apoiar os objetivos da comunidade, de processos, de conjunto de sistemas ou da organização. Arquiteturas de serviço são redes de papéis de participantes provendo e consumindo serviços para atingir um objetivo. Elas definem os requisitos para os papéis de participantes e realizações de serviço que atendem a estes papéis.

Expressando o uso dos serviços, a arquitetura de serviços implica em algum grau de conhecimento sobre as dependências entre os participantes no contexto que define. Cada uso de um serviço em uma arquitetura é representado pelo uso de um contrato de serviço ligado aos participantes que assumem um papel do contrato.

O uso de uma arquitetura de serviços é opcional, mas é recomendado para apresentar uma visão de alto nível de como um conjunto de participantes trabalha junto para algum propósito. A arquitetura pode representar o domínio do processo de negócio utilizado para identificação dos serviços que irão automatizá-lo.

Os mesmos serviços e participantes podem ser utilizados em várias arquiteturas, fornecendo reuso.

Uma arquitetura de serviços pode ser modelada em dois níveis de granularidade. O primeiro nível corresponde à arquitetura de serviços do domínio, uma visão de alto nível apresentando como participantes independentes trabalham em conjunto para realizar um propósito. O segundo nível corresponde à arquitetura de serviços de um participante e especifica como as partes de um participante, por exemplo, departamentos dentro de uma organização, trabalham juntos para prover serviços.

No nosso exemplo do processo de ordem de compra, o serviço Compra contém uma única operação que representa a provisão do processo como um todo. Os detalhes deste serviço foram especificados através da arquitetura de serviço ilustrada na Figura 15. Esta ilustração permite a visualização dos participantes que interagem com o objetivo de automatizar o processo de ordem de compra, das relações entre eles através dos

contratos e dos papéis que estes participantes desempenham nos contratos.

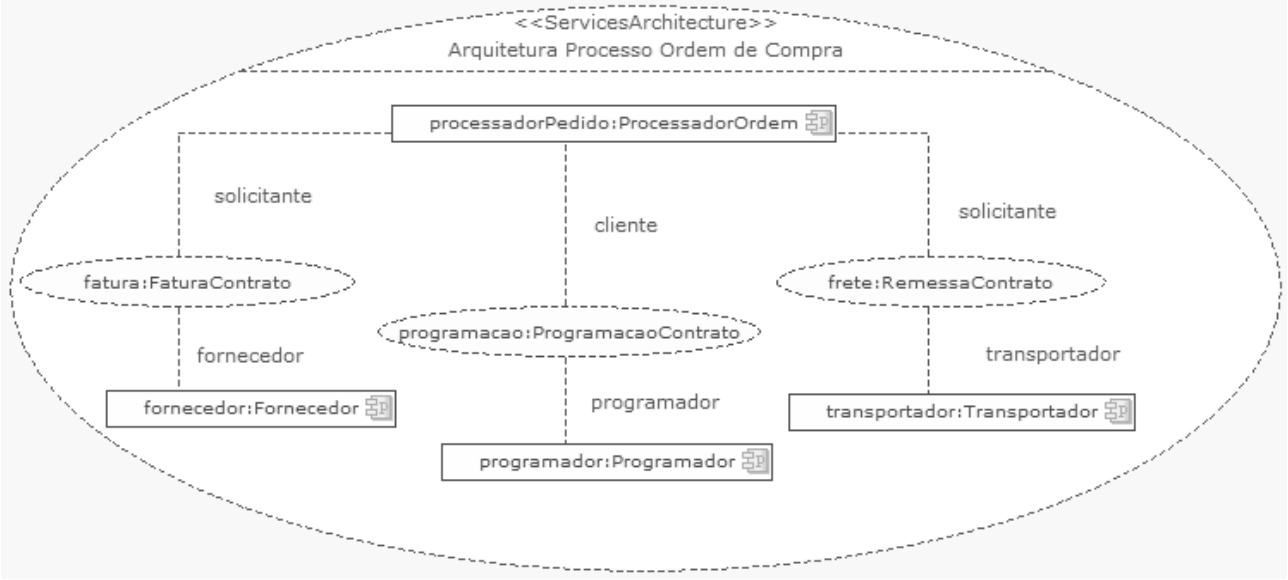


Figura 15. **Arquitetura de serviços do processo de ordem de compra**

## 4. Geração automática a partir de modelos

SoaML apóia atividades de modelagem e projeto de serviços de forma adequada a possibilitar uma abordagem de desenvolvimento orientado por modelos (Model-Driven Development – MDD) [16, 19].

Durante o processo de desenvolvimento de software, alguns problemas são encontrados, como: trabalho para escrever os softwares, mudança nas tecnologias exigem mudanças nos softwares, mudanças nos requisitos que geram mudanças nos códigos, e documentos desatualizados de forma que o conhecimento acaba ficando na equipe. Em uma abordagem MDD, todo o desenvolvimento do software é realizado utilizando-se modelos como ponto de partida, de forma que o código dos programas é gerado automaticamente a partir deles [10].

A motivação do desenvolvimento orientado a modelos é mover o foco da programação para a modelagem da solução. Esta abordagem possui o potencial de aumentar a produtividade e qualidade do desenvolvimento. Para atingir os objetivos desejados, MDD permite o desenvolvimento em um nível mais alto de abstração e utilizando conceitos próximos ao domínio do problema, do que aquele oferecido por linguagens de programação. MDD transforma esses modelos de alto nível em modelos específicos de plataforma, e estes em código [21].

A ferramenta Modelio Case Tool Enterprise Edition [14] oferece suporte para transformação dos tipos de mensagens (*«MessageType»*) em XSD. A Figura 16 apresenta os tipos de mensagens do processo de ordem de compra com os estereótipos que definem as regras utilizadas durante o processo de transformação. O estereótipo *«XSDRoot»* indica que o tipo de mensagem *FaturaMensagem* será transformado em documento XSD, enquanto o estereótipo *«XSDElement»* indica que os atributos presentes no tipo de mensagen *FaturaMensagem* serão transformados em elementos (*elements*) no documento XSD correspondente.

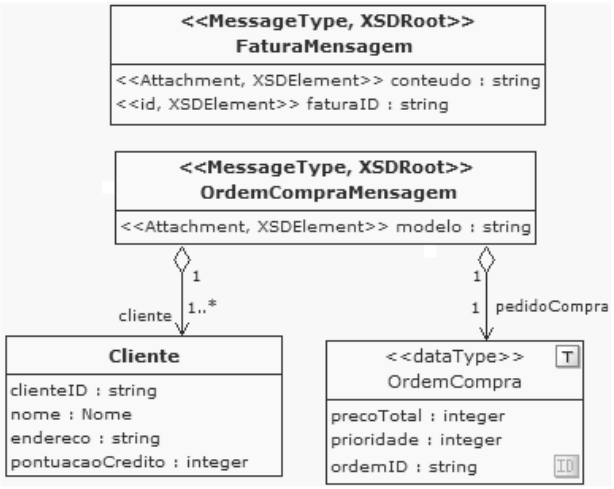


Figura 16. **Tipos de mensagens do processo de ordem de compra com os estereótipos XSD**

A partir do modelo apresentado na Figura 16 é possível gerar os documentos XSD. A Figura 17 ilustra o arquivo *FaturaMensagem.xsd* gerado automaticamente pela ferramenta.

A ferramenta também permite a geração de arquivos WSDL através da interface do serviço (*«ServiceInterfaces»*), que define as operações providas pelo serviço, além das informações necessárias para a

sua invocação. A interface do serviço Fatura (Figura 6) será utilizada para a geração automática do WSDL.

A Figura 18 apresenta as operações (*operation*) suportadas pela interface de serviço *FaturaServico*.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:FaturaMensagem="http://modeliosoft/xsddesigner/FaturaMensagem.xsd"
        targetNamespace="http://modeliosoft/xsddesigner/FaturaMensagem.xsd">
  <element name="conteudo" type="string"/>
  <element name="faturaID" type="string"/>
</schema>
```

Figura 17. **Arquivo *FaturaMensagem.xsd***

```
<ws:portType name="Fatura">
<ws:documentation/>
    <ws:operation name="iniciarcalculopreco">
<ws:documentation/>
 </ws:operation>
    <ws:operation name="finalizarcalculopreco">
<ws:documentation/>
    </ws:operation>
    <ws:operation name="processarfatura">
<ws:documentation/>
    </ws:operation>
</ws:portType>
```

Figura 18. **Operações providas pelo serviço *FaturaServico***

O componente *binding* que descreve o padrão de mensagem e o protocolo utilizado para acessar o serviço é ilustrado na Figura 19.

Por fim, a Figura 19 apresenta o componente *service* que descreve o *endpoint* da implementação do serviço provido.

```
<ws:binding name="IFaturaServiceBinding" type="tns:Fatura">
  <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  <ws:operation name="iniciarcalculopreco">
    <soap:operation soapAction="http://SOATransformer/wsdl/FaturaService.wsdl/iniciarcalculopreco"/>
  </ws:operation>
  <ws:operation name="finalizarcalculopreco">
    <soap:operation soapAction="http://SOATransformer/wsdl/FaturaService.wsdl/finalizarcalculopreco"/>
  </ws:operation>
  <ws:operation name="processarfatura">
    <soap:operation soapAction="http://SOATransformer/wsdl/FaturaService.wsdl/processarfatura"/>
  </ws:operation>
</ws:binding>
```

Figura 19. **Componente *Binding* gerado a partir do serviço *FaturaServico***

```
<ws:service name="FaturaServiceService">
  <ws:documentation/>
  <ws:port name="FaturaServicePort" binding="tns:IFaturaServiceBinding">
    <ws:documentation/>
    <soap:address location="http://localhost/projetopedidocompra"
                  ws:required="false"/>
  </ws:port>
</ws:service>
```

Figura 20. **Componente *service* gerado a partir do serviço *FaturaServico***

Além das transformações apresentadas neste tutorial, a ferramenta Modelio Case Tool Enterprise Edition [14] realiza transformações de SoaML para Java (com anotações *web service*) e de BPMN [17] para BPEL [2], simplificando o desenvolvimento dos serviços modelados durante a fase de projeto do ciclo de vida SOA.

## 5. Conclusão

Em uma abordagem SOA, as decisões tomadas durante a fase de análise de serviços devem ser especificadas utilizando uma linguagem de modelagem de artefatos de software.

A UML não possui todos os artefatos necessários para modelagem de serviços em SOA. Dessa forma, foram propostas extensões da UML (*profiles*) para tratar características específicas para SOA. Dentre os *profiles*, a OMG propõe que seu padrão SoaML seja adotado como padrão de fato para especificação de Arquiteturas Orientadas a Serviço

O *profile* SoaML foi criado para permitir a modelagem de Arquiteturas Orientadas a Serviço, já que a UML padrão não possui suporte nativo à modelagem de serviços. Com esse objetivo, o *profile* introduziu alguns conceitos novos à UML 2.0, como: capacidades que identificam necessidades da organização ou serviços candidatos, tipos de mensagens trocadas entre provedores e consumidores de serviço, interfaces de serviço que especificam capacidades providas e necessidades para realização de um serviço, contratos de serviços que definem papéis envolvidos em um serviço e o fluxo de comunicação permitido entre eles, portas de requisição e provisão de serviços e participantes que consomem e/ou provêem serviços através delas, especificação da arquitetura de serviços de um domínio. Além disso, a introdução destes novos conceitos pelo profile SoaML foi realizada de forma a apoiar a geração automática de artefatos derivados dos modelos, como documentos XSD e WSDL.

Este trabalho apresentou os principais conceitos de SoaML ilustrando em um exemplo que facilita o entendimento do *profile*.

## 6. Referências


[1] Amir, R.; Zeid, A. "A UML profile for Service Oriented Architectures". Conference on Object Oriented Programming Systems Languages and Applications: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, 2004.

[2] OASIS. BPEL 2.0. Disponível em <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>. Acessado em Novembro de 2010.

[3] Dumas, M; Hofstede, A. ter. UML Activity Diagrams as a Workow Specication Language. In M. Gogolla and C. Kobryn, editors, Fourth International Conference on the Unified Modeling Language (UML 2001), pages 76-90, Toronto, Canada, 2001.

[4] Emig, C.; Krutz, K.; Link, S.; Momm, C.; Abeck, S. "Model-Driven Development of SOA Services", 2008.

[5] Ermagan, V.; Krüger, I.H.. "A UML2 Profile for Service Modeling". Model Driven Engineering Languages and Systems - 10th International Conference, 2007.

[6] Gommo, R., Skogan, D., Solheim, I., Oldevik, J. "Model-driven Web Services Development". IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE-04), Taipei, Taiwan, 2004.

[7] Johnston, S. "UML 2.0 Profile for Software Services", 2005. http://www.ibm.com/developerworks/rational/library/05/419_soa.

[8] Josuttis, N. M., SOA in Practice – The Art of Distributed System Design, Beijing; Cambridge; Farnham; Köln; Paris; Sebastopol; Taipei; Tokyo: O'Reilly, 2007, 324 p. Bibliografia: ISBN-10: 0-596-52955-4 / ISBN-13: 978-0-596-52955-0.

[9] Kenzi, A.; El Asri, B.; Nassar M; Kriouile A.. "Engineering adaptable service oriented systems: A model driven approach". IEEE International Conference on Service-Oriented Computing and Applications (SOCA), 2009.

[10] Kleppe, A.; Warmer, J.; Bast, W.. MDA Explained The Model Driven Architecture: Practice and Promise. Addison-Wesley, 2003.

[11] Lopez-Sanza, M.; Acuña, C. J.; Cuesta, C. E.; Marcos, E.. "Modelling of Service-Oriented Architectures with UML". Electronic Notes in Theoretical Computer Science 194 (2008) 23–37 – Elsevier, 2008.

[12] Mayer, P.; Schroeder, A.; Koch, N.. "A Model-Driven Approach to Service Orchestration". IEEE International Conference on Services Computing, 2008a.

[13] Mayer, P.; Schroeder, A.; Koch, N.. "MDD4SOA: Model-Driven Service Orchestration". Enterprise Distributed Object Computing Conference, 2008. 12th International IEEE, 2008b.

[14] Modelio, Modelio Modeling Solutions. Disponível em <http://www.modeliosoft.com/en/products/modelio-enterprise-edition.html>. Acessado em Novembro de 2010.

[15] OMG. Catalog of UML Profile Specifications, 2009. http://www.omg.org/technology/documents/profile_catalog.htm

[16] OMG. "Service-oriented Architecture Modeling Language (SoaML)", 2008. http://www.omg.org/spec/SoaML.

[17] OMG. BPMN 2.0 Beta 2. Disponível em <http://www.omg.org/cgi-bin/doc?dtc/10-06-04>. Acessado em Novembro de 2010.

[18] Papazoglou, Mike P.; Heuvel, Willem-Jan. "Service oriented architectures: approaches, technologies and research issues". VLDB Journal, Springer-Verlag, 2007.

[19] Revoredo, K.; Azevedo, L.; Santoro, F.; Baião, F. "Estudo do Profile SoaML". Relatórios Técnicos do Departamento de Informática Aplicada da UNIRIO (RelaTe-DIA). Disponível em http://www.seer.unirio.br/index.php/monografiasppgi/article/view/966

[20] Scheer. A..-W. ARIS - Business Process Modelling. Springer, Berlin, 2000.

[21] Sendall, S.; Kozaczynski W. "Model Transformation – the Heart and Soul of Model-Driven Software Development". IEEE Software, 2003. ISSN 0740-7459