UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

Relatórios Técnicos
do Departamento de Informática Aplicada
da UNIRIO
n° 0006/2010

Estudos do Profile SoaML

Kate Cerqueira Revoredo
Leonardo Guerreiro Azevedo
Flávia Santoro
Fernanda Baião

Departamento de Informática Aplicada

UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
Av. Pasteur, 458, Urca - CEP 22290-240
RIO DE JANEIRO – BRASIL

Projeto de Pesquisa

Grupo de Pesquisa Participante

Patrocínio

PETROBRAS

Relatórios Técnicos do NP²Tec, No. 0006/2010
Editor: Prof. Sean W. M. Siqueira Setembro, 2010

Estudos do Profile SoaML*

Kate Cerqueira Revoredo[1,2], Leonardo Guerreiro Azevedo[1,2], Flávia Santoro[1,2], Fernanda Baião[1,2]

[1]Núcleo de Pesquisa e Prática em Tecnologia (NP2Tec)
[2]Departamento de Informática Aplicada (DIA) – Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

{katerevoredo, Azevedo, flavia.santoro, fernanda.baiao}@uniriotec.br,

Abstract. Traditional techniques for modeling applications as, for example, UML, can not be directly applied to the modeling of services in a SOA approach. Thus, it is necessary to extend them so that all information about services, their providers and consumers are specified. This can be done using extensions of UML, which are called profiles. These profiles add stereotypes in UML language to represent artifacts with more specific semantics. Some SOA UML profiles have been proposed to represent specific characteristics of SOA, such as: technical specification of the service, the relationship between service providers and its consumers, messages exchanged etc. This report presents some basic concepts about SoaML. This profile has been proposed by the OMG and it is intended to become a standard specification for SOA artifacts.

Keywords: SoaML, UML profile for SOA, service modeling, service modelling.

Resumo. As técnicas tradicionais para modelagem de aplicações como, por exemplo, UML, não podem ser diretamente aplicadas à modelagem de serviços em uma abordagem SOA. Dessa forma, é necessário estendê-las a fim de que todas as informações inerentes a serviços, seus provedores e consumidores sejam especificadas. Isto pode ser feito utilizando extensões da UML. Estas extensões da linguagem são chamadas de *profiles*. Estes *profiles* UML adicionam estereótipos na linguagem para representar artefatos com semântica mais específica. Alguns *profiles* UML para SOA foram propostos para representar características específicas de SOA como: especificação técnica do serviço, relacionamento do provedor do serviço com seus consumidores, mensagens trocadas etc. Neste relatório são apresentados alguns conceitos básicos sobre SoaML. Este *profile* foi proposto pela OMG e pretende-se que este se torne um padrão para especificação de artefatos em SOA.

Palavras-chave: SoaML, profile UML para SOA, modelagem de serviços.

* Trabalho patrocinado pela Petrobras.

Sumário

1 Introdução

As decisões tomadas durante a fase de análise de serviços devem ser especificadas utilizando uma linguagem de modelagem de artefatos de *software*, como a UML em [Azevedo *et al,* 2009c]. Existem diversas ferramentas que apóiam a modelagem utilizando UML. A especificação UML permite criar extensões da linguagem, chamadas de *profiles* [OMG, 2009a]. Estes *profiles* UML permitem adicionar estereótipos na linguagem para representar artefatos com semântica mais específica. Para modelagem de serviços, pode-se utilizar de *profiles* específicos para especificação de serviços web, por exemplo. Numa abordagem SOA, contudo, é importante que a especificação dos serviços se dê através de múltiplas visões de análise, como, por exemplo, a especificação técnica do serviço, a especificação do ponto de vista da arquitetura de *software* da empresa, o relacionamento do provedor do serviço com seus consumidores etc.

Na literatura, alguns *profiles* UML foram propostos tanto no meio acadêmico [Heckel *et al,* 2003; López-Sanz *et al*, 2007], quanto empresarial [Johnston, 2005a; OMG, 2009b]. Dentre estas propostas, podemos destacar os *profiles* que estão atualmente disponíveis para utilização em ferramentas *case*, como a proposta de [Johnston, 2005a], criada pela IBM e disponibilizada na suíte da Rational, e a proposta conjunta da [OMG 2009b], que, embora ainda esteja em fase de refinamento da especificação, pode se tornar um padrão para especificação de artefatos em SOA. Este *profile* está disponível para avaliação na ferramenta ModelPro [ModelDriven, 2009].

Neste relatório são apresentados alguns conceitos básicos sobre SoaML. O estudo realizado seguiu os seguintes passos:

1. Estudo da especificação "Service oriented architecture Modeling Language (SoaML) - specification for the UML Profile and Metamodel for Services (UPMS)"
2. Elaboração deste relatório descrevendo os principais conceitos existentes na especificação do perfil SoaML.
3. Elaboração de exemplo de uso do *profile*.

Este relatório foi produzido pelo Projeto de Pesquisa em SOA como parte das iniciativas dentro do contexto do Projeto de Pesquisa do Termo de Cooperação entre NP2Tec/UNIRIO e a Petrobras/TIC-E&P/GDIEP.

Esse relatório está organizado em 5 capítulos, sendo o capítulo 1 a presente introdução. No capítulo 2, é descrito um domínio que utilizaremos para exemplificar os componentes utilizados pelo profile SoaML. No capítulo 3, o profile SoaML é descrito. Finalmente, o capítulo 4 e 5 apresentam as conclusões e as referências do trabalho, respectivamente.

2 Domínio utilizado nos exemplos

Esta seção descreve o domínio do negócio utilizado como exemplo.

Suponha como cenário, um domínio composto por um conjunto de fabricantes (*manufacturer)*, revendedores (*dealer*), e empresas de transporte (*shipper),* todos independentes uns dos outros, que desejam trabalhar juntos de forma coerente e sem a necessidade de re-projetar os processos de negócio ou sistemas quando estiverem se co-

municando. Por outro lado eles desejam ser capazes de ter seus próprios processos de negócio, regras e informação. É interessante então utilizar uma arquitetura orientada a serviço (SOA) para este domínio de forma a permitir que este ambiente de negócio seja aberto e ágil.

Este tipo de arquitetura poderia ter sido criada por uma organização ou um participante principal. A intenção da arquitetura é permitir que revendedores, fabricantes e empresas de transporte trabalhem em conjunto. Para isso, (i) a arquitetura permite participantes existentes ou novos no comércio através do provimento ou uso de serviços como definido na arquitetura; (ii) os serviços são independentes de qualquer participante, tecnologia ou processos internos do negócio; (iii) Existem contratos bem definidos para determinar como esses participantes trabalham juntos.

A Figura 1**Erro! Fonte de referência não encontrada.** exibe um exemplo de cenário onde a comunidade é composta de um fabricante, a *industria Acme*, um revendedor, *Plubers Are US*, e uma empresa de transporte, *GetItThereFreight*. Neste cenário os membros da comunidade interagem entre si: o revendedor faz um pedido (*Order*) de mercadoria ao fabricante, este confirma este pedido (*Confirmation*); o fabricante solicita então o envio da mercadoria requisitada (*Ship req*), que é enviada pela empresa de transporte (*shipped*). A todo momento é possível verificar o status atual do pedido até que o processo seja finalizado.

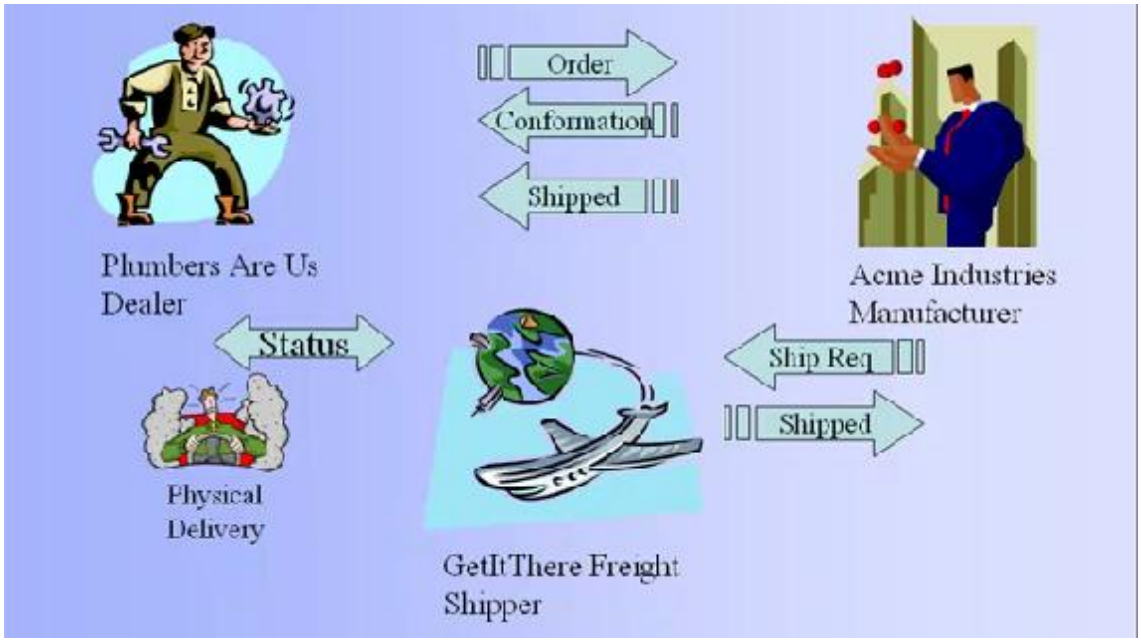

Figura 1 – Cenário da compra de um produto

O que desejamos é especificar essa arquitetura colaborativa, assim como os participantes e os serviços que a compõem. Existem várias abordagens para criar essa arquitetura de serviços. Algumas mais "top down" e outras mais "bottom up". Independente da "direção", o objetivo é gerar essa arquitetura preenchendo com os detalhes quando eles são conhecidos, baseando-se na metodologia utilizada. Neste relatório descrevemos uma abordagem "top down" de como definir uma arquitetura de serviços utilizando a especificação SoaML (Service oriented architecture Modeling Language).

3 SoaML

Esta seção apresenta os principais elementos da especificação SoaML.

A especificação SoaML (Service oriented architecture Modeling Language) foi criada como resposta à requisição por proposta (RFP – Request for Proposal) UPMS (UML Profile and Metamodel for Services) e descreve um *profile* e um metamodelo para projeto de serviços em uma arquitetura orientada a serviços. Os objetivos de SoaML são suportar as atividades de modelagem e projeto e defini-las de forma a serem adequadas para uma abordagem de desenvolvimento orientado ao modelo (*Model-Driven Development – MDD*). Ao invés de pressupor um método específico, o *profile* trata as perspectivas do provedor do serviço, consumidor do serviço e do projeto de sistema que define como consumidores e provedores irão interagir a fim de alcançar os objetivos. SoaML suporta a modelagem de requisitos, incluindo a especificação de sistemas de serviços, especificação de interfaces de serviços individualmente, e especificação de implementação de serviços.

O metamodelo SoaML é baseado em UML 2.0, metamodelo nível 2 da MOF (Meta Object Facility)[1] e provê extensão mínima à UML 2.0. O profile provê uma versão específica do metamodelo que pode ser incorporada em ferramentas de modelagem UML padrão.

SoaML permite modelar as seguintes capacidades:

- Especificar serviços incluindo capacidades funcionais que eles provêem, que capacidades consumidores esperam que sejam providas, os protocolos ou regras para utilizá-las, e a troca de informações entre consumidores e provedores.
- Definir consumidores e provedores de serviços, que requisições e serviços eles consomem e provêem, como eles estão conectados e como as capacidades funcionais dos serviços são usadas pelos consumidores e implementadas pelos provedores de uma forma consistente, tanto com os protocolos de especificação dos serviços quanto com as restrições especificadas.
- Fornecer as regras para usar e prover serviços;
- Definir serviços e restrições de uso de serviços, além de links entre eles com outros metamodelos OMG, tais como: o metamodelo course_of_action do BMM[2] (Business Motivation Modeling) o metamodelo Process do BPDM[3] (Business Process Definition Meta-model), o metamodelo OperationalCapability do UPDM[4] (Unified Profile for DoDAF and MODAF) e/ou elementos UseCase da UML que eles realizam, suportam ou preenchem.
- Consultar serviços, as necessidades que eles pretendem satisfazer e possíveis dependências entre eles.

[1] http://www.omg.org/mof/

[2] http://www.businessrulesgroup.org/second_paper/BRG-BMM.pdf

[3] http://www.omg.org/technology/documents/br_pm_spec_catalog.htm, http://www.omg.org/spec/BPDM/1.0/

[4] http://www.architectureframework.com/updm/

SoaML foca nos conceitos de modelagem de serviços, onde a intenção é usar isto como fundamentos para outras extensões relacionadas com integração com outros metamodelos OMG como BPDM e BPMN2.0, assim como SBVR, OSM, ODM e outros.

3.1 Conceitos Básicos

O primeiro conceito básico é o de serviço. Serviço é definido como um oferecedor de valor para consumidores através de uma ou mais capacidades. O acesso para o serviço é provido usando um contrato e esse serviço é utilizado de forma consistente com as restrições e políticas especificadas pelo contrato de serviço. Um serviço é provido por um participante que age como provedor de serviço. Os eventuais consumidores de serviço podem não ser conhecidos pelo provedor e devem utilizar os serviços de acordo com o que foi originalmente determinado pelo provedor.

Provedores e consumidores podem ser pessoas, organizações, componentes tecnológicos ou sistemas, os quais são denominados participantes. Participantes oferecem serviços através de portas via o estereótipo *ServicePoint* e requisitam serviços em portas com o *ResquestPoint*. Essas portas representam pontos onde o serviço é oferecido ou consumidor.

O *ServicePoint* tem um tipo que descreve como usar o serviço. O tipo pode ser uma interface UML (para serviços muito simples) ou uma *ServiceInterface*. Em ambos os casos o tipo do *ServicePoint* especifica diretamente ou indiretamente, tudo que é necessário para interagir com o serviço – é o contrato entre os provedores e consumidores do serviço.

A Figura 2 descreve um participante, denominado *Productions,* o qual prove um serviço *scheduling*. O tipo da porta do serviço é a interface UML Scheduling que tem duas operações, *requestProductionScheduling* e *sendShippingSchedule*. A interface define como um consumidor do serviço *Scheduling* deve interagir. Note que um participante pode também oferecer outros serviços em outros *ServicePoints*. O participante *Production* tem dois *behaviors* que são os métodos providos através do serviço *scheduling*.

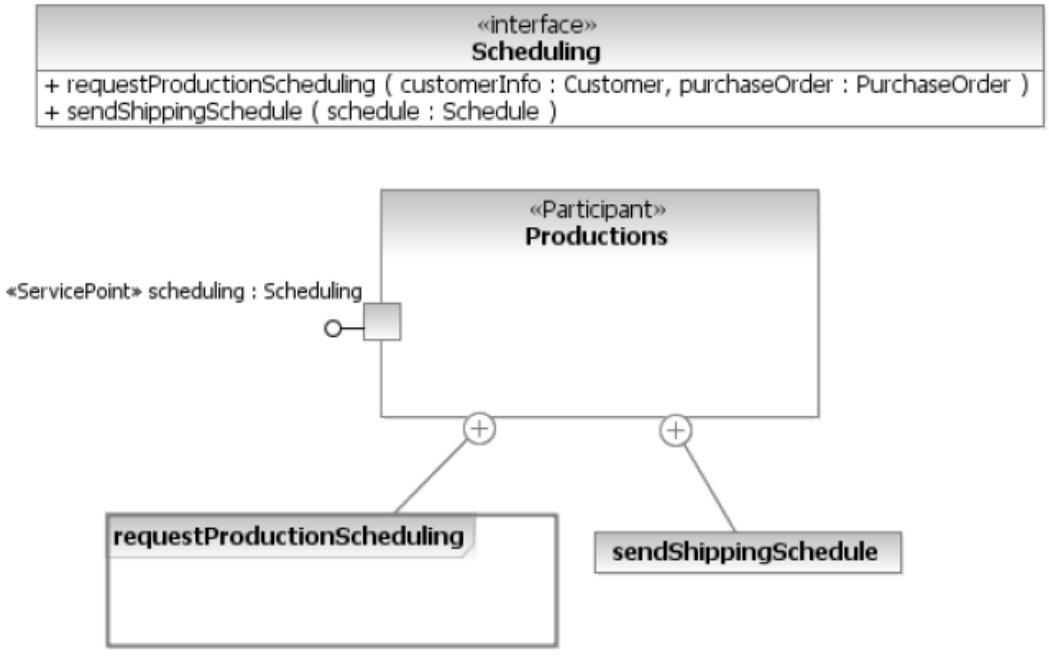

Figura 2 – Participante Production provedor do serviço Scheduling

Nas seções seguintes serão apresentados os principais elementos utilizados na especificação de um serviço utilizando SoaML.

3.2 ServiceInterface

Uma *ServiceInterface* define a interface e as responsabilidades para um participante para que ele assuma um determinado papel em um *ServiceContract*. São as formas para especificar como um participante interage como provedor ou consumidor de acordo com o especificado no *ServiceContract*. Uma *ServiceInterface* é modelada como uma classe UML.

Assim como uma interface UML, uma *ServiceInterface* pode ser do tipo de uma porta do serviço. A *ServiceInterface* tem como característica adicional permitir especificar um serviço bi-direcional – onde tanto o provedor quanto o consumidor têm responsabilidades no envio e recebimento de mensagens e eventos. A *ServiceInterface* é definida da perspectiva do provedor de serviço usando três elementos primários:

- Interfaces necessárias: as interfaces necessárias especificam as mensagens enviadas/recebidas por consumidor/provedor.
- Classe *ServiceInterface*: As partes incluídas na *ServiceInterface* representam os papéis que serão assumidos pelos participantes envolvidos com o serviço. O papel que é realizado pela interface será assumido pelo provedor do serviço; o papel que é feito pelo usuário da interface será assumido pelo consumidor.
- Protocolo *Behavior*: O *Behavior* especifica as interações válidas entre o provedor e o consumidor – o protocolo de comunicação da interação, sem especificar como cada uma das partes implementa o seu papel. Qualquer especificação UML *behavior* pode ser utilizada, mas diagramas de interação e atividade são os mais comuns.

As capacidades e necessidades de um serviço são definidas pelo seu tipo, o qual é uma *ServiceInterface*, ou em alguns casos simples, uma *Interface* UML2. Uma *ServiceInterface* especifica as seguintes informações:

- O nome do serviço indicando o que ele faz ou sobre o que é;
- As capacidades do serviço através das interfaces;
- As necessidades ou capacidades exigidas pelo consumidor para utilizar o serviço através das interfaces;
- Uma especificação detalhada de cada capacidade usando uma operação; incluindo seu nome, pré-condições, pós-condições, entradas e saídas e qualquer exceção que possa surgir;
- Qualquer protocolo ou regras para uso de capacidade ou como se espera que consumidores respondam e quando através de um comportamento próprio;
- Regras para como os serviços devem ser implementados pelos provedores;
- Restrições que podem determinar se o serviço atingiu com sucesso o seu propósito.

Estas são as informações que consumidores potenciais precisarão para determinar se um serviço responde suas necessidades e como usá-lo em caso positivo. Também

fornece a informação que os provedores precisam para implementar o serviço. A Figura 3 apresenta um exemplo genérico de especificação de interface de serviço utilizando o elemento *ServiceInterface*. Na Figura podemos ver que a *ServiceInterface* implementa o que é definido na interface *Capacity* usando o definido pela Interface *Necessity*. Faz parte da especificação da ServiceInterface a definição da ordem com que as operações são executadas. Na Figura 3, no diagrama de atividade (*Protocol*) é apresentado que ao implementar a *Capacity* será necessário usar *Necessity*.

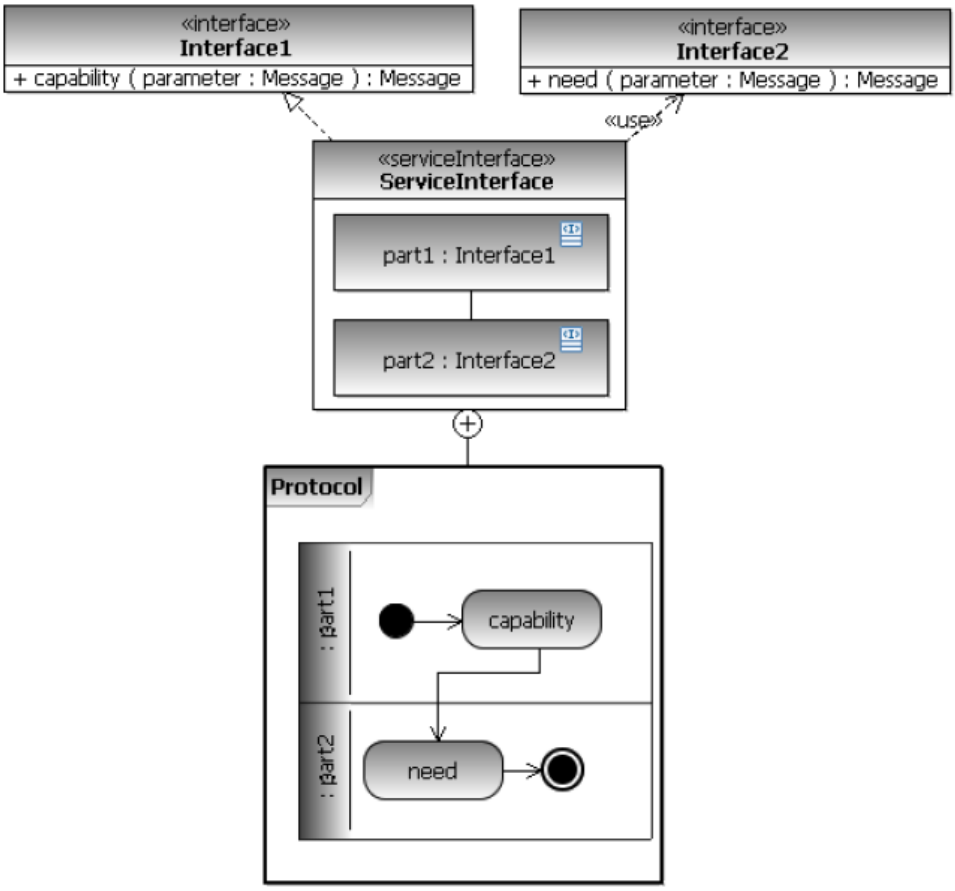

Figura 3 – Exemplo de especificação de interface de serviço

3.3 Participantes

Participantes representam componentes de software, organizações, sistemas ou indivíduos que provêem e usam serviços. Participantes provêem capacidades e consomem serviços através de portas de serviços pelos estereótipos *ServicePoints* e *RequestPoints*, respectivamente, que tem os seus tipos definidos pelas *ServiceInterfaces* ou, em casos mais simples, interfaces UML.

Um serviço usa o conceito UML de porta e indica o ponto de interação onde um participante interage com outros participantes. Na Figura 4, o Participante *Shipper* tem a porta <<service>> *shippingService*, onde ele recebe a solicitação de envio (*ScheduleProcessing*) e provê o envio (*Shipping*).

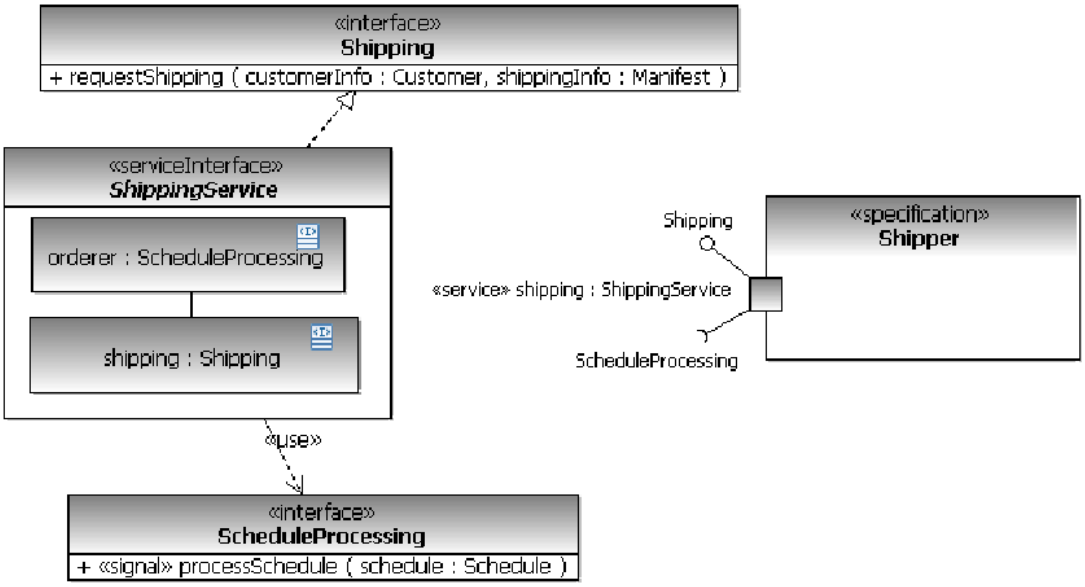

Figura 4 – Exemplo de participante com uma porta de serviço

Assim como definimos os serviços providos por um participante usando uma porta de serviço através do estereótipo *ServicePoint*, podemos definir os serviços que um participante precisa consumir. Um participante expressa sua necessidade através de uma solicitação para serviços de outros Participantes. Uma requisição é definida usando uma porta correspondendo ao estereótipo *RequestPoint*.

Figura 5 – Participante com serviços e requisições

O tipo de *RequestPoint* também é uma *ServiceInterface*, ou uma *interface* UML, assim como para o *ServicePoint*. A Figura 5 mostra um participante com um *ServicePoint* (estereótipo <<*service*>>) e três *RequestPoints* (estereótipo <<*request*>>).

Enquanto é importante não detalhar muito os participantes com relação ao domínio sendo especificado, de forma que a modelagem fique suficientemente genérica, é igualmente importante para um fabricante (*Manufacturer*) em particular ser capaz de criar ou adaptar sua arquitetura interna de modo a preencher suas responsabilidades dentro do domínio. Então queremos ser capazes de especificar as características de um fabricante em particular e ver como eles estão adaptados para esses serviços.

Como exemplo de especificação detalhada para um participante em particular temos a especificação apresentada na Figura 6. Nesta temos a arquitetura de serviços para um participante fabricante em particular, ou seja, temos detalhada a arquitetura de colaboração para este participante. Note que a arquitetura de um participante é definida para o fabricante como uma estrutura composta. Neste caso o fabricante delegou o serviço *invoicing* e *ordering* para os participantes *accounting* e *order processing*

respectivamente. *Accounting* e *Order Processing* são participantes, mas participantes agindo dentro do contexto do fornecedor.

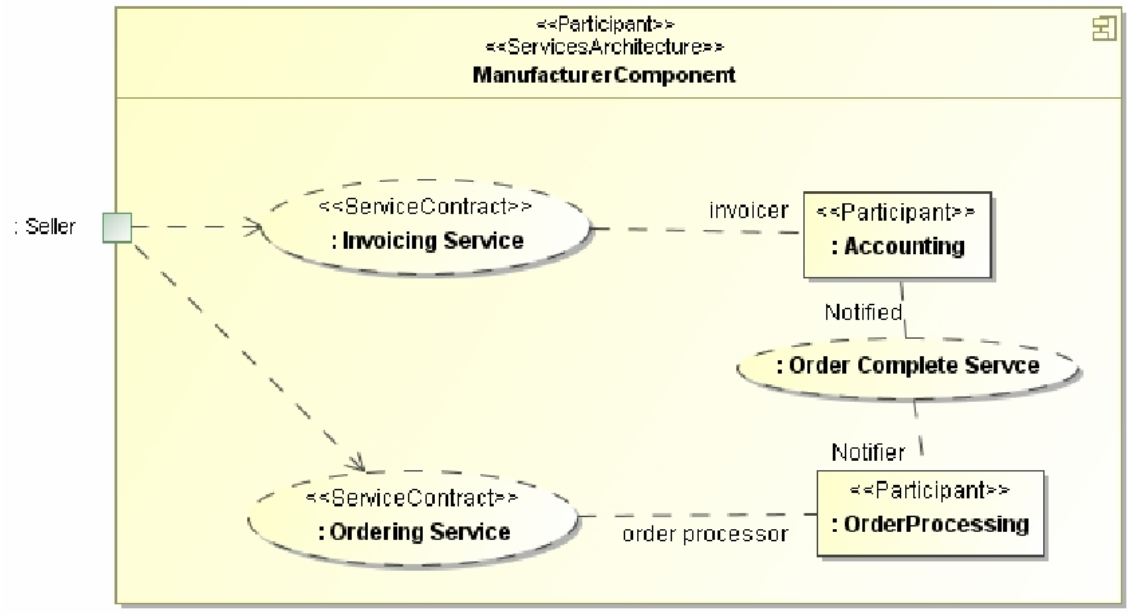

Figura 6 – Especificação do participante fabricante (*Manufacturer*)

3.4 ServiceContract

A parte chave de um serviço é o seu contrato (*ServiceContract*). Um *ServiceContract* é a especificação do acordo entre os provedores e os consumidores de um serviço. Os contratos apresentam especificações dos serviços sem considerar a sua realização, capacidades ou implementação. Um *ServiceContract* define os termos, condições, interfaces e coreografia que os participantes precisam estar de acordo para que (direta ou indiretamente) o serviço possa ser usado. Um contrato de serviço é ligado tanto nos provedores quanto nos consumidores daquele serviço. A base para descrição de um contrato de serviço também é um diagrama de colaboração da UML que foca nas interações envolvidas em prover um serviço.

Os *ServiceContracts* definem os papéis que serão executados por consumidores e provedores do serviço. Muitos *ServiceContracts* têm apenas dois papéis, um para o consumidor e outro para o provedor, como mostrado na Figura 7. Neste exemplo, o serviço *Ordering* tem dois papéis: o do consumidor *Orderer* e o do provedor *OrderProcessor*. O primeiro solicita um pedido e o segundo processa essa solicitação provendo o que foi pedido ou indicando que não será possível prover.

Figura 7 – Exemplo de um *ServiceContract* com dois papéis.

Entretanto, qualquer quantidade de papéis é permitida, como mostra a Figura 8. Nela é possível ver que o serviço *Purchasing* é composto por outros dois serviços: *Ordering* e *Invoicing*. O primeiro especificado em mais detalhes na **Figura 7**. Neste cenário, o comprador (*buyer*) é o solicitante de pedido (*Orderer*) do serviço *Ordering* e o pagador (*payer*) do serviço *Invoicing*. O vendedor (*seller*) é o processador do pedido (*OrderProcessor*) do serviço *Ordering* e o gerador de fatura (*invoicer*) do serviço *Invoicing*.

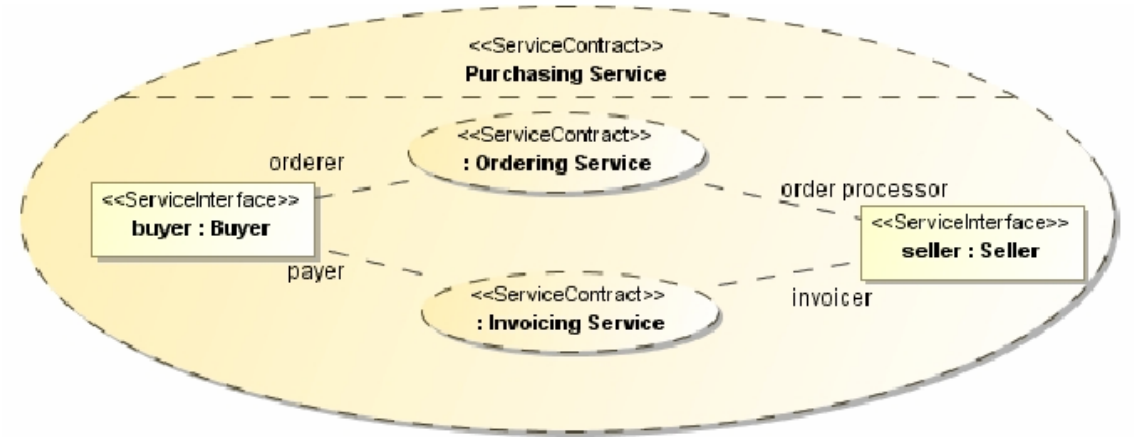

Figura 8 – Exemplo de um *ServiceContract* com quatro papéis.

Uma parte importante do *ServiceContract* é a coreografia, a qual especifica o que é transmitido entre os papeis e quando isto ocorre. A coreografia especifica trocas entre os papeis – os dados, artefatos e obrigações entre as partes. Ela especifica o que acontece entre os papeis provedores e consumidores sem definir seus processos internos, os quais devem estar compatíveis com seu contrato de serviço (*ServiceContract*).

Uma coreografia de um *ServiceContract* é um comportamento UML que pode ser modelado como um diagrama de interação ou um diagrama de atividade, como mostra a Figura 9.

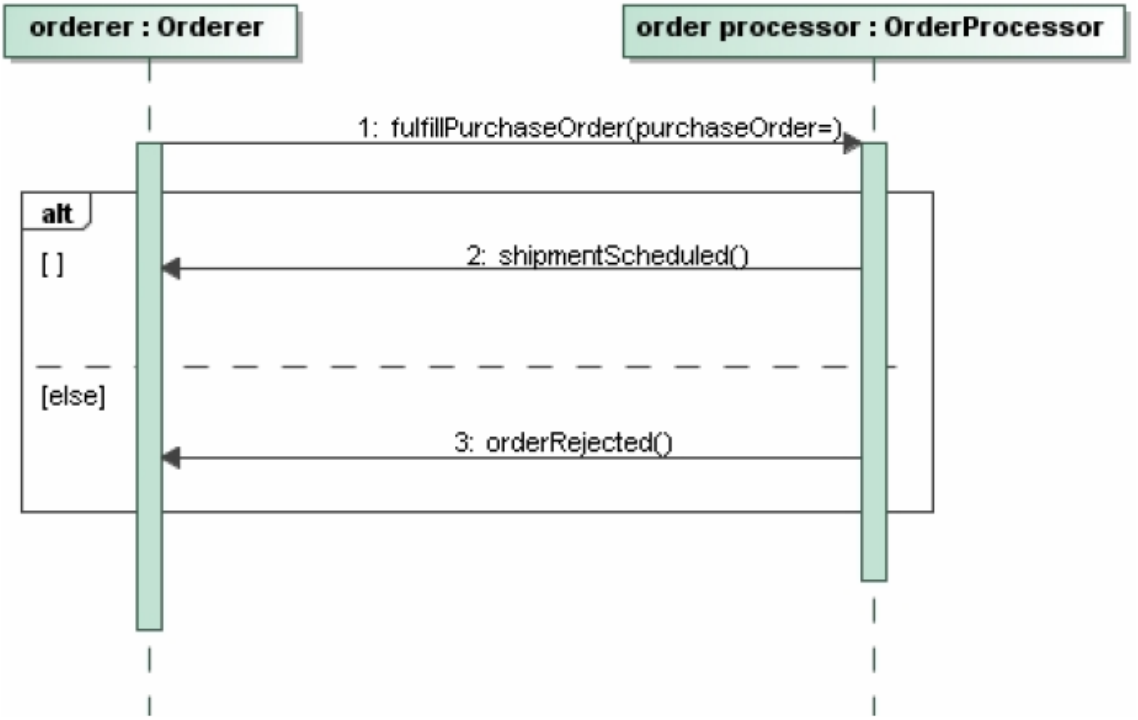

Figura 9 – Coreografia de serviços

Cada papel no contrato é representado por uma linha a qual age como a fonte e o objetivo das mensagens enviadas. No nosso exemplo, representado na Figura 9, temos dois papéis - *orderer:Orderer* e *order processor: OrderProcessor*. Logo, temos duas linhas de vidas. Mensagens são modeladas como sendo passadas via chamadas para operações na interface dos papeis. No nosso exemplo, temos três mensagens sendo enviadas. São permitidos fluxos condicionais usando fragmentos de interação construídos dentro do diagrama de sequência. No exemplo, temos um fluxo condicionado, que indica que ou o *shipmentScheduled* ou *orderRejected* é retornado pelo *order processor*.

Os pontos de serviço dos participantes têm um tipo que define as responsabilidades dos participantes em relação a um serviço, isto é a *ServiceInterface*. A *ServiceInterface* é o tipo de um papel em um e somente um contrato de serviço. O contrato de serviço detalha as responsabilidades de todos os participantes, suas responsabilidades com respeito a um serviço. Cada interface de serviço tem um tipo que realiza a interface exigida daquele tipo assim como usando o serviço que deve ser suprido por um consumidor daquele serviço.

A Figura 10 mostra como as *ServiceInterfaces* são relacionadas à interface UML que define os sinais e operações implementadas, assim como aquelas usadas. Este padrão comum define um serviço bi-direcional assíncrono. Note que uma interface de serviço são os tipos dos papeis em um contrato de serviço. As mesmas interfaces de serviço serão os tipos dos *servicePoints* nos participantes, dessa forma definindo os contratos que cada participante é exigido em conformar.

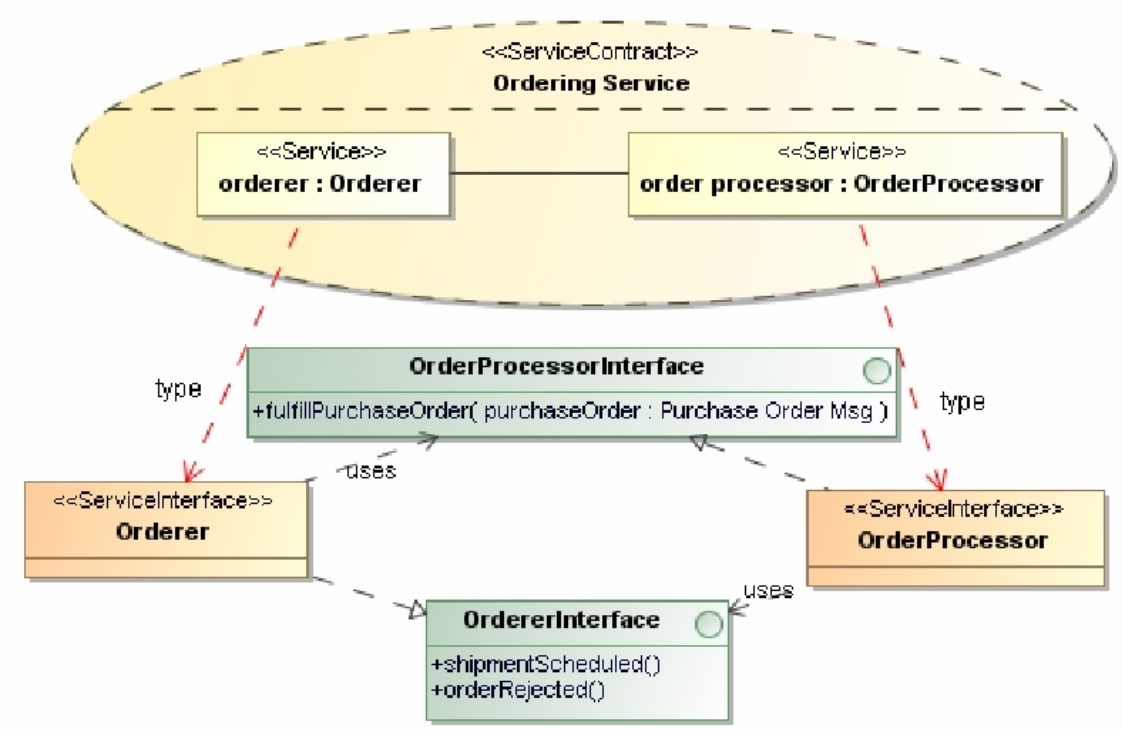

Figura 10 – Relação entre a *ServiceInterface* e o *ServiceContract*

Participantes são ligados a papéis específicos que eles executam de um contrato. Na Figura 11, por exemplo, o participante *handler: Claims Handler* assume o papel de provedor. Associado ao elemento *Participant* temos o seu tipo. O participante ligado a um *ServiceContract* exige que o tipo do participante tenha uma porta com a *ServiceInterface* correspondente. Neste caso, o participante *handler: Claims Handler* tem uma *ServicePoint*. A porta prove a interface provedora e exige a interface consumidora. Note que o significado de interface provida e consumida é revertido em um *RequestPoint*: a porta consome a interface do provedor e provê a interface do consumidor. As dependências relativas aos *RequestPoint* e *ServicePoint* ficam juntas para permitir uma conexão entre os serviços consumidores e provedores.

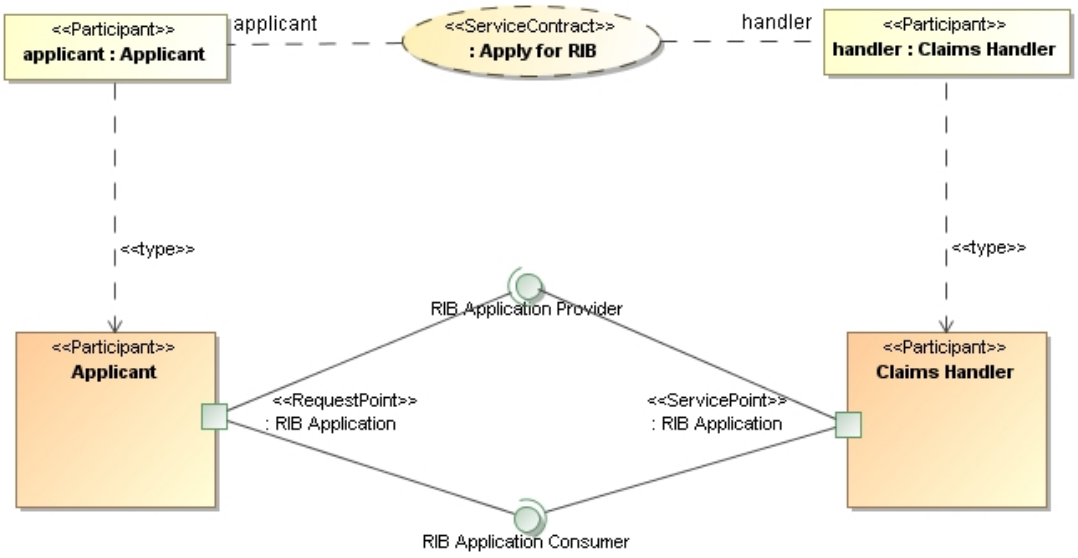

Figura 11 – Cada participante tem um papel especificado no *ServiceContract*

3.5 ServiceArchitecture

Uma das principais características da SoaML é a capacidade de representar um contexto de interação de serviços e participantes e representar o relacionamento entre esses contextos de interação. Dessa forma, é possível visualizar, em alto nível, a arquitetura de serviços da organização. Para tal, o estereótipo *ServicesArchitecture* é disponibilizado. Uma *ServicesArchitecture* é uma rede de papéis de participantes provendo e consumindo serviços para realizar um propósito. A arquitetura de serviços define os requisitos dos tipos de participantes e realizações dos serviços que preenchem esses papéis. Com a definição de papéis, a SoaML permite modelar pessoas, organizações e sistemas que colaboram, sem se preocupar, em um primeiro momento, em quais são esses sistemas, pessoas ou organizações.

A arquitetura de serviços é 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 do participante e especifica a arquitetura para um participante em particular.

A arquitetura de serviços do domínio do exemplo é definida usando um diagrama UML de colaboração, como apresentado na Figura 12.

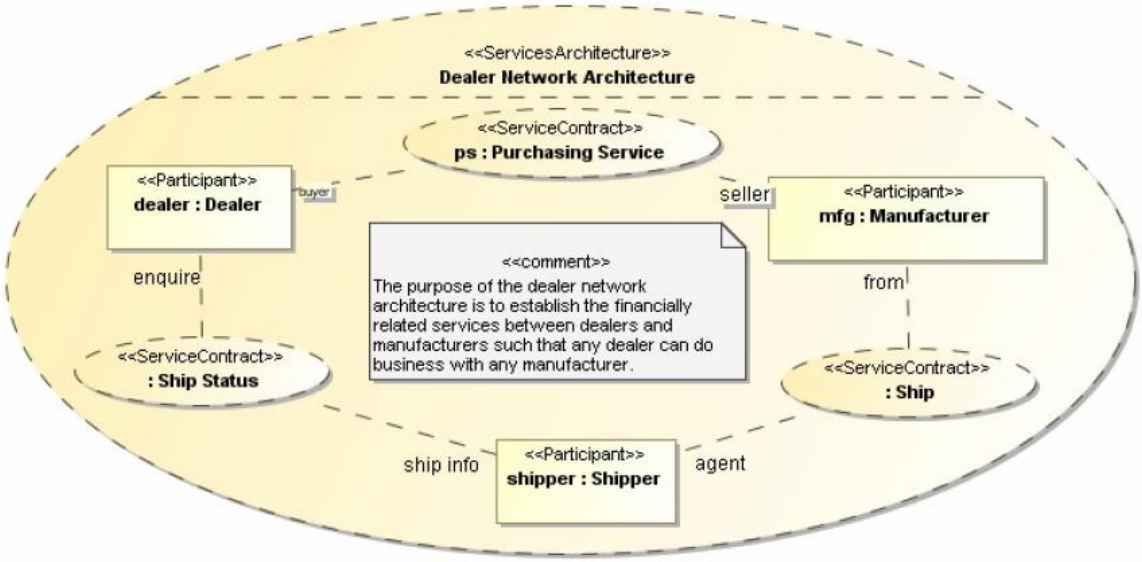

Figura 12 – Arquitetura de serviços da comunidade

O propósito da colaboração, apresentado na Figura 12, é ilustrar como tipos de entidades trabalham em conjunto para algum propósito. Colaborações são baseadas no conceito de papéis para definir como entidades estão envolvidas na colaboração (como e porquê elas colaboram) sem depender de que tipo de entidade está envolvida (por exemplo, pessoa, organização ou sistema). No exemplo, a arquitetura *Dealer Network Archicture* define um ambiente onde três participantes (um fabricante – *manufacturer*, um revendedor – *dealer*, e uma empresa de envio – *shipper*) se relacionam através dos serviços de compra (*Purchasing Service*), envio (*Ship*) e estado da remessa (*Ship Status*). Assim, um revendedor compra um item de um fabricante através do serviço de compra. O fabricante repassa o item para uma empresa de envio através do serviço de envio e o revendedor, por sua vez, pode verificar a condição do envio do item que foi enviado pela empresa de envio através do serviço de verificação do estado da remessa (*Ship Status*).

A arquitetura de serviços do participante, por sua vez, ilustra como sub-participantes e colaboradores externos trabalham em conjunto e quase sempre representa um processo de negócio. Um *ServicesArchitecture* (tanto da colaboração quanto do participante) pode ser composto de outras arquiteturas de serviços e contratos de serviços. Como mostra a Figura 13, participantes (*Participant*) são classificadores definidos tanto pelos papéis que executam na arquitetura de serviços (o papel do participante) quanto nos requisitos do "contrato" das entidades que executam esses papéis.

Cada participante pode "executar um papel" diferente em arquiteturas diferentes.

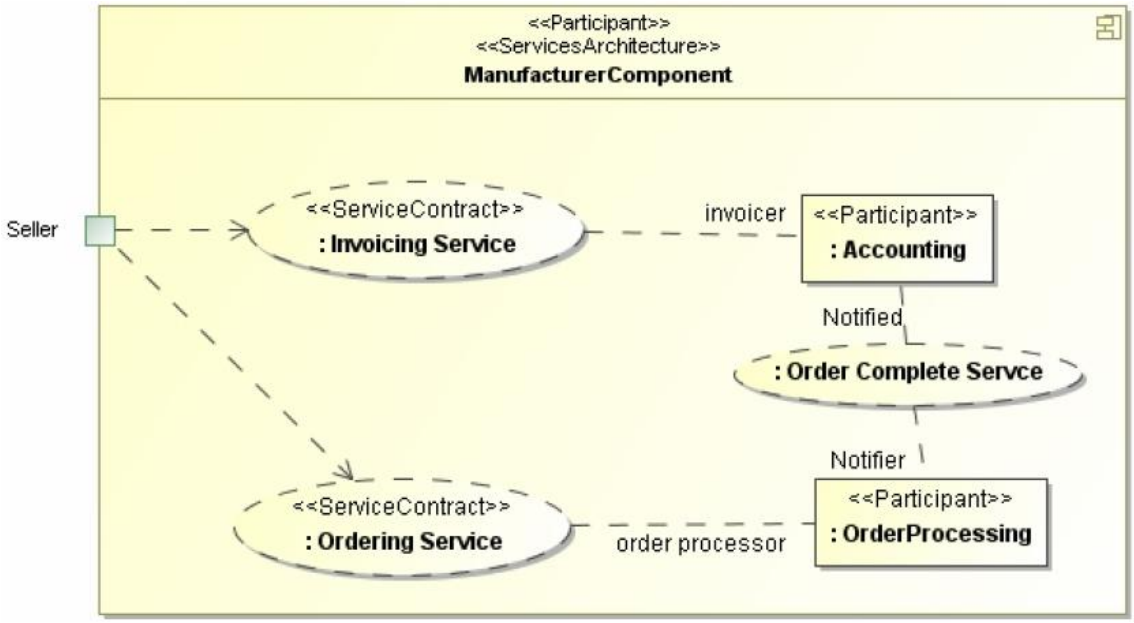

Figura 13 – Arquitetura de serviços do participante

A especificação completa de um participante inclui as portas para todos os *ServicesContracts* para os quais o participante participa dentro da arquitetura de serviços. A Figura 14 mostra um diagrama com a especificação de todos os participantes do domínio.

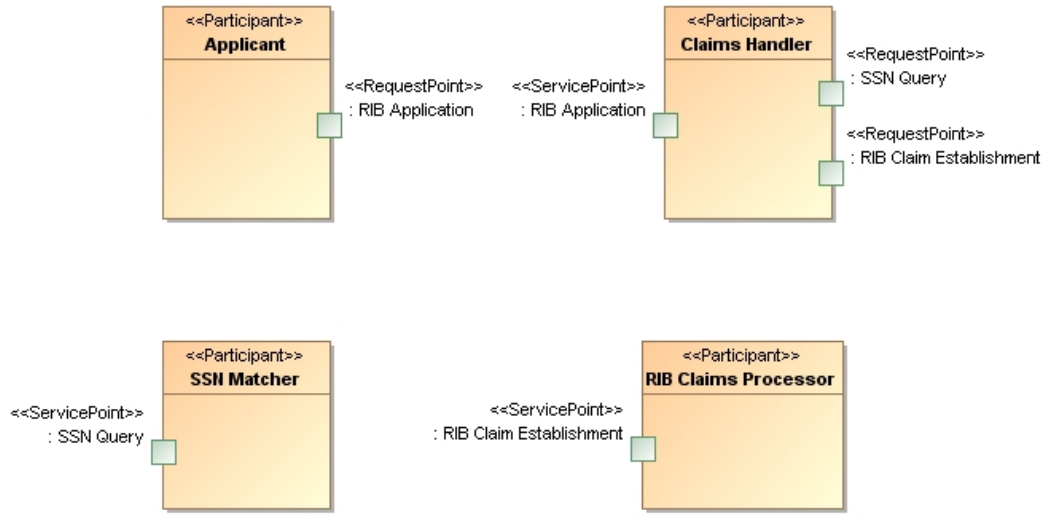

Figura 14 – Diagrama exibindo todos os participantes do domínio

A Figura 15 exibe o relacionamento de todos os componentes que abordamos neste relatório. É possível ver o caminho entre uma arquitetura de serviço passando pelos participantes e seus *ServicePoints* para os *ServicesContracts* que definem as interfaces de sérviço para cada um dos pontos de serviço.

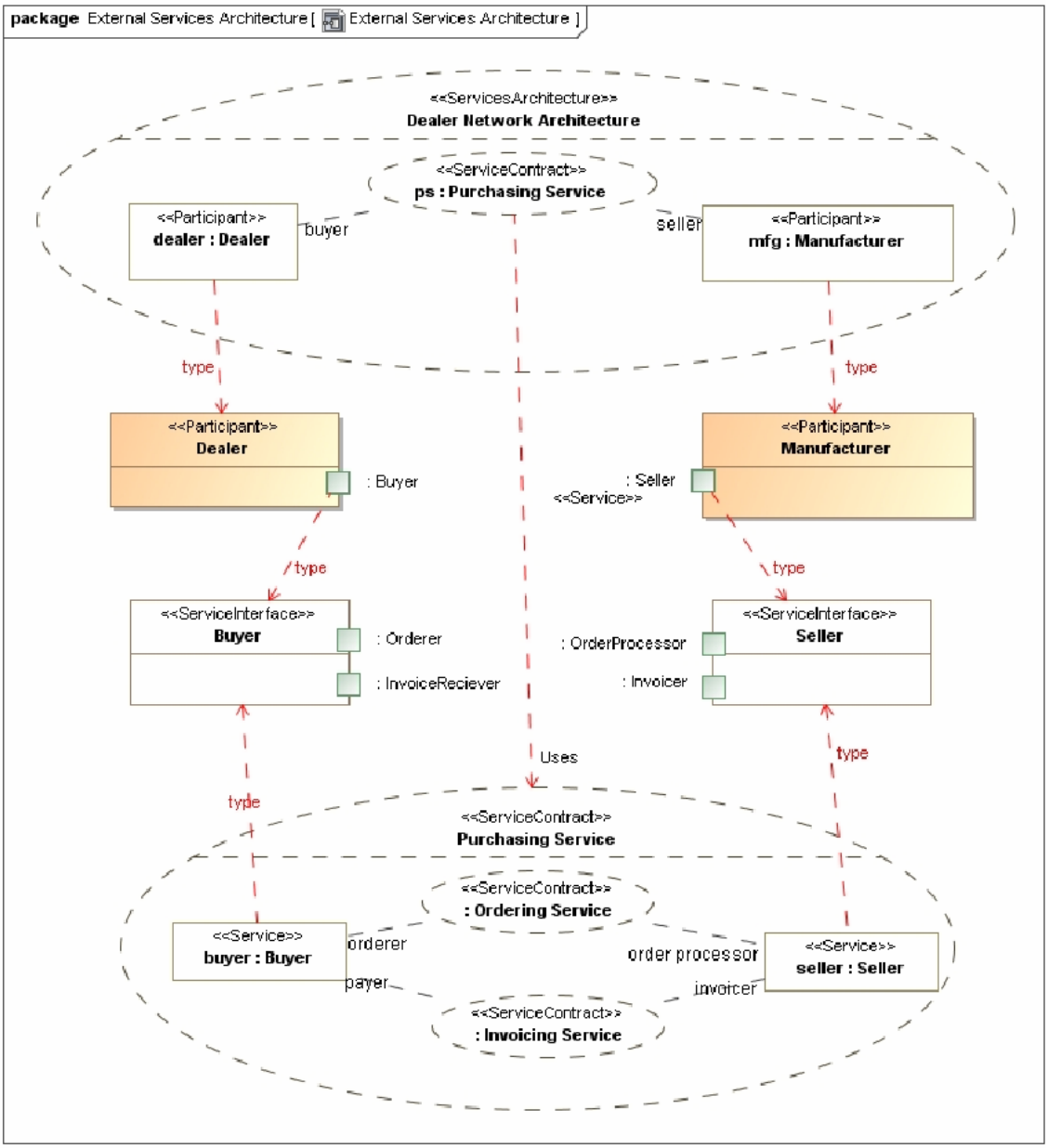

Figura 15 – Relacionamento dos componentes abordados no relatório

O que a Figura 15 mostra é que o revendedor e o Fornecedor são participantes na *Dealer Networ Architecture*, na qual o negociante executa o papel de comprador no *PurchasingSerice* e o fornecedor executa o papel de vendedor no mesmo serviço. Participar nestes serviços exige que eles tenham portas de serviço definidas no tipo do participante, estas são as portas nos tipos *Dealer* e *Manufacturer*. Estas portas têm um tipo *ServiceInterface* definido no *ServiceContract Purchasing*. Estas *ServiceInterfaces*, cada uma, tem duas portas porque o serviço *Purchasing* é um componente *ServiceContract*. A *ServiceInterface* tem uma orta para cada serviço composto: *OrderingService* e *InvoicingService*, respectivamente.

3.5.1 Exemplo de uma Arquitetura de serviços

Nesta seção vamos criar uma arquitetura de serviços, usando um projeto *top-down*, para o domínio apresentado no Capítulo 2.

O objetivo é modelar esse cenário como uma arquitetura de serviços, modelando as interações entre os participantes como serviços e definindo os contratos de seviços . Os passos para realizar esta modelagem são:

1. Usar uma ferramenta de UML para criar o modelo;
2. Criar um pacote de estruturas típicas para SoaML;
3. Definir as arquiteturas de serviços e os participantes;
4. Definir os contratos de serviços;
5. Usar os contratos de serviços completando a arquitetura de serviços.

A ferramenta UML utilizada foi o *MagicDraw*, por ser a ferramenta recomendada para trabalhar com SoaML, junto com o perfil *SoaML Cameo SOA+* [OMG,2009b]. A Figura 16 inlustra o ambiente do MagicDraw UML.

Figura 16 – Tela inicial da ferramenta MagicDraw

Começamos a modelar a arquitetura de serviços criando um novo projeto, que denominaremos *DealerNetwork*. A Figura 17 mostra a tela onde o projeto é criado no MagicDraw.

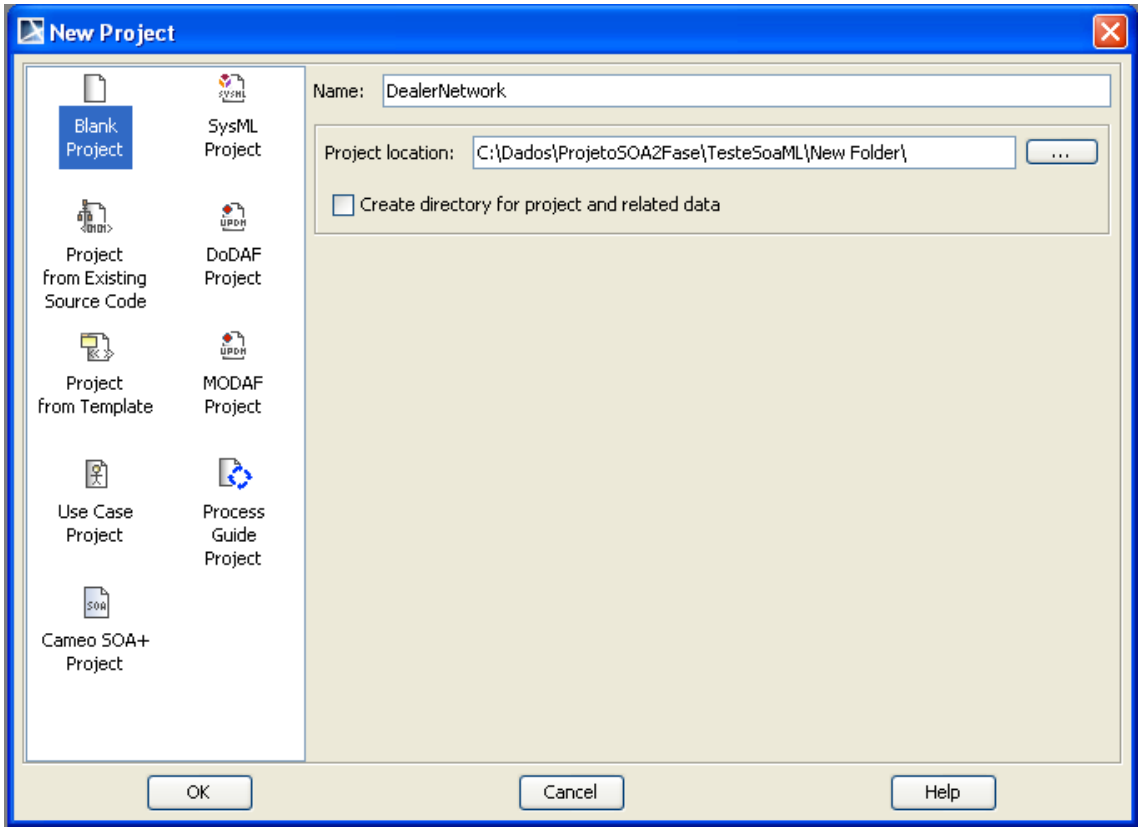

Figura 17 – Tela para criação de um novo projeto

Criamos um pacote para armazenar todos os elementos da arquitetura, que denominamos também *DealerNetwork*. Pode-se criar subpacotes para organizar os modelos. Criamos então um pacote para a arquitetura de serviços, um para os serviços, onde nestes criaremos posteriormente outros pacotes para colocar os elementos de cada um dos serviços, e um para as mensagens. A Figura 18 mostra o pacote *Dealer Network* já com os subpacotes definidos, onde serão armazenados os serviços, a arquitetura de serviços e as mensagens.

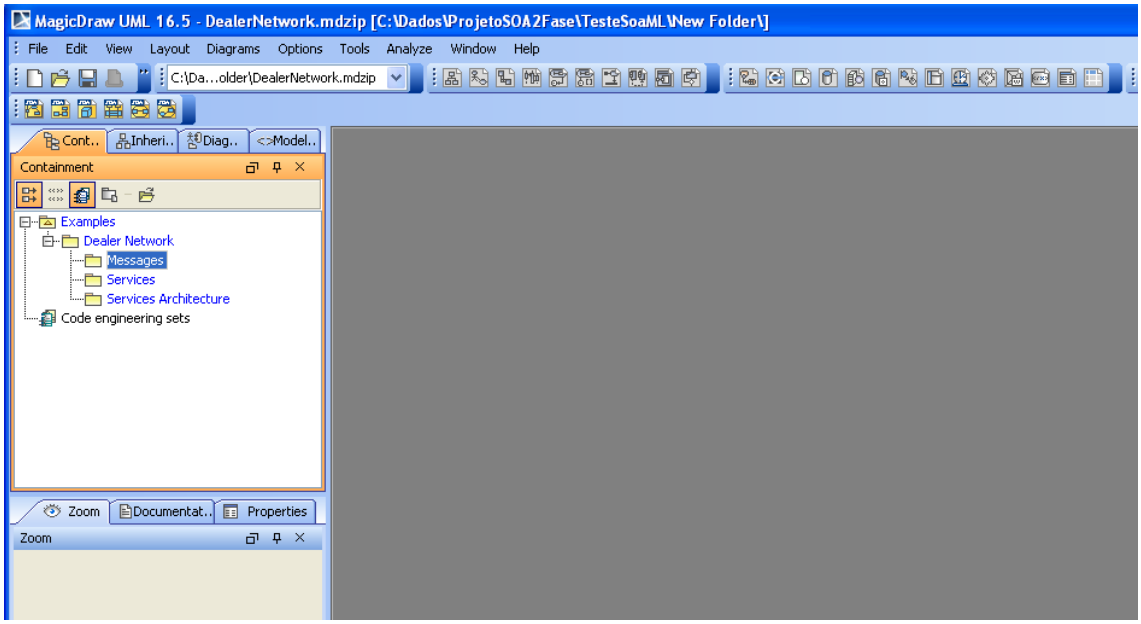

Figura 18 – Tela mostrando os pacotes criados

Em seguida, criamos um diagrama para a arquitetura de serviços (*Services Architecture Diagram*), como exibido na Figura 19.

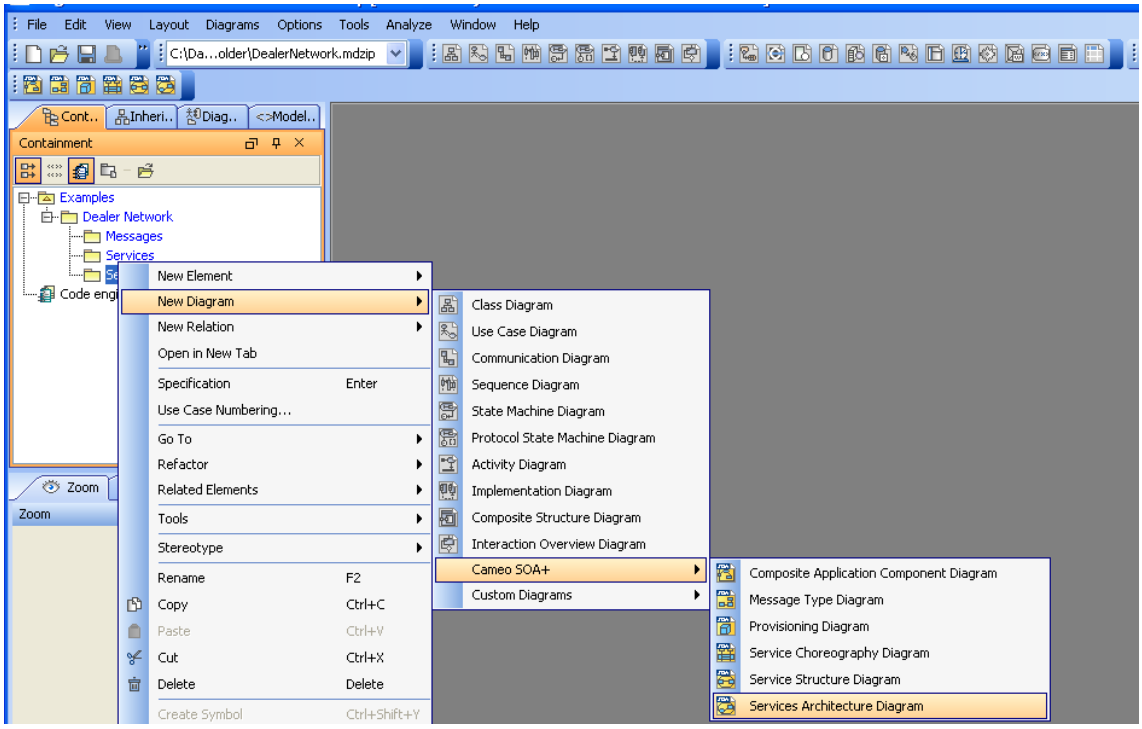

Figura 19 – Criação de um diagrama para especificar a arquitetura de serviços

O primeiro elemento inserido no diagrama é o elemento *ServiceArchitecture*, como exibido na Figura 20. O elemento foi denominado *Dealer Network Architecture*.

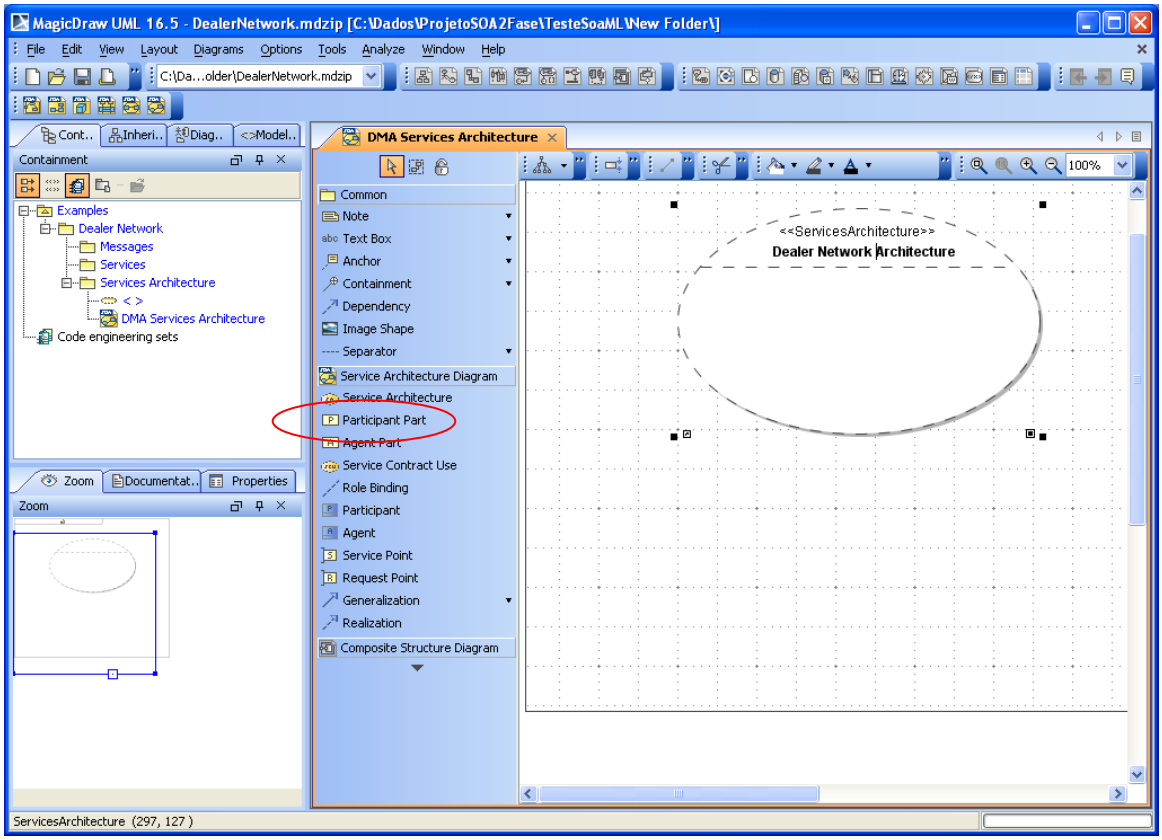

Figura 20 – Elemento *ServiceArchitecture*

Em seguida, inserimos os participantes utilizando o elemento *ParticipantPart* . A Figura 21 mostra o elemento *ParticipantPart* que representa o fornecedor no nosso exemplo.

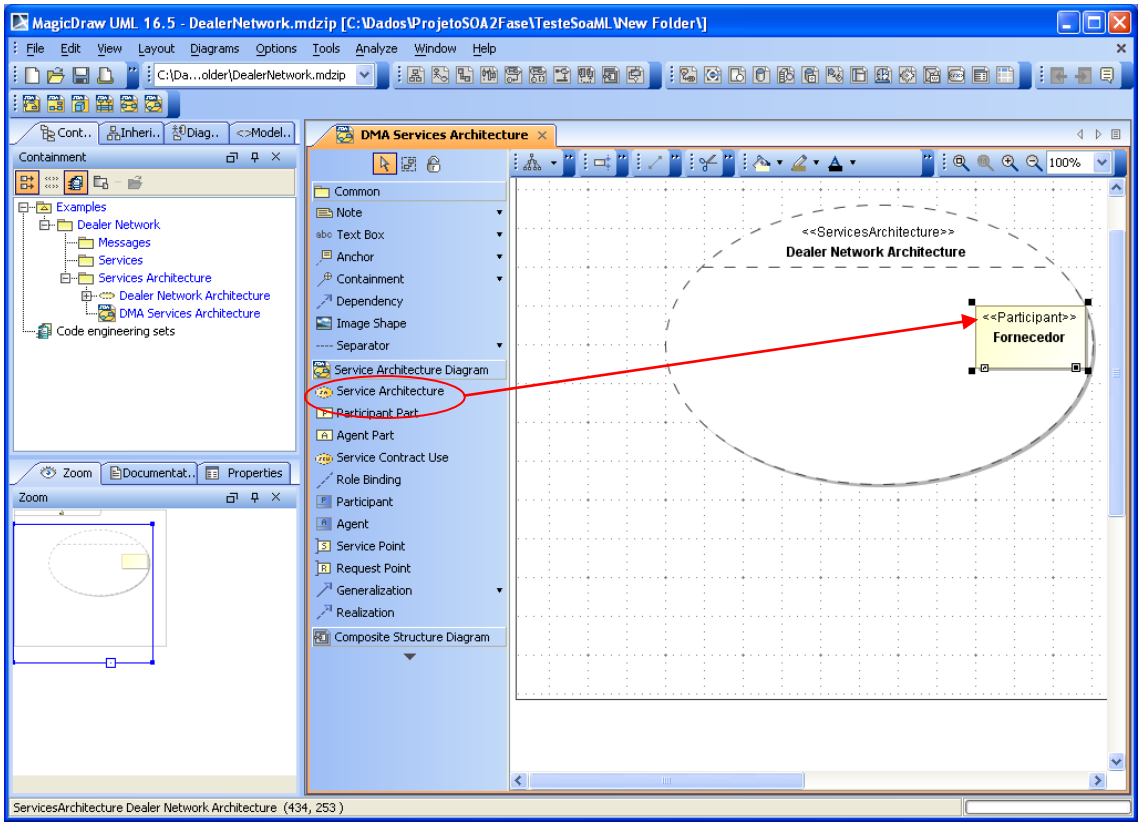

Figura 21 – Incluindo o elemento *ParticipantPart* na *ServiceArchitecture*

Um elemento *ParticipantPart* é inserido para representar os outros participantes do domíno: revendedor e empresa de transporte, como exibido na Figura 22.

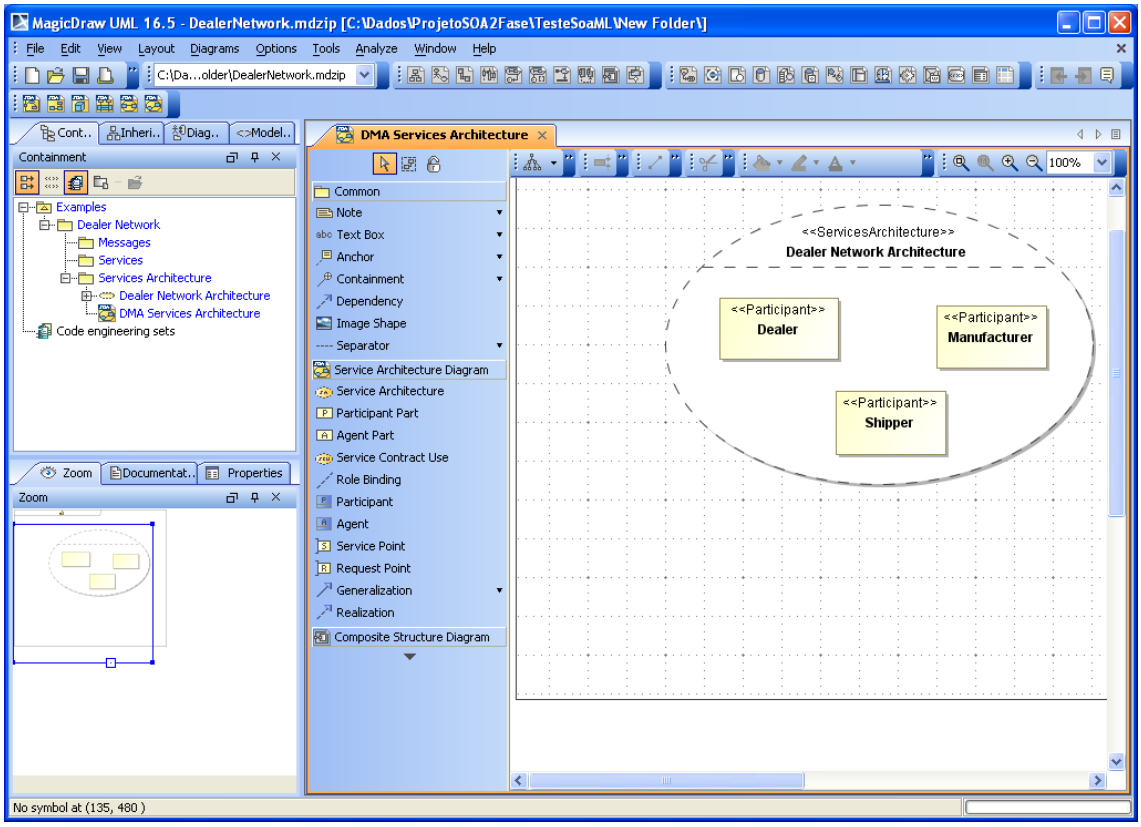

Figura 22 – Arquitetura de serviços com os participantes inseridos

Em seguida são criadas as estruturas básicas para os serviços. Para cada estrutura de serviço criamos um pacote para organizar melhor o nosso projeto. O primeiro pacote denominamos *Place Order Services*. Dentro desse pacote geramos o diagrama de estrutura do serviço (*Service Structure Diagram*) como exibido na Figura 23.

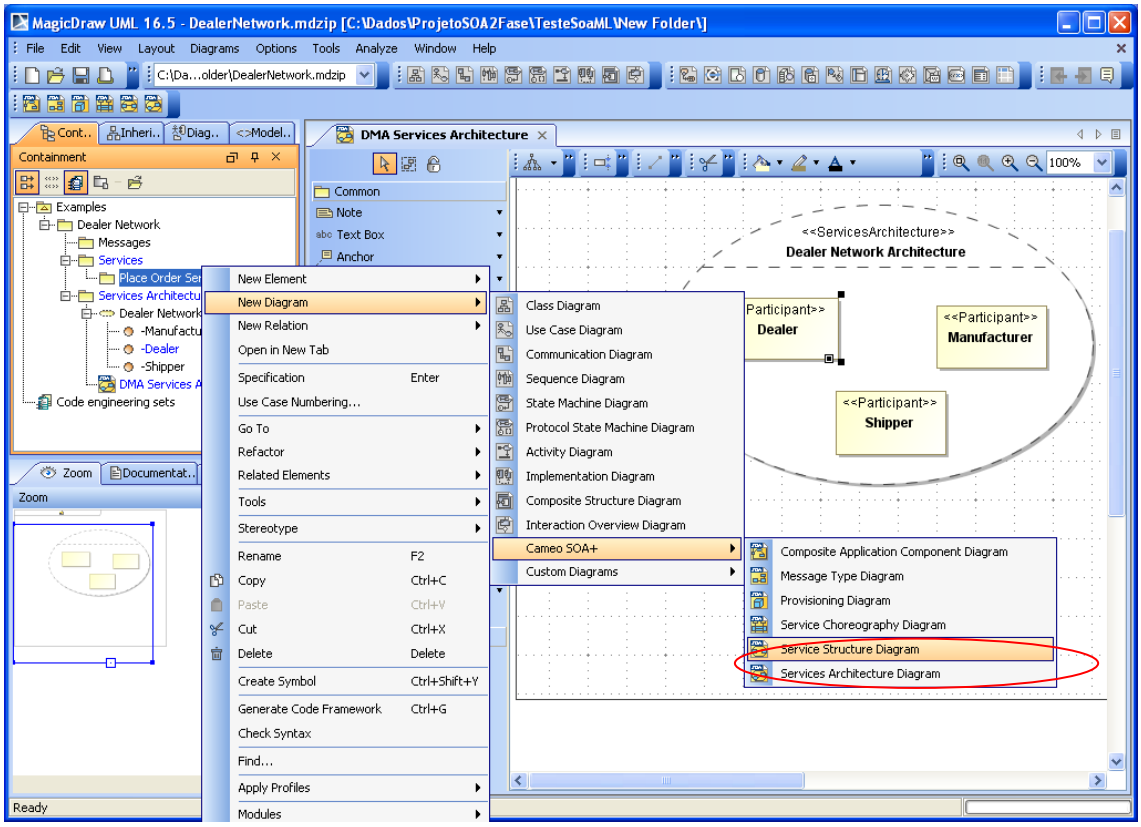

Figura 23 – Criação de um diagrama para definir os serviços

No diagrama *Service Structure Diagram*, modelamos os *ServicesContracts*, onde é definido como as partes (provedor e consumidor) interagem baseados em quais regras. A Figura 24 mostra o *ServiceContract* para o serviço *PlaceOrder*. Podemos verificar que três elementos foram inseridos no diagrama: *ServiceContract: Place Order* representado em linhas pontilhadas; *Consumer Part: Order Place* representado em linhas tracejadas finas; *Provider Part: Order Taker* representado em linhas sólidas. Os elementos *Consumer Part* e *Provider Part* foram conectados mostrando que eles interagem.

Para cada *Consumer Part* ou *Provider Part* uma *Service Interface* é criada, como as exibidas em linhas tracejadas largas na Figura 24.

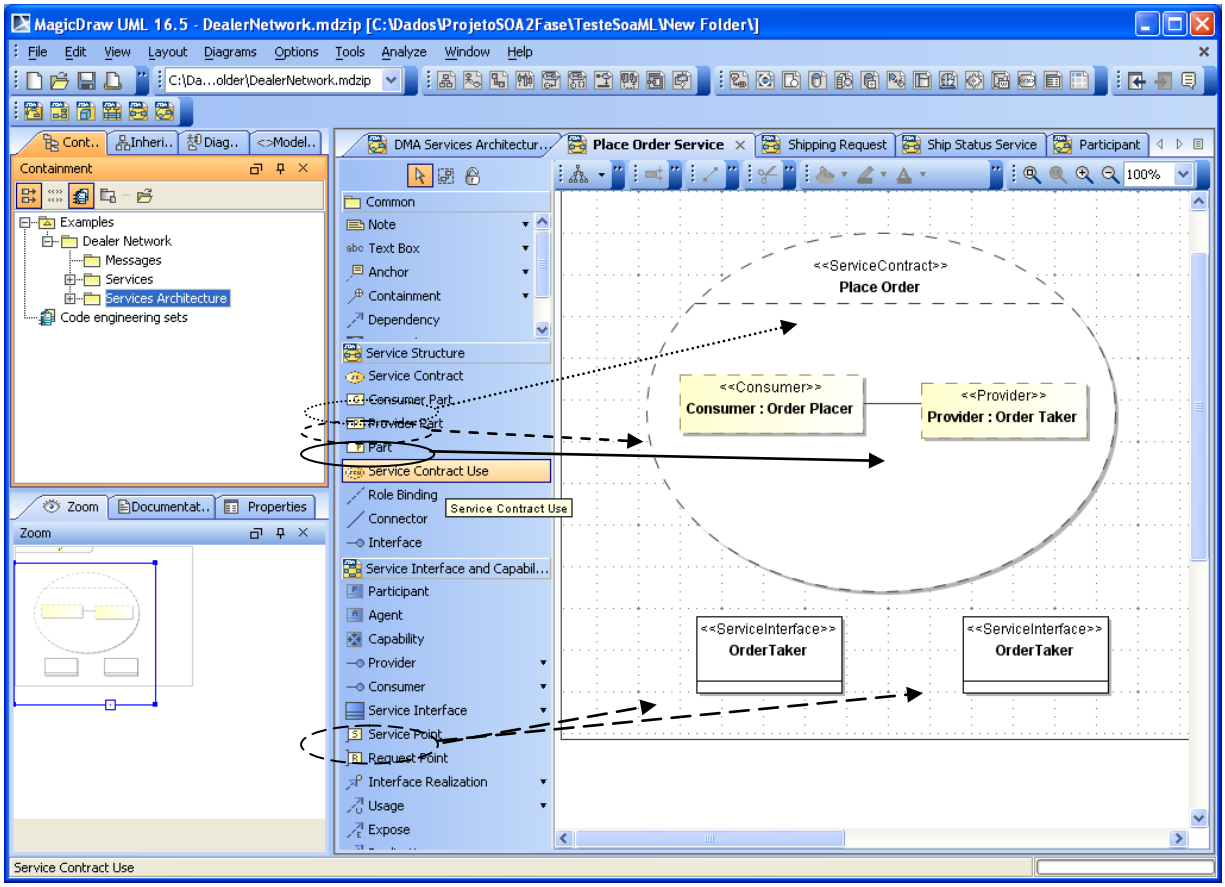

Figura 24 – Diagrama dos *ServiceContract Place Order*

O mesmo é feito para os serviços *ShippingRequest* e *ShipStaus*, ou seja, um *ServiceContract* é criado para cada um dos dois indicando os papeis, como mostrado nas Figura 25 e Figura 28.

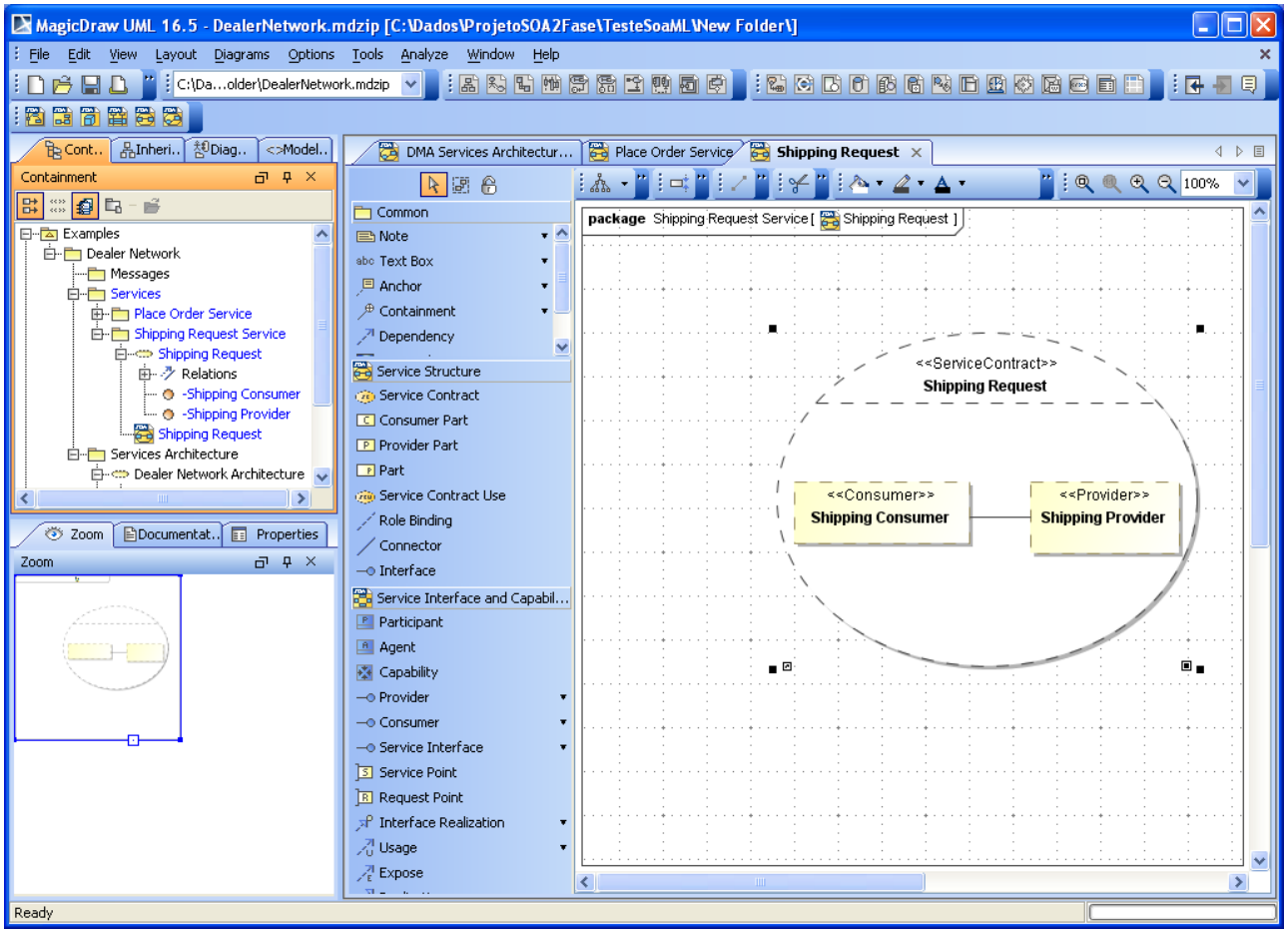

Figura 25 – *ServicesContract* para o *Shipping Request*

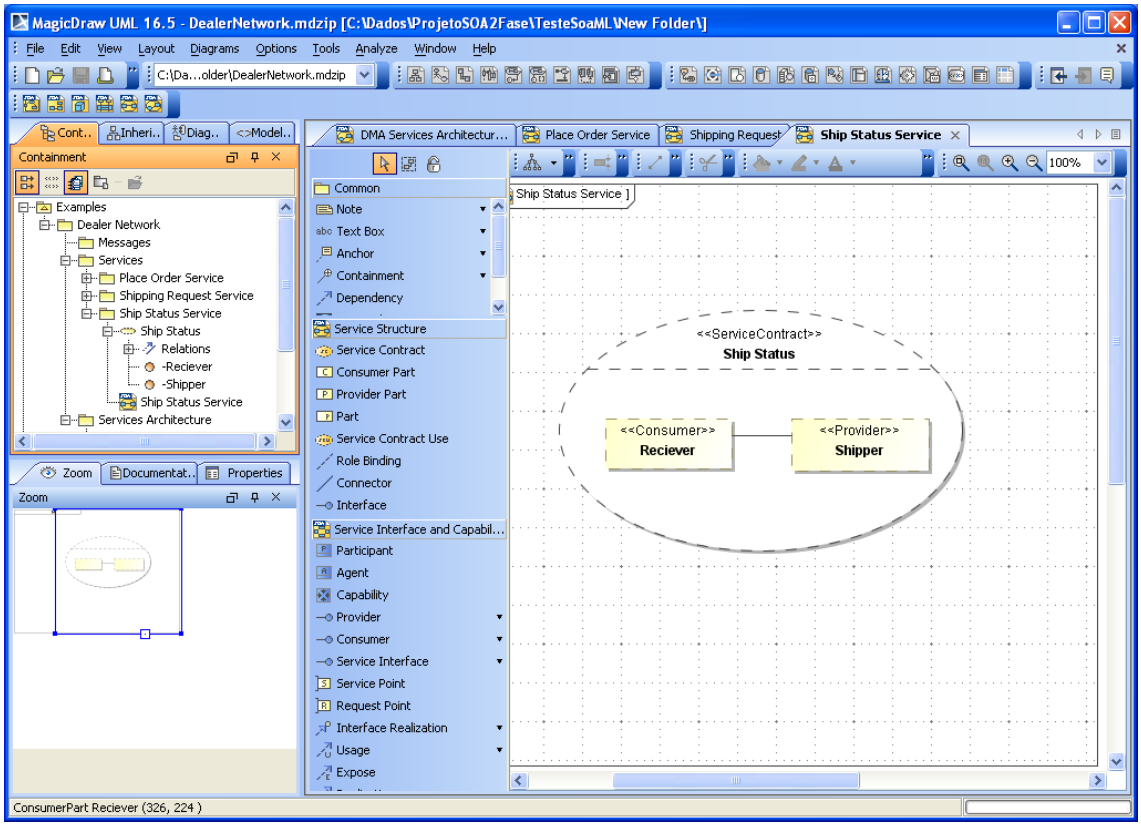

Figura 26 – *ServicesContract* para o *Ship Status*

Tendo definido todos os *ServicesContract* para o cenário em questão, podemos voltar ao diagrama da arquitetura de serviços para então representar os *ServicesContracts* envolvidos. Isto é feito arrastando o elemento *ServiceContract* que desejamos para dentro do diagrama da arquitetura de serviços. Em seguida indicamos qual participante desempenha o papel especificado no *ServiceContract* , utilizando o elemento *role binding*. Ao inserir o elemento role binding, os papeis do *ServiceContract* ao qual o participante foi unido são exibidos para que indiquemos qual deles o participante executa, como mostra a Figura 27.

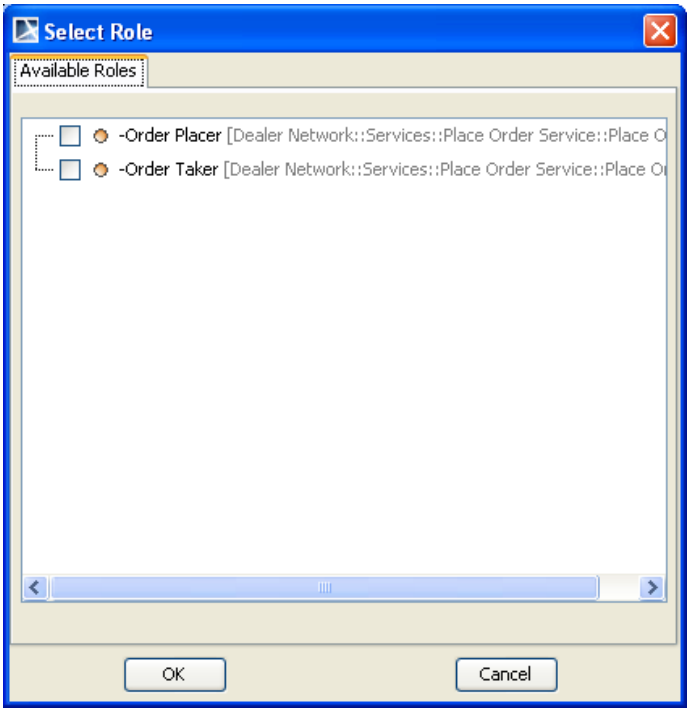

Figura 27 –Tela onde o papel desempenhado pelo participante é indicado

A Figura 28 mostra o diagrama completo para a arquitetura de serviços para o nosso exemplo de cenário.

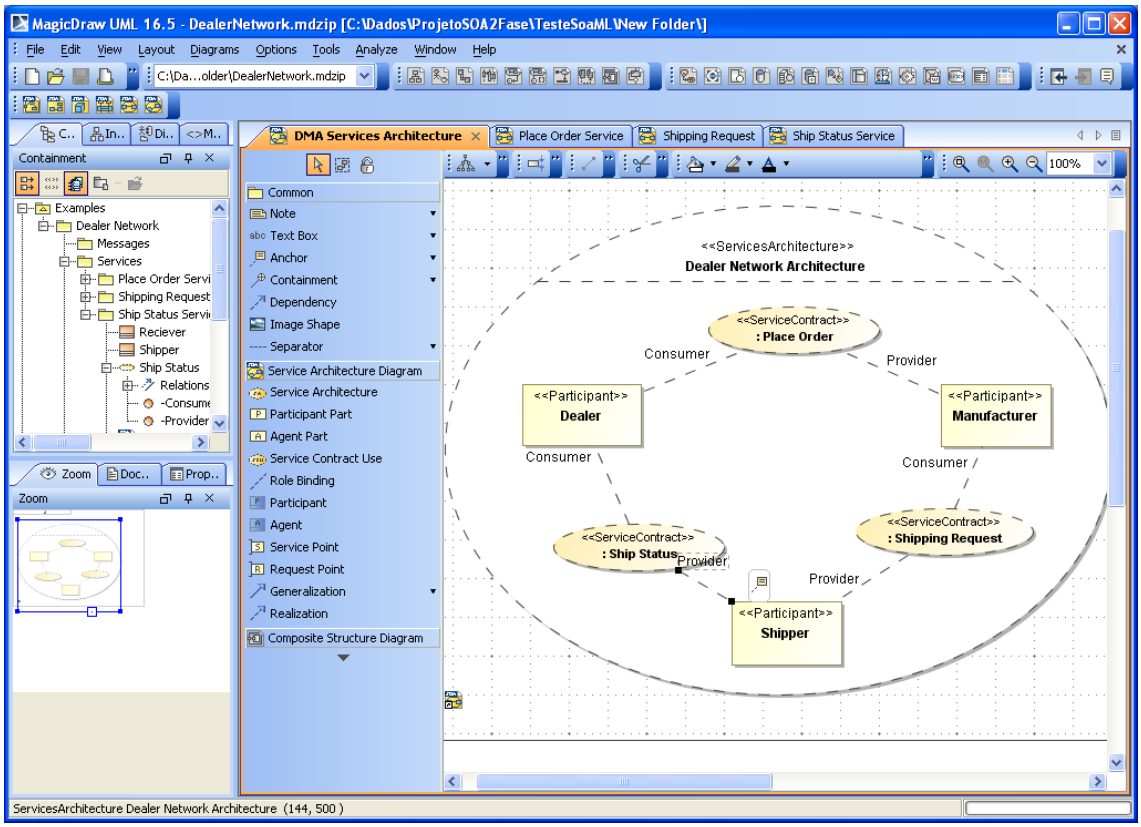

Figura 28 – Arquitetura de serviços considerando os *ServicesContracts*

Se outra arquitetura de serviços for criada os participantes e *ServicesContracts* definidos podem também ser utilizados, caso desejado.

Para completar o diagrama da arquitetura de serviços é preciso indicar o tipo dos Participantes, ou seja indicar a classe a qual eles pertencem. Existem três classes de Participantes no nosso exemplo. Essas três classes são criadas através do elemento *Participant* no diagrama de arquitetura de classe, como mostra a Figura 32.

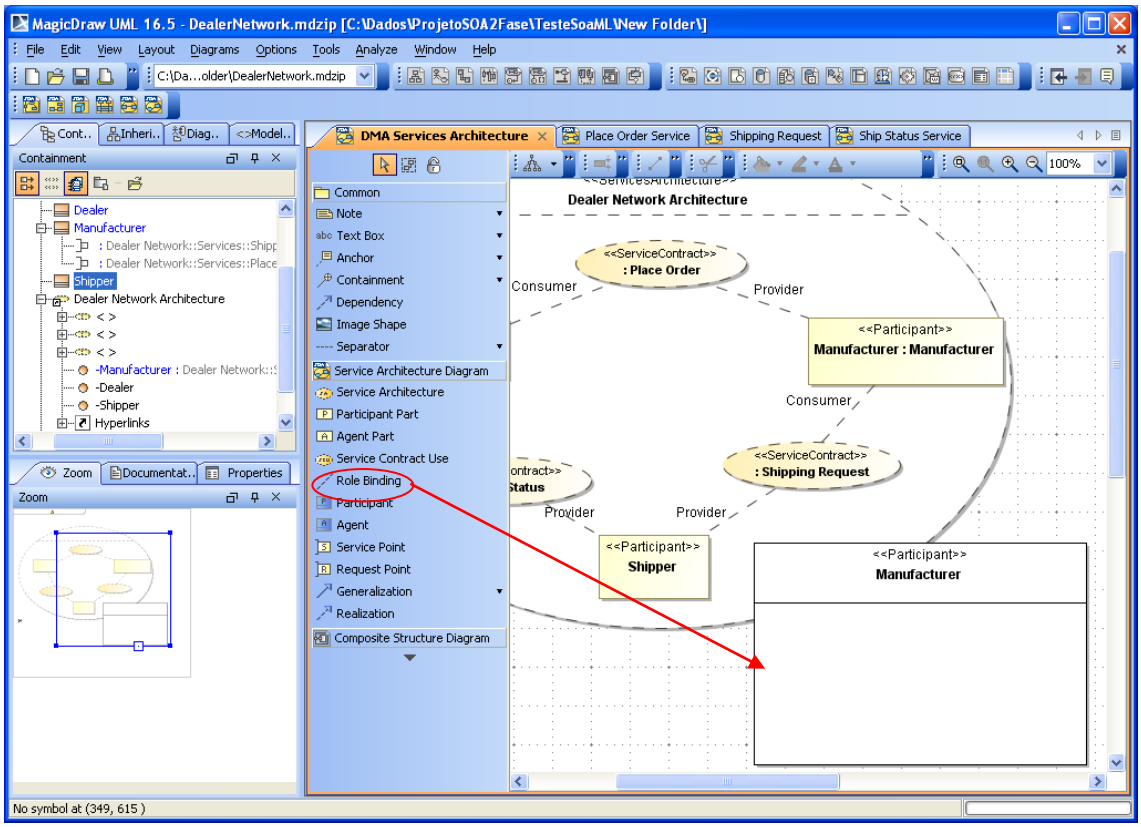

Figura 29 – Acrescentando o Participante *Manufacturer*

Em seguida, associamos ao elemento *Participant Part*, que temos no nosso diagrama de arquitetura de serviço, a classe *Participant* correspondente, através da criação de um novo tipo. A Figura 30 mostra como isto é feito.

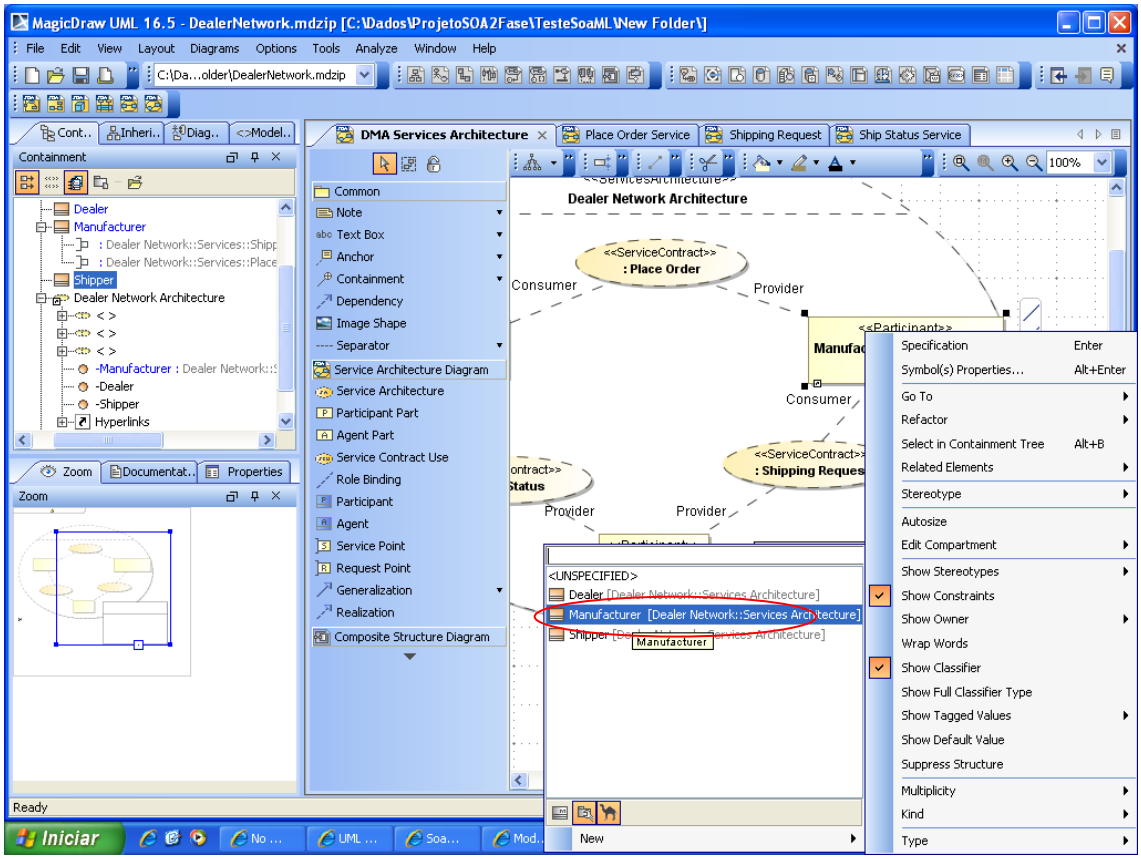

Figura 30 – Definição do tipo do Participante

Ao criar o tipo do participante, o elemento *Participant Part* fica envolvido com uma linha vermelha, indicando que alguma informação está faltando. Essa informação é o tipo do participante. O tipo do participante fornecedor, por exemplo, não tem especificado um *ServicePoint* ou *RequestPoint*, e isso não pode ocorrer tendo em vista que é através de um *ServicePoint* ou *RequestPoint* que ocorre a especificação do serviço que o participante provê ou consome. O tipo da porta deve ser o tipo do papel que o participante executa; logo, a especificação do *ServicePoint* e do *RequestPoint* é feita através de associação com os papeis assumidos pelo participante em questão. Por exemplo, para o participante fornecedor, temos que ele assume o papel de consumidor do serviço *ShippingRequest* e provedor do serviço *PlaceOrder*, dessa forma ele tem duas portas uma *RequestPoint* e a outra *ServicePoint* respectivamente como mostra a Figura 31.

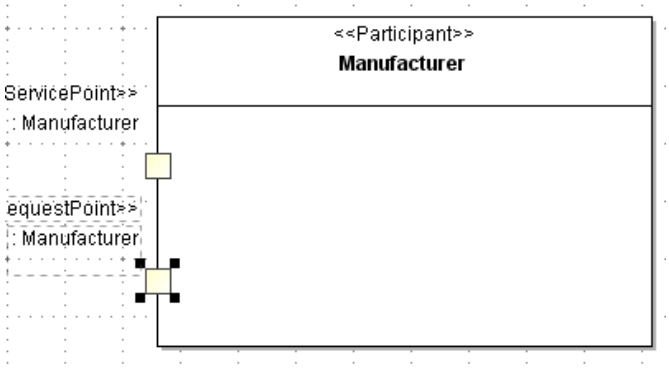

Figura 31 – Especificação do tipo do participante

Para ficar mais claro quais serviços os participantes proveem e consomem, criamos um diagrama de arquitetura de serviço para representar isso, como mostra a Figura 32.

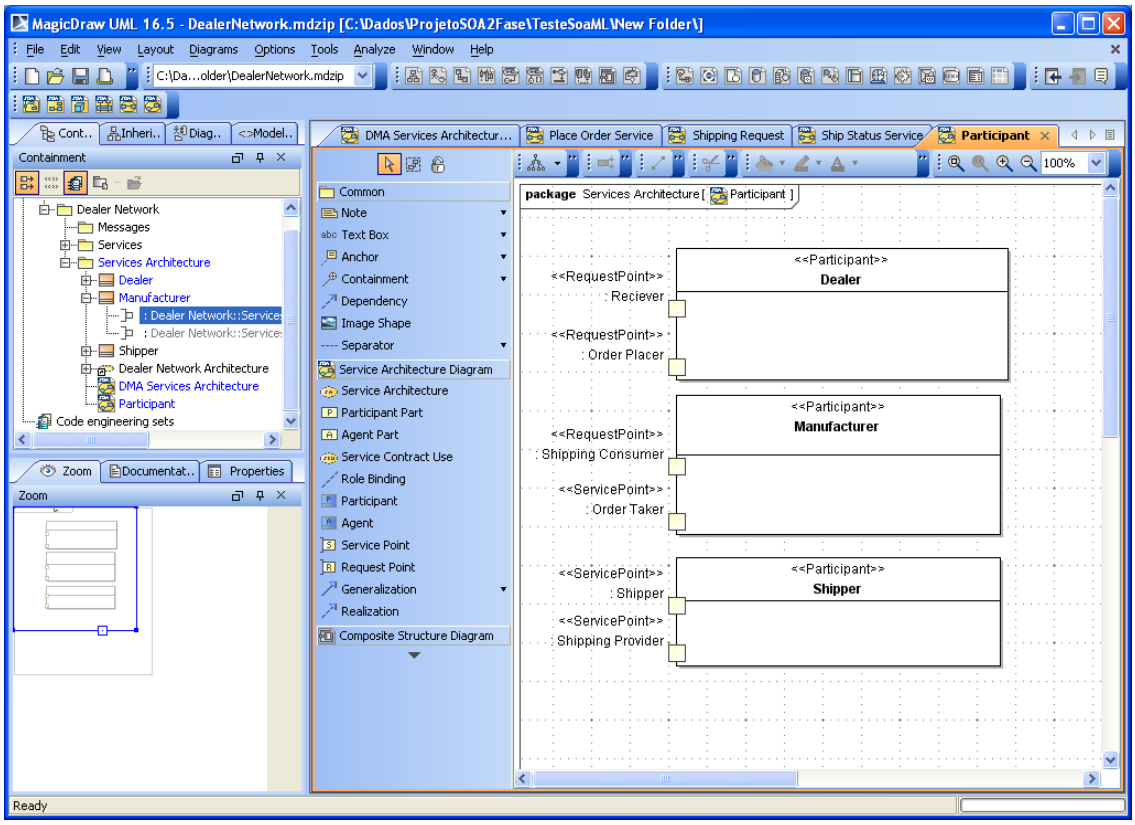

Figura 32 – Diagrama de especificação dos participantes

3.6 *MessageType*

MessageType é a especificação da informação trocada entre serviços consumidores e provedores. Esse estereótipo estende *Metaclasse*, *datatype* e *class*.

Um *MessageType* é um tipo de valor de objeto que representa informações trocadas entre requisições de um participante e serviços. Esta informação consiste de dados passados para dentro e/ou retornados a partir da invocação de uma operação ou sinal de evento definido em uma interface de serviço.

MessageTypes são utilizados para agregar entradas, saídas e exceções para operações de serviços como em WSDL. *MessageType* representa "dados puros" que podem ser comunicados entre as partes. É responsabilidade então das partes, baseado na especificação SOA, interpretar estes dados e agir de acordo. Como "dados puros" *MessageTypes* não podem ter dependências no ambiente, localização ou sistemas de informação de ambas as partes. Uma boa prática de projeto sugere que o conteúdo e a estrutura da mensagem fornecem uma interação rica entre as partes sem necessariamente acoplar ou restringir seu comportamento ou interesses internos.

Algumas restrições do MessageType são:

- Não pode conter operações próprias
- Não pode conter *Behaviors* próprios

- Todos os atributos devem ser *Public*
- Todos os atributos devem ser *PrimitiveType*, *DataType* ou outro *MessageType* ou uma referência para um desses tipos.

MessageTypes representam dados de serviço trocados entre serviços consumidores e provedores. Dados de serviço geralmente são uma visão de informações ou modelos de classe de um domínio, representando os dados da entidade usada para implementar participantes de serviço. *MessageType* pode ser utilizado para representar entradas, saídas e exceções de operações de serviço em um tipo baseado em direção. A notação utilizada é como um DataType na UML2 com <<messageType>>.

A Figura 33 mostra exemplos de *MessageTypes*. O *MessageType ShipmentStatus*, por exemplo, contém três atributos: *waybaill number* do tipo *String*, *shipped* do tipo *Boolean* e *delivery date* do tipo data.

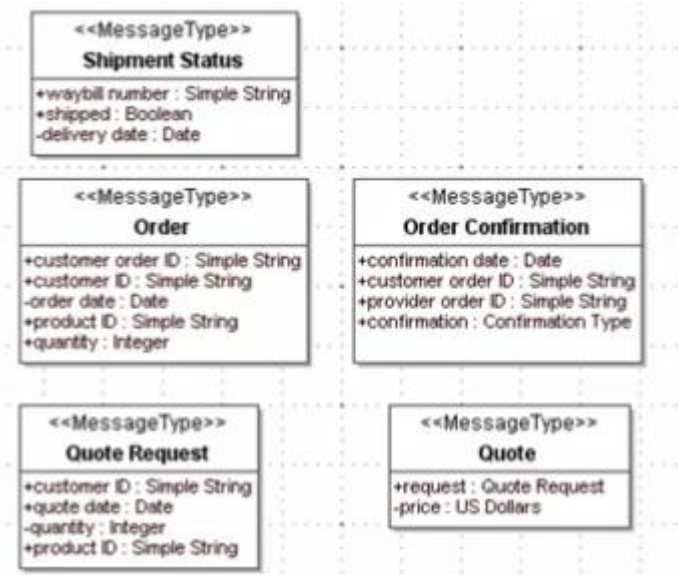

Figura 33 – Exemplos de *MessageType*

MessageTypes podem ter associações com outros *MessageTypes* e *Datatypes*, como apresentado na Figura 34 pela relação entre *POMessage* e *Customer*. Tais associações devem ser agregações.

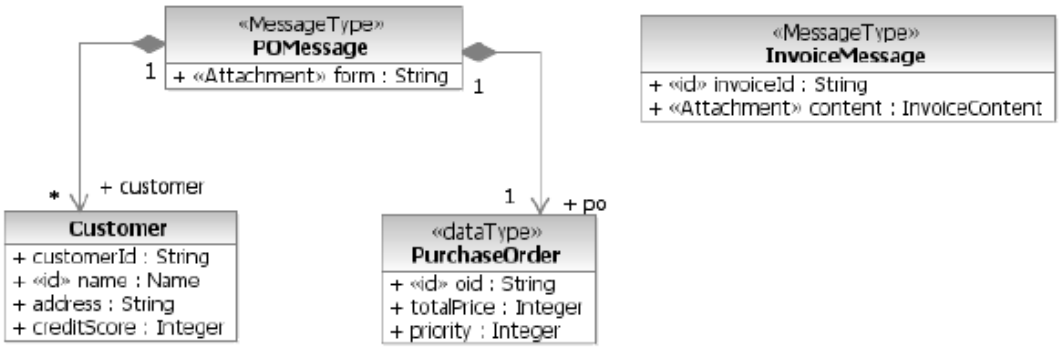

Figura 34 – Exemplos de *MessageType* com associações com outros *MessageTypes* e *DataTypes*

Na modelagem dos parâmetros dos métodos de serviços, pode-se utilizar estilo documento ou estilo RPC. A Figura 35 mostra dois exemplos da Interface *Purchasing* considerando estes casos. O primeiro exemplo usa parâmetros com estilo documento ou mensagem, onde o tipo dos parâmetros são o *MessageTypes* mostrado na Figura 34. A segunda versão usa um estilo mais "orientado a objeto" *Remote Procedure Call* (RPC), que suporta várias entradas, saídas e um valor retornado. A escolha de uso depende da preferência do modelador e possivelmente na plataforma que será utilizada. Algumas plataformas como *WebServices* e WSDL exigem parâmetros com estilo mensagem, que podem ser criadas de ambos os estilos de modelagem. E possível traduzir um estilo RPC para parâmetros de mensagem na transformação, e esse é o propósito do estilo de mensagem empacotado em documento de WSDL. Mas isto pode resultar em muitas mensagens WSDL contendo a mesma informação que pode causar problemas de interoperabilidade na plataforma de execução.

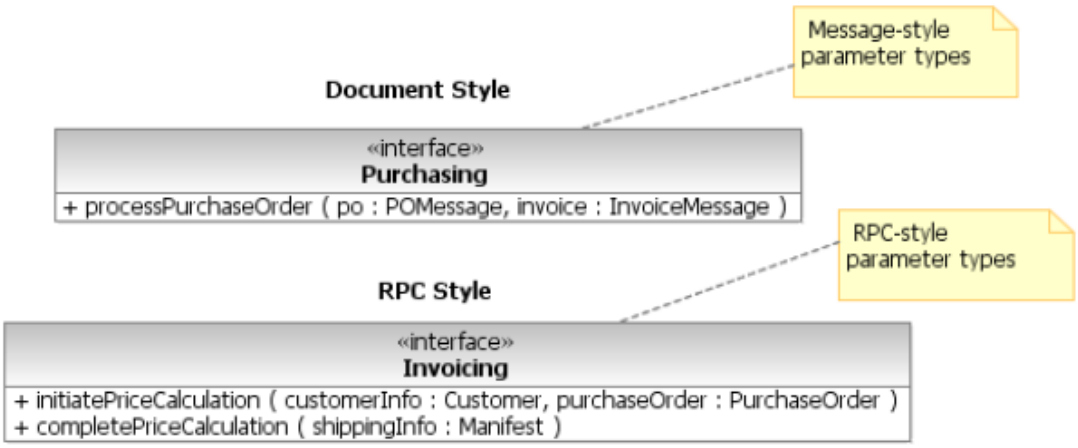

Figura 35 – Exemplos de interface que utilizam um estilo de documento e um estilo RPC

O relacionamento entre *MessageTypes* e as entidades classificadoras que atuam como fonte de dados é estabelecida pela semântica do serviço. Como os parâmetros do serviço pegam seus dados das entidades do domínio e como estas entidades de domínio são atualizadas (baseadas em mudanças nos dados dos parâmetros) é responsabilidade da implementação do serviço.

A

Figura **36** mostra o uso dos *MessageTypes*. A *ServiceInterface Principal Query* tem três operações definidas. Estas operações têm parâmetros de entrada e saída que são do tipo *MessageType*, como por exemplo a operação *FindPrincipals*, que tem como parâmetro de entrada *findPrincipals* e como saída *PrincipalList*, ambas *MessageTypes*. Um estereótipo *MessageType* é definido então para cada um, onde só é possível ter atributos e nenhuma operação.

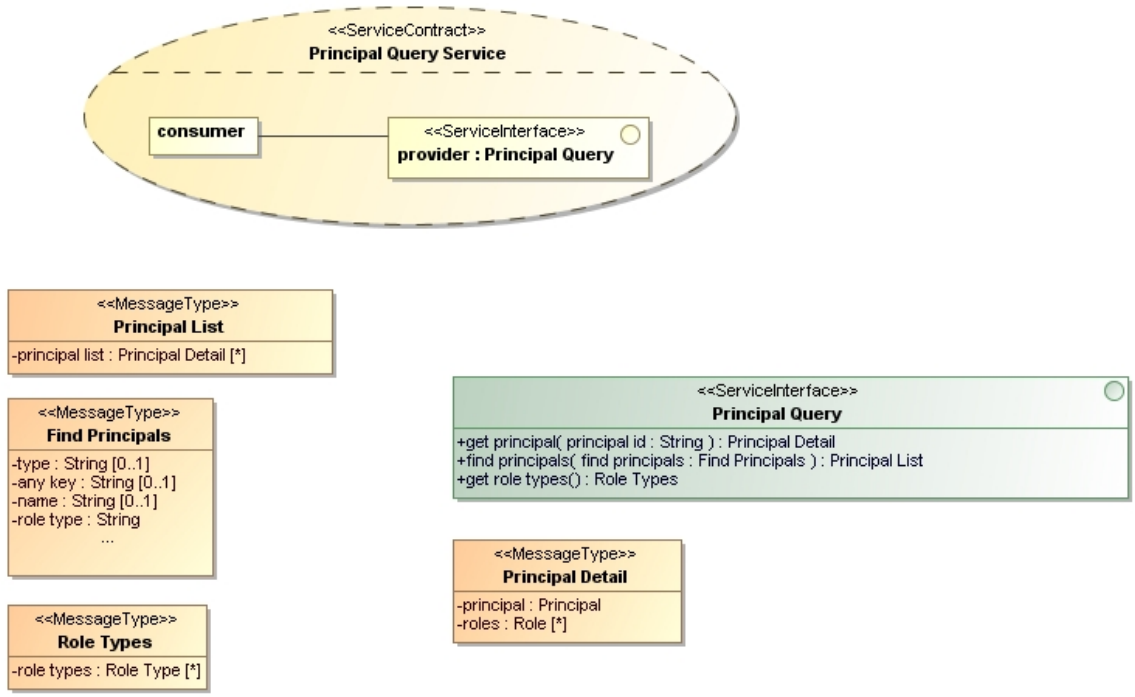

Figura 36 – Exemplos de uma *ServiceInterface* que tem operações com parâmetros do tipo *MessageType*

3.7 Provision

Uma questão importante é como os serviços especificados serão disponibilizados e por que tecnologia. SoaML fornece o estereótipo Provision, onde é especificada a tecnologia que proverá o serviço e que serviço será provido. A Figura 37 descreve que o serviço provido pelo participante *Dealer* será provido através da tecnologia JEE. O elemento *Provision* é inserido no diagrama de *Provisioning* e associamos a ele o estereótipo JEE *Provisioning* indicando a tecnologia a ser utilizada. O participante *Dealer* é inserido e alteramos o estereótipo dele para JEE Web Service para indicar que o serviço que ele prove será implementado como um web service.

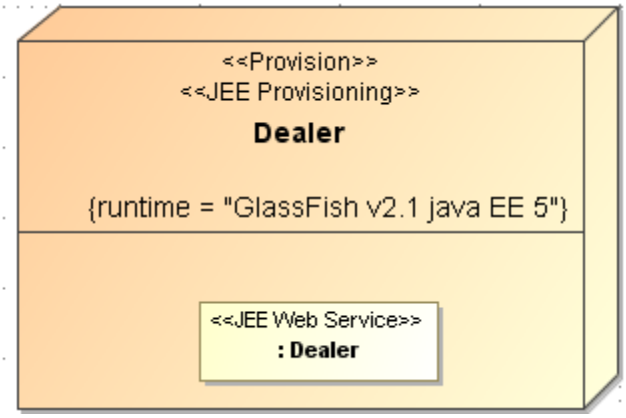

Figura 37 – Exemplo do estereótipo *Provision*

A partir dessa especificação é possível gerar código utilizando o Eclipse.

4 Conclusões

Neste relatório abordamos os principais componentes do profile SoaML. O profile SoaML estende UML com novos estereótipos, onde os principais são: *Participant*, *ServiceArchitecture*, *ServiceContract* e *ServiceInterface*.

A utilização da especificação de serviços com o profile SoaML permite a geração automática de código desde que o estereótipo *Provision* tenha sido utilizado, indicando a tecnologia onde será implementado e o serviço que será provido.

5 Referências Bibliográficas

AZEVEDO, L. G.; SANTORO, F. M.; BAIAO, F.; SOUZA, J. F.; REVOREDO, K.; PEREIRA, V. B.; HERLAIN, I.. **A Method for Service Identification from Business Process Models in a SOA Approach**. In: 10th International Workshop on Business Process Modeling, Development, and Support (BPMDS), Amsterdam, 2009a.

AZEVEDO, L. G.; BAIAO, F.; SANTORO, F. M.; SOUZA, J. F.; REVOREDO, K.; PEREIRA, V. B.; HERLAIN, I. **Identificação de serviços a partir da modelagem de processos de negócio**. In: Simpósio Brasileiro de Sistemas de Informação (SBSI), Brasília, 2009b.

AZEVEDO, L.; SOUSA, J., BAIÃO, F., SANTORO, F., **Relatório sobre Análise e Projeto de Serviço**, Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT 0023/2009, 2009c.

AZEVEDO, L.; PEREIRA, V.; BAIAO, F.; SANTORO, F.; SOUZA, J.; REVOREDO, K. **Relatório de Conceituação em SOA,** Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT 0012/2009, 2009d.

AZEVEDO, L.; PEREIRA, V.; BAIAO, F.; SANTORO, F.; SOUZA, J.; REVOREDO, K. **Relatório de Metodologias de Identificação de Serviços**, Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT 0021/2009, 2009e.

HECKEL, R.; LOHMANN, M.; THÖNE, S. **Towards a UML Profile for Service-Oriented Architectures**, Workshop on Model Driven Architecture: Foundations and Applications, 2003.

JOHNSTON, S. **UML 2.0 Profile for Software Services,** 2005a. Disponível em <http://www.ibm.com/developerworks/rational/library/05/419_soa>.

MODELDRIVEN. **Ferramenta ModelPro**, 2009. Disponível em <http://portal.modeldriven.org/project/modelpro>.

LÓPEZ-SANZ, M.; ACUÑA, C.J.; CUESTA, C.E.; MARCOS, E. **UML Profile for the Platform Independent Modelling of Service-Oriented Architectures**, First European Conference, ECSA 2007 Aranjuez, Spain, September 24-26, 2007.

OMG. **Catalog of UML Profile Specifications**, 2009a. Disponível em <http://www.omg.org/technology/documents/profile_catalog.htm>.

OMG. **Service-oriented Architecture Modeling Language (SoaML),** 2009b. Disponível em <http://www.omg.org/docs/ad/08-08-04.pdf>.