PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL

FACULDADE DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UMA INFRA-ESTRUTURA DE INTEGRAÇÃO DE SISTEMAS UTILIZANDO NOTIFICAÇÕES POR MEIO DE *WEB SERVICES*

EVERTON SEBASTIANY REISDORFER DEWES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, pelo Programa de Pós-Graduação em Ciência da Computação da Faculdade de Informática da Pontifícia Universidade Católica do Rio Grande do Sul.

Orientadora: Profa. Dra. Karin Becker

Porto Alegre

2005

PUC

Campus Central
Av. Ipiranga, 6681 - prédio 16 - CEP 90619-900
Porto Alegre - RS - Brasil
Fone: +55 (51) 3320-3544 - Fax: +55 (51) 3320-3548
Email: bceadm@pucrs.br
www.pucrs.br/biblioteca

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
Reconhecido pelo Parecer No. 930/98.C.N.E. Homologação Publicada no D.O.U. de 30/12/98.

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "***Uma Infra-Estrutura de Integração de Sistemas Utilizando Notificações por Meio de Web Services***", apresentada por **Everton Sebastiany Reisdorfer Dewes,** como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Sistemas de Informação, aprovada em 18/11/2005 pela Comissão Examinadora:

Profa. Dra. Karin Becker – PPGCC/PUCRS
Orientadora

Prof. Dr. Avelino Francisco Zorzo – PPGCC/PUCRS

Prof. Dr. Marcelo Blois Ribeiro - PPGCC/PUCRS

Prof. Dr. Sérgio Crespo Coelho da Silva Pinto – UNISINOS

Homologada em 2/10/2006, conforme Ata No. 24 pela Comissão Coordenadora.

Prof. Dr. Avelino Francisco Zorzo
Coordenador.

AV. IPIRANGA, 6681 - PRÉDIO 16/SALA 106
90619-900 - PORTO ALEGRE -RS -BRASIL
TELEFONE: 55 51 33203611 FAX 55 51 3320.3621
http: //www.inf.pucrs.br e-mail: ppgcc@inf.pucrs.br

AGRADECIMENTOS

Aos meus pais, Ilse e José, e a meu irmão Heitor, pelo apoio e motivação.

A minha orientadora, Karin Becker, por ter compartilhado comigo seu conhecimento e sua paciência durante o período do mestrado.

Ao Programa de Pós-Graduação em Ciência da Computação pela ótima infra-estrutura disponível para os alunos.

Ao Centro de Desenvolvimento e Pesquisa (CDPe), do Convênio DELL/PUCRS, pelo financiamento do meu curso de mestrado.

Aos meus amigos que souberam compreender as minhas ausências.

Muito Obrigado!

RESUMO

A integração de aplicações pode ser identificada como uma tarefa vital devido a muitos dos requisitos dos sistemas de informação envolverem a comunicação com diferentes parceiros (ex. clientes, fornecedores, aplicativos internos) em tempo real. Porém, o desenvolvimento dessa interoperabilidade é caro e consome tempo. *Web Services* fornecem uma solução interessante com relação a esses problemas, permitindo aos sistemas trocarem informação com um pequeno esforço de integração e maior flexibilidade. Porém, ainda existem problemas em aberto na integração de sistemas com *Web Services*.

A interoperação com base em mensagens desacopla os sistemas, focando no controle sobre as interações do sistema e a troca de informação. Dessa forma, ela ajuda a resolver alguns dos problemas encontrados na integração direta de sistemas tradicional. A *WS-Notification* é uma padronização de troca de mensagens para *Web Services*, composta por um conjunto de especificações abertas que usam *Web Services* para trocar informações através de mensagens assíncronas. Porém, muitos sistemas de informação (legados) não estão preparados para interoperar com outros sistemas através de mensagens e, conseqüentemente, necessitam ser adaptados especialmente para este tipo de integração.

Este trabalho propõe uma Infra-Estrutura de integração que guia a adaptação de sistemas de informação tradicionais e suas integrações, considerando a padronização *WS-Notification*. O projeto da Infra-Estrutura proposta tem base em questões identificadas que necessitam ser tratadas para a integração de sistemas usando mensagens de notificação. Padrões arquiteturais e de projeto que podem ser aplicados para resolver essas questões foram identificados, resultando na definição de um conjunto de componentes que, juntos, formam a Infra-Estrutura de integração. A Infra-Estrutura é detalhada em termos de seus componentes, das questões que ela trata, bem como dos seus relacionamentos com outros componentes e sistemas aos quais ela é integrada. A Infra-Estrutura foi aplicada em três estudos de caso com diferentes questões de integração e uma análise de desempenho preliminar foi desenvolvida.

Palavras-chave: Integração de Sistemas, *Web Services*, EAI, *WS-Notification* e Padrões de Projeto.

ABSTRACT

Application integration is a critical task, since it requires dealing with information systems of different partners (e.g. costumers, suppliers, internal applications) on-line. However, the development of this interoperability is expensive and time consuming. Web Services provide interesting solutions with regard to these problems, allowing systems to exchange information with less integration efforts and more flexibility. However, there are still open problems in the integration of systems with Web Services.

Message-based interoperation decouples systems, focusing on the control over system interaction and exchanged information. In this way, it helps solving some of the problems faced in the traditional integration of systems. WS-Notification is a Web Service messaging standard that is composed of a set of open specifications that use Web Services to exchange information among applications through asynchronous messages. However, many (legacy) information systems are not prepared to interoperate with other systems through messages, and therefore, they need to be specifically adapted for this kind of integration.

This work proposes an integration Infra-Structure that guides the adaptation of conventional information systems and their integration, considering the standard WS-Notification. The design of the proposed Infra-Structure was based on the identification of issues that must be handled for system integration using message notification. Design and architectural patterns that could be applied to solve these issues were identified, resulting in the definition of a set of components that, together, form the integration Infra-Structure. The Infra-Structure is detailed in terms of its components, the issues they handle, as well as their relationships with other components and with systems that are integrated. The Infra-Structure is applied in three case studies with different integration issues, and a preliminary performance analysis was developed.

Key-words: Systems Integration, Web Services, EAI, WS-Notification and Design Patterns.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

Sigla	Descrição
ACT	*Application Center Test*
BPEL4WS	*Business Process Execution Language for Web Services*
BSC	*Business-Service-Computing*
CORBA	*Common Object Request Broker Architecture*
DBC	*Desenvolvimento Baseado em Componentes*
EAI	*Enterprise Application Integration*
EDA	*Event-Driven Architecture*
EDI	*Electronic Data Interchange*
EJB	*Enterprise JavaBeans*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
IIS	*Internet Information Services*
J2EE	*Java 2 Enterprise Edition*
JMS	*Java Message Service*
JSP	*JavaServer Pages*
MSDE	*Microsoft SQL Server Desktop Engine*
MSMQ	*Microsoft Message Queuing*
RPC	*Remote Procedure Call*
SCM	*Supply Chain Management*
SGML	*Standard Generalized Markup Language*
SMTP	*Simple Mail Transfer Protocol*
SOA	*Service Oriented Architecture*
SOAP	*Simple Object Access Protocol*
UDDI	*Universal Description, Discovery and Integration*
UML	*Unified Modeling Language*
URI	*Universal Resource Indicator*
W3C	*World Wide Web Consortium*
WS	*Web Services*
WSDL	*Web Services Description Language*
WSFL	*Web Services Flow Language*
WS-I	*Web Services Interoperability*
WSIL	*Web Services Inspection Language*
XLANG	*Web Services for Business Process Design*
XML	*Extensible Markup Language*
XSD	*XML Schema*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, grandes e pequenas empresas consumidoras de *software* já possuem boa parte de seus processos internos informatizados. Essas aplicações guardam a inteligência da organização e são o resultado de investimentos realizados em diversas épocas, refletindo na arquitetura de seus sistemas várias fases tecnológicas.

Com o passar do tempo, sistemas de *software* precisam ser mantidos, modificados e integrados a outros sistemas ou, então, se tornam obsoletos. Essa é uma evolução natural, pois, com o crescimento das empresas e alterações constantes dos ambientes de computação, surge a necessidade da melhoria e integração de suas aplicações.

A integração de aplicações pode ser identificada como uma tarefa vital devido a muitos dos requisitos dos sistemas de informação envolverem a comunicação com clientes, fornecedores, parceiros e aplicativos internos, muitas vezes em tempo real. Porém, habilitar essa interoperabilidade é, normalmente, um trabalho árduo que consome muito tempo e recursos. A questão de integração de sistemas através de protocolos-padrão que possuem baixo acoplamento não é nova. Por exemplo, EDI (*Eletronic Data Interchange*) é um mecanismo de integração que possui um baixo acoplamento e permite a integração de sistemas de diferentes plataformas, já sendo utilizado com sucesso em projetos de integração.

Web Services [1] [2] vêm ao encontro dos mesmos objetivos, que consistem em permitir que sistemas diferentes troquem informações com o mínimo de esforço de integração e máxima flexibilidade. *Web Services* são aplicações disponíveis através da Internet que fornecem algum tipo de serviço para outras aplicações, tanto de processamento como de informação. Diferenciam-se de outras aplicações para a *Web*, pois envolvem, normalmente, comunicação entre sistemas e não são projetados para serem acessados diretamente pelo usuário. Ainda, clientes desses serviços podem ser escritos em qualquer plataforma desde que suportem os protocolos utilizados pelos *Web Services*. O uso de tais tecnologias padronizadas reduz a heterogeneidade e, dessa forma, facilita a integração de aplicações [3].

Uma arquitetura de *software* diz respeito à organização do sistema de software por completo [4] [5] [6] [7], e envolve a seleção dos elementos estruturais, suas interfaces e a composição desses elementos. Essa arquitetura é utilizada como uma estrutura básica que permite o entendimento dos componentes do sistema e seus relacionamentos internos e externos.

Os *Web Services* renovaram a idéia de arquitetura orientada a serviços (*Service Oriented Architecture - SOA*). SOA é, em essência, uma forma de projetar um sistema de *software* para fornecer serviços tanto para aplicações de usuário quanto para outros serviços através da

utilização de publicações e descobertas de interfaces [8]. Uma arquitetura orientada a serviços pode oferecer muitos benefícios, como uma redução no custo de manutenção em sistemas de informação que sofrem alterações constantes, acelerando o processo de desenvolvimento através do reuso, melhorando a utilização de recursos e reduzindo a redundância. Além dos benefícios já apresentados, SOA consiste em uma arquitetura que promove o baixo acoplamento, o que facilita o reuso e permite que as aplicações funcionem independentemente.

A utilização de *Web Services* para a integração de aplicações pode ser realizada com diversos formatos e configurações em uma arquitetura orientada a serviços. Um cenário normal é a integração direta entre um cliente e um servidor remoto. Essa solicitação, muitas vezes, tem um caráter de sincronização de estados, ou seja, realiza uma busca por informações que eventualmente tenham sido alteradas ou adicionadas, o que pode não ter ocorrido. Essa é uma forma totalmente síncrona de interação, na qual o cliente fica paralisado até que o servidor retorne os dados solicitados. Em uma situação comum, como esquematizada na Figura 1, uma aplicação cliente entra em contato com diversas aplicações servidoras para a realização de suas tarefas. Já do ponto de vista do servidor, é possível que existam diversos clientes acessando os seus serviços, cada qual com seu objetivo específico. Pode-se, então, visualizar um cenário no qual existe uma combinação de interações de diversos clientes para diversos servidores (Figura 2).

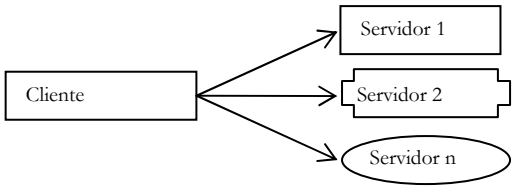

Figura 1 – Um cliente interagindo com diversos servidores (1:N).

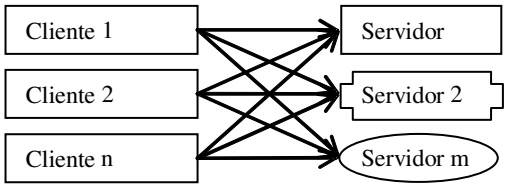

Figura 2 – Diversos clientes interagindo com diversos servidores (N:M).

Essa configuração típica de grandes empresas é ineficiente, pois:

1. Para a integração, é necessário realizar a adaptação de diversos sistemas, tanto clientes como servidores, para que possam interoperar. Se existem 5 clientes distintos e 5 servidores distintos a serem acessados, podem ser necessárias até 25 adaptações.

2. Para a manutenção, cada alteração na interface de um servidor implicará modificação de todos os clientes que acessam o seu serviço. De forma análoga, a modificação de um cliente requer que ele seja novamente adaptado para o funcionamento com os servidores.
3. Sua execução acaba por sobrecarregar os servidores, pois a busca de dados para sincronização de estados faz com que os clientes realizem chamadas desnecessárias a serviços.

A utilização de comunicação assíncrona é uma forma de aperfeiçoar a troca de informações, pois permite a realização de tarefas em paralelo, economizando o tempo de espera por respostas e tornando o trabalho dos sistemas mais eficiente. É nesse ponto que as arquiteturas dirigidas a eventos (*Event-Driven Architecture - EDA*) tornam-se interessantes. Elas permitem que diversos sistemas cooperem de forma não intrusiva, estimulando o paralelismo de aplicações.

EDA é um paradigma arquitetural baseado na utilização de eventos como gatilhos, que informam diversos assinantes sobre um evento de tal forma que cada um deles possa tomar uma ação apropriada. Para o gerenciamento dos eventos, utiliza-se o padrão arquitetural *Broker* [9], que tem por função distribuir um conjunto de notificações e repassá-las para os sistemas interessados. Assim, sistemas podem trocar informações dinamicamente, sem necessariamente conhecerem seus parceiros de interação, sendo possível alterar regras, rotear mensagens entre diversos *Brokers*, filtrar e transformar mensagens, antes destas chegarem aos seus destinatários. Além disso, o baixo acoplamento permite a substituição parcial ou total de qualquer um dos sistemas que participam de uma arquitetura, de maneira totalmente transparente para clientes ou servidores.

Neste trabalho, apresenta-se um estudo sobre a integração de sistemas através de um *NotificationBroker*. Resumidamente, ele é especificado na *WS-Notification*[‡], sendo uma padronização para a utilização de eventos sobre *Web Services*, tendo como objetivo o gerenciamento de envio e recebimento de mensagens entre elementos publicadores e assinantes.

Como resultado deste estudo, propõe-se uma Infra-Estrutura capaz de facilitar a integração de sistemas com a padronização *WS-Notification*, permitindo que esses, de maneira transparente, possam trocar informações com os mais diversos sistemas, e que sua evolução ou até mesmo substituição não afete os demais participantes da arquitetura. Isso significa um ganho de esforço no que se refere ao projeto de adaptação e extensão de sistemas de informação, já que a Infra-Estrutura proposta identifica componentes necessários à integração através de

[‡] http://www-128.ibm.com/developerworks/library/specification/ws-notification/

notificações entre *Web Services*, detalhando seu papel, bem como os relacionamentos e interação com os demais componentes e sistemas a serem integrados.

A base da Infra-Estrutura proposta é a utilização de padrões de projeto e padrões arquiteturais ([9], [10]) como auxílio na solução das principais questões encontradas para integração de sistemas via *Web Services*, encontradas ao longo do desenvolvimento deste trabalho. Padrões tornaram-se parte integral do projeto e arquitetura de *software*, e os benefícios de sua utilização podem ser aplicados também aos *Web Services*.

1.1 Objetivos

Dadas as características de integração de sistemas, é extremamente útil ter uma estrutura que auxilie a configuração, adaptação e extensão de (sub)sistemas em um ambiente orientado a eventos. Assim, o objetivo deste trabalho é propor uma Infra-Estrutura que auxilie na integração de sistemas de informação através de mensagens, considerando que os sistemas a serem integrados não estejam originalmente preparados para interoperarem segundo o paradigma EDA.

Para o desenvolvimento deste trabalho, foram definidos os seguintes objetivos específicos:

1. Identificação de problemas existentes na integração de sistemas através de *Web Services*, e a contribuição de EDA à solução dos mesmos;
2. Identificação de questões de integração que devem ser resolvidas para que sistemas possam trabalhar de forma transparente com EDA;
3. Identificação de padrões de projeto e padrões arquiteturais que resolvam as questões de integração identificadas;
4. Especificação de uma Infra-Estrutura de integração que traga resposta às questões de integração identificadas, usando padrões de projeto/arquiteturais existentes;
5. Desenvolvimento de um estudo de caso, mostrando a aplicabilidade da Infra-Estrutura proposta, bem como testes de desempenho.

1.2 Estrutura do Documento

O restante deste documento está organizado da seguinte forma: O Capítulo 2 apresenta a classificação utilizada no decorrer do trabalho para a distinção dos sistemas de informação, permitindo, assim, limitar o seu escopo. O Capítulo 3 apresenta os principais conceitos utilizados à continuação deste trabalho, servindo como base para o entendimento das soluções propostas. No Capítulo 4 são apresentados os principais trabalhos relacionados a este, bem como a contextualização desses trabalhos relação a este. O Capítulo 5 apresenta a idéia da Infra-

Estrutura, o seu planejamento e as questões que ela soluciona. No Capítulo 6 é apresentado o projeto dos componentes da Infra-Estrutura e de sua interação. Ainda apresenta as diretrizes para a aplicação da Infra-Estrutura e algumas variações que podem ser utilizadas. O Capítulo 7 traz os estudos de caso da aplicação da Infra-Estrutura, os resultados dos testes de desempenho realizados e a análise dos mesmos. O Capítulo 8 apresenta as conclusões e considerações finais sobre o trabalho realizado.

2 SISTEMAS DE INFORMAÇÃO

O objetivo deste capítulo é apresentar uma breve introdução à classificação de sistemas e, com base nela, contextualizar o tipo de sistema que este trabalho irá abranger. As classificações apresentadas a seguir foram construídas em um contexto de adaptação e manutenção de sistemas legados, porém, são genéricas o suficiente para permitirem a sua utilização mesmo sobre sistemas novos ou que ainda estão em construção. Sistemas legados são definidos em [11] como aplicações que ainda são utilizadas pelas empresas, porém, que não possuem mais suporte ou atualização por parte do seu fornecedor. Tal fator limita a possibilidade de introdução de novas funcionalidades ou manutenção de funcionalidades antigas. Utilizando o raciocínio inverso, sistemas não legados são os que ainda são mantidos por equipes de suporte, tendo sua manutenção e evolução garantidas.

Segundo [12], todos os sistemas de informação possuem três componentes: interfaces, aplicações e serviços de banco de dados. As arquiteturas de sistema de informação variam amplamente de um espectro estruturadas (isto é, modulares, capazes de serem decompostas em partes e módulos) até sem estrutura (isto é, não decomponíveis).

Durante a revisão bibliográfica sobre classificação de sistemas, duas abordagens bastante semelhantes foram encontradas. A proposta de [12] classifica os tipos de sistemas em três grupos: totalmente decomponíveis, semi-decomponíveis e monolíticos. A proposta de [11] refina a classe de semi-decomponíveis em duas classes: semi-decomponíveis de dados e semi-decomponíveis de programas. A classificação está descrita na Figura 3 e o detalhamento, apresentado nas próximas subseções.

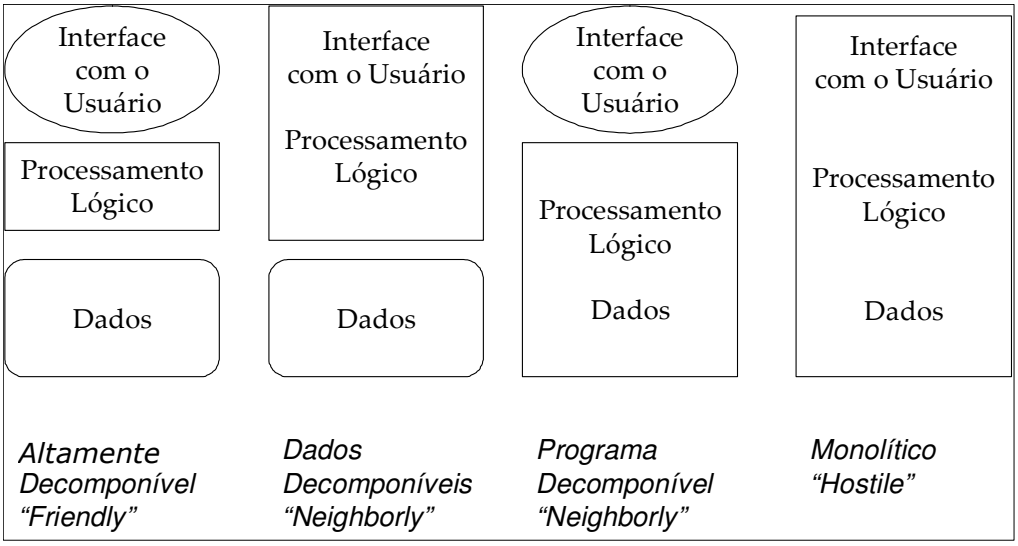

Figura 3 - Classificação de sistemas [11].

2.1 Sistema altamente decomponível

O melhor tipo de sistema para propósitos de migração são os altamente decomponíveis, no qual as interfaces de usuário, as aplicações e os serviços de banco de dados podem ser considerados como componentes distintos, com interfaces bem definidas. Ou seja, é uma coleção independente de módulos de aplicação que possuem suas próprias interfaces, cada qual trocando informações com os serviços de banco de dados, possuindo uma interface de usuário ou uma interface de sistema. É através dessas interfaces que esse tipo de sistema troca dados com um ou mais sistemas de informação estrangeiros. Ambas as interfaces (de usuário e de sistemas) precisam ser consideradas separadamente, já que se diferem significantemente em termos de tecnologia, projeto, requisitos de desempenho e impacto [12].

Uma definição parecida é apresentada por [11], onde esse tipo de sistema é tido como bem estruturado, contendo uma separação clara entre interface com o usuário, lógica de processamento da aplicação e serviços de acesso a dados. Ainda, permite o acesso direto às suas funcionalidades, independência de cada módulo sem dependência hierárquica e com interfaces bem definidas em todas as camadas.

Esse tipo de sistema, normalmente, é bastante amigável para ser transformado e suas interfaces são acessíveis para outras aplicações. Seus componentes podem ser acessados diretamente e a sua migração, realizada gradualmente.

2.2 Sistema Semi-Decomponível

Sistemas semi-decomponíveis são mais difíceis de serem migrados em contraste com os sistemas altamente decomponíveis. Para [11], é possível separar essa categoria em duas: sistemas decomponíveis de dados e sistemas decomponíveis de programas. As duas características principais dos sistemas decomponíveis de dados são: separação em camada de acesso a dados e camada de interface com o usuário/processamento de dados. Nessa categoria de sistema, os dados podem ser acessados diretamente, por exemplo, utilizando aplicações ou ferramentas remotas devido às interfaces bem definidas dos serviços de dados. Já a lógica de aplicação não pode ser acessada diretamente por outras aplicações, por estar altamente acoplada com a interface do usuário.

Sistemas decomponíveis de programas são aplicações semi-estruturadas semelhantes aos sistemas decomponíveis de dados, porém as camadas são: interface do usuário e as aplicações de processamento/dados. Portanto, as interfaces para acesso através de outros programas são bem definidas, porém os dados não podem ser acessados diretamente.

2.3 Sistema monolítico ou Não Decomponível

Esse tipo de sistema é visto como uma peça única, pois os seus componentes básicos não são divisíveis. Os usuários finais e o sistema de informações interagem diretamente um com o outro, aparentemente de maneira desestruturada de componentes ou módulos [12].

Para [11], nessa categoria, todo o sistema aparece como um único bloco, com uma única camada. Tais dados de aplicação podem ser acessados apenas através do terminal de usuário da aplicação, sendo indisponíveis para acesso por meio de outros softwares.

2.4 Resumo

Para que seja possível avaliar os resultados da aplicação de uma nova tecnologia em sistemas pré-existentes, deve-se, primeiramente, definir qual o tipo de sistema no qual o estudo será aplicado. Nesse sentido, a classificação de sistemas serve como um guia para que o estudo possa ser repetido em outros sistemas e seus resultados, comparados. Além disso, a classificação, segundo as categorias apresentadas, é bastante simples. Tais categorias foram concebidas com o foco na migração de sistemas legados através do encapsulamento ou reengenharia, porém, podem ser utilizadas para classificar qualquer tipo de sistema devido a sua generalidade. A abordagem proposta neste trabalho faz o uso de sistemas tanto legados como não legados, e a categoria na qual o sistema a ser integrado se encontra deve ser levada em consideração, principalmente porque limitações impostas por certos tipos de sistemas podem tornar as técnicas estudadas ineficientes.

O que se espera de um sistema, no escopo deste trabalho, são interfaces de aplicação bem definidas e acessíveis, permitindo o reuso da lógica do sistema através de uma integração com baixo acoplamento. Dessa forma, um sistema altamente decomponível pode ser aplicado no presente estudo, devido às suas interfaces de aplicação bem definidas, permitindo que seja possível utilizar diretamente a lógica do sistema.

Os sistemas decomponíveis de programas também podem ser enquadrados no escopo deste trabalho, pois o acesso direto aos dados não é requerido. Porém, os sistemas decomponíveis de dados não podem ser utilizados, uma vez que a lógica do sistema não pode ser acessada diretamente.

Para o escopo atual, o foco é a integração de sistemas, enquadrando, dessa forma, esses dois tipos de sistemas nos objetivos deste trabalho. Assim, será trabalhada apenas a análise dos sistemas preexistentes, mantendo o foco na solução das questões de interoperabilidade entre eles.

3 *WEB SERVICES* E *WS-NOTIFICATION*

Este capítulo tem como objetivo a introdução dos principais conceitos relacionados e da terminologia utilizada no restante do trabalho. Com relação a *Web Services*, esses apresentam a sua estrutura e principais elementos, além de listarem seus problemas e limitações na tarefa de integrar sistemas. É sugerido, também, um ciclo de vida que define um conjunto de atividades padrão de composição e utilização dos *Web Services* na integração de sistemas.

O Capítulo finaliza apresentando uma visão de EDA e os principais conceitos da *WS-Notification*, que é o elemento arquitetural central da integração a ser utilizado no restante do trabalho. O resumo final apresenta um paralelo de como a utilização de EDA pode solucionar os problemas encontrados na utilização de *Web Services* para a integração de sistemas.

3.1 Conceitos Básicos de *Web Services*

Com uma especificação não proprietária, *Web Service* é um esforço de padronização da W3C (*World Wide Web Consortium*), formada por um grupo de organizações que desenvolvem protocolos que auxiliam na sua evolução e asseguram sua viabilidade. Seu foco principal é a comunicação entre aplicativos, abstraindo as plataformas nas quais estes foram desenvolvidos.

Um *Web Service* ([1], [2]) é um aplicativo projetado para suportar interoperabilidade entre máquinas através de uma rede. Deve possuir uma interface apresentada em WSDL (*Web Services Description Language*), que descreve como outros sistemas podem interagir com ele através da utilização de mensagens SOAP (*Simple Object Access Protocol*). Essas mensagens são tipicamente transportadas usando HTTP (*Hypertext Transfer Protocol*) com uma formatação XML (*Extensible Markup Language*) e em conjunto com outras convenções relacionadas à *Web*.

A Figura 4 mostra a estrutura do padrão *Web Service*, os seus principais elementos e a forma com que esses elementos estão relacionados [13]. No restante dessa seção, serão detalhadas as principais tecnologias envolvidas e suas funções.

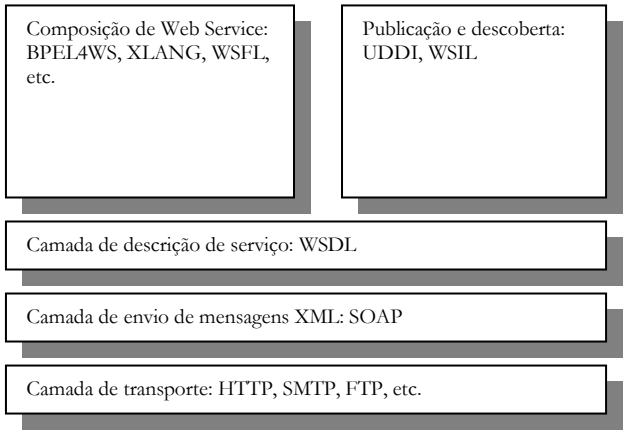

Figura 4 - Estrutura dos *Web Services* [13].

3.1.1 XML

XML é uma das tecnologias-chave para a construção e utilização de *Web Services* [14]. XML é um subconjunto da SGML (*Standard Generalized Markup Language*), um padrão complexo para descrever a estrutura do conteúdo de documentos. XML é uma linguagem para organização dos dados de documentos, e não apenas da sua apresentação, como HTML (*Hypertext Meta Language*). Por ser uma metalinguagem, permite que o usuário defina sua própria linguagem de marcadores personalizados ("*tags*") para cada tipo diferente de documento. XML possui um conjunto de padrões e tecnologias relacionados que facilitam e viabilizam a sua utilização. Entre as principais tecnologias e padrões, serão abordadas apenas as utilizadas neste trabalho: XML *Namespaces* e XML *Schema*.

Um XML *Namespace* é uma coleção de tipos de elementos e nomes de atributos identificados, unicamente, por um nome de duas partes: o URI (*Universal Resource Indicator*) e seu nome local. Um XML *Namespace* distingue entre tipos de elementos e atributos duplicados, então, é possível misturar duas ou mais descrições XML em um documento sem provocar conflitos ou ambigüidades [1].

O XML *Schema* fornece um *framework* necessário para criar documentos XML através da especificação da estrutura válida, restrições e tipos de dados para os vários tipos de elementos e atributos de um documento XML. Outra função-chave do XSD (XML *Schema*) é a de suportar herança. É possível criar novos esquemas através da derivação de esquemas já existentes. XML *Schema* também é bastante integrado com o XML *Namespace*, tornando tranqüila uma tarefa de criar elementos e atributos para um *Namespace* [1].

3.1.2 SOAP

SOAP [15] é um protocolo de comunicação baseado em XML utilizado para a execução de RPCs (*Remote Procedure Call*), chamadas remotas em ambientes distribuídos, e o envio de mensagens em nível lógico (mensagens SOAP). Por se utilizar de XML, SOAP pode ser transportado por qualquer protocolo de transporte de mais alto nível como HTTP, SMTP (*Simple Mail Transfer Protocol*), entre outros [1].

A estrutura de uma mensagem SOAP é composta por uma parte mais externa chamada de envelope. O envelope é o elemento do documento XML que representa o início e o final da mensagem e expressa o que está na mensagem, quem precisa tratar suas partes internas e se esse tratamento é opcional ou obrigatório [16].

Em seu núcleo, uma mensagem SOAP tem uma estrutura bastante simples: um elemento XML com dois elementos filho: cabeçalho (opcional) e o corpo da mensagem [17]. O corpo contém a real mensagem XML a ser transmitida. O cabeçalho SOAP, quando presente, tipicamente contém informação importante para a segurança, roteamento ou o tratamento correto da mensagem [1].

Em um nível de funcionalidade básica, é possível enviar mensagens do tipo SOAP. Supondo que o cliente de um serviço saiba sua localização e o formato no qual a mensagem deve ser enviada, ele pode simplesmente criar e enviar a mensagem para que o serviço a processe. Outra utilização do SOAP é na realização de RPC. Nesse caso, é necessário definir: (1) um protocolo RPC, o qual deve informar como os valores tipados podem ser transportados entre a representação de tipos de dados do SOAP e a representação de tipos de dados da aplicação, e (2) onde as várias partes do RPC são executadas (identidade do objeto, nome das operações e parâmetros) [17].

3.1.3 WSDL

Um documento WSDL descreve a interface de um *Web Service* e fornece para os usuários um ponto de contato que habilita a troca de determinadas mensagens [17] [18].

As informações fornecidas pelo WSDL podem ser divididas em dois grupos: as descrições de serviço em nível de aplicação, ou interface abstrata, e detalhes específicos dependentes de protocolos que usuários precisaram ter para acessar o serviço nos seus pontos finais [17].

A interface abstrata possui três componentes principais: o vocabulário, a mensagem e a interação. O acordo no vocabulário é a base de qualquer tipo de comunicação. WSDL usa sistemas de tipos externos para fornecer definições de tipos de dados para a troca de informação.

Embora WSDL possa suportar qualquer sistema de tipos, a maioria dos serviços usa XML *Schema* que pode ser importada de arquivos externos [17].

Para completar a descrição da interação cliente-serviço, são necessárias algumas informações, tais como o protocolo de comunicação deve ser usado (ex. SOAP sobre HTTP), como serão realizadas as interações individuais do serviço sobre este protocolo e onde a comunicação termina (o ponto de ligação) [17].

Para usuários e desenvolvedores, WSDL fornece uma descrição formalizada da interação cliente-serviço. Na fase de desenvolvimento de um *Web Service*, os desenvolvedores utilizam o WSDL como uma entrada para um gerador de *Proxy* que produz código cliente de acordo com os requisitos do serviço. WSDL também pode ser usado como entrada para um *Proxy* de invocação dinâmica, o qual pode, então, gerar a requisição correta do serviço em tempo de execução. O resultado, em ambos os casos, serve para liberar o usuário e o desenvolvedor da necessidade de lembrar ou entender todos os detalhes do acesso ao serviço [17].

3.1.4 Publicação, descoberta e composição

A publicação e descoberta de um *Web Service* têm como objetivo oferecer uma descrição de maneira que um cliente possa encontrá-lo. Para realizar essas tarefas, existem duas tecnologias básicas, o UDDI (*Universal Description, Discovery and Integration*) e o WSIL (*Web Services Inspection Language*). O UDDI é utilizado basicamente com um repositório público onde fornecedores cadastram seus serviços e os clientes podem buscá-lo [19]. Já o WSIL serve mais para indicar os serviços publicados em um determinado site [20].

A composição de *Web Services* é utilizada basicamente para prover serviços de uma maior granularidade, através de interações de negócio de longa duração. A composição de *Web Services* é tipicamente descrita por linguagens como BPEL4WS (*Business Process Execution Language for Web Services*), WSFL (*Web Services Flow Language*) ou XLANG (*Web Services for Business Process Design*), com o objetivo de estabelecer um modelo de processo baseado na concatenação ordenada de serviços WSDL e suas operações correspondentes [20].

3.2 Problemas Relacionados à Utilização de *Web Services*

Durante a pesquisa bibliográfica sobre *Web Services*, alguns problemas relacionados à utilização desses foram encontrados. Tais problemas estão intimamente ligados à origem dos *Web Services*, ou seja, à Internet. Ao mesmo tempo em que a Internet oferece todas as suas vantagens

ela também traz consigo uma série de desvantagens. Nessa seção, serão estudados alguns problemas para os quais o *NotificationBroker* traz alguma contribuição.

3.2.1 Gerência da configuração dos *Web Services*

Quando uma organização desenvolve aplicações utilizando *Web Services* de terceiros, possui menos controle sobre as partes da aplicação, já que esses serão controlados e mantidos por outras empresas. Devido à natureza anônima da Internet, essas empresas, por sua vez, podem desconhecer seus clientes e assim, a quebra da integração através da atualização de um *Web Service* pode acontecer. Essa quebra pode ser *explícita*, onde a interface WSDL e as mensagens SOAP mudam, ou *semântica*, onde a interface, sintaticamente, não muda, porém a natureza da informação produzida e fornecida é alterada. A quebra de integração explícita permite a detecção do erro, pois a aplicação que utiliza o serviço pára de funcionar, enquanto que a quebra semântica é mais difícil de ser detectada, podendo causar maiores problemas. O risco de alterações em *Web Services* deve ser gerenciado em tempo de execução através de políticas e ferramentas [21].

3.2.2 Limitações Derivadas da Internet

Web Services não deixam de ser aplicações *Web* e, portanto, possuem muitos dos problemas arquiteturais que os desenvolvedores de aplicações tradicionais da *Web* devem enfrentar. Enquanto que o protocolo HTTP permite que aplicações sejam acessadas a partir de qualquer máquina ele é, por natureza, sem estados (*stateless*) e não possui descrição implícita de atributos de qualidade. Um paradoxo dos *Web Services* é que HTTP não é um bom protocolo para automatizar processos de negócios, porém a sua onipresença pesa mais que suas limitações. Entre as principais limitações das aplicações *Web*, podem ser citadas [22]:

1. Confiança frágil na rede;
2. Gerenciamento da seção e do estado do cliente;
3. Segurança;
4. Performance/Escalabilidade;
5. Ausência de padrões de qualidade.

3.2.3 Descoberta Dinâmica de *Web Services*

Encontrar um *Web Service* é uma outra tarefa dos desenvolvedores da aplicação cliente e que, normalmente, utilizam para esse fim o UDDI. Isso é feito usualmente em tempo de desenvolvimento. Algumas aplicações, porém, são projetadas para encontrar os *Web Services* em tempo de execução. Nesse caso, é necessário um mecanismo para que o *Web Service* possa ser localizado, o qual pode consultar um serviço UDDI, para a procura de um serviço compatível com as suas necessidades, por exemplo [22].

Ainda sobre [22] esse tipo de sofisticação, não é usual devido à complexidade com que é realizada, sendo perceptível a necessidade de um sistema/mecanismo que facilite essa tarefa.

3.2.4 Projeto de Interfaces

Desenvolvedores de *Web Services* devem projetar interfaces com poucos métodos, os quais possuem assinaturas grandes e mais complexas [8]. Isso porque a invocação de *Web Services* é algo que possui um custo computacional maior que a invocação de componentes normais, por exemplo. Assim, utilizando assinaturas grandes, é possível melhorar o desempenho das aplicações e minimizar o acoplamento entre as partes.

Embora o uso eficiente dos *Web Services* talvez requeira uma estrutura de dados complexa, também é uma boa prática tentar manter essa estrutura o mais genérica possível, devendo ser minimizada qualquer codificação especial ou customizada e usados somente os tipos de dados fundamentais, tornando o *Web Service* menos acoplado às aplicações clientes e favorecendo o reuso do serviço [8].

3.2.5 Incompatibilidades de Tecnologias

Teoricamente, as ferramentas e *frameworks* já existentes (por exemplo, fornecidas pela Microsoft, IBM ou Sun) podem ser todas usadas para criar *Web Services* que serão descobertos e invocados por aplicações construídas por ferramentas de outros fornecedores. Na prática, é possível encontrar algumas incompatibilidades entre as diversas implementações do SOAP de alguns fornecedores. Apesar de incomuns, o fato é que elas existem e a verificação de compatibilidades deve ser adicionada aos testes normais aos quais uma aplicação deve ser submetida [22].

3.3 Ciclo de Vida para o Desenvolvimento Baseado em *Web Services*

Para a construção de um sistema que resolva os problemas encontrados, percebe-se a necessidade da definição de um cliclo de atividades para a utilização de *Web Services* em integrações de sistemas.

Em [23] é apresentado um ciclo de vida composto por atividades padrões para a utilização de *Web Services*. Esse ciclo de vida é utilizado como auxílio na construção de um sistema para a solução dos problemas mapeados. Esse ciclo de vida é ilustrado na Figura 5. As atividades estão detalhadas abaixo:

- Encapsulamento de serviços nativos: assegurar que um serviço nativo ou proprietário (ex. sistema legado) possa ser invocado por outro *Web Service*, independente do modelo de dados no qual ele está fundamentado, do formato das mensagens e do protocolo de interação;
- Publicação e descoberta de serviço: gerar uma descrição do serviço e publicá-la para que possa ser encontrada;
- Estabelecimento de acordos de terceirização (*outsourcing*): negociar, estabelecer e reforçar obrigações entre parceiros, ou seja, contratos de utilização de *Web Services* de terceiros. Em outras palavras, existe uma ligação bi-lateral entre cliente e servidor.
- Montagem de serviços compostos: identificar os serviços que irão participar de uma composição, especificando suas interações em um nível de abstração alto, e derivar descrições externas e acordos em nível de serviço para os serviços compostos resultantes;
- Execução de serviços: cumprir as especificações dos serviços de acordo com os modelos de execução que satisfazem a determinadas restrições práticas (ex. eficiência, disponibilidade);
- Monitoração da execução de serviços: supervisionar a execução dos serviços (ex. documentar as invocações do serviço, mudanças de estado e trocas de mensagens) de maneira a detectar a violação de contratos, medir a performance e prever exceções;
- Desenvolvimento de serviços: adaptar serviços para acomodar mudanças organizacionais, adiantar-se a oportunidades tecnológicas ou monitorar informações do cliente.

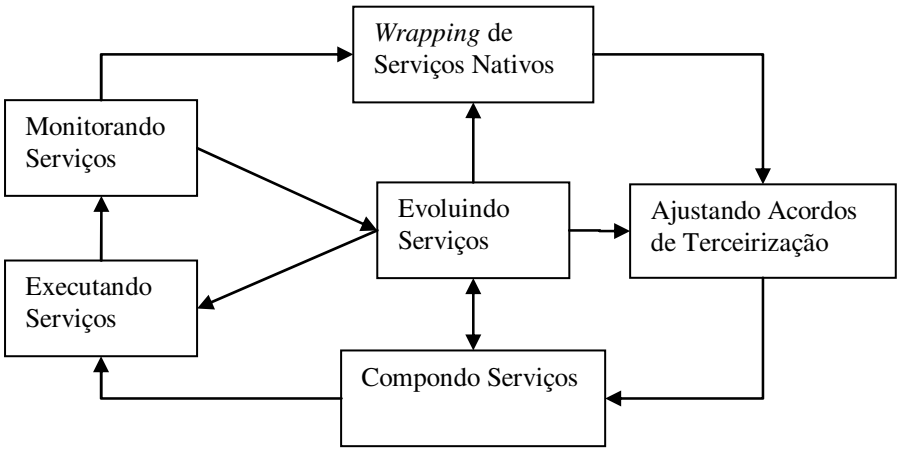

Figura 5 - Ciclo de vida de um serviço composto [23].

3.4 Utilizando EDA para a Integração de Aplicações

Uma EDA funciona basicamente através de notificações, ou seja, troca de mensagens entre os sistemas participantes. Cada vez que um evento é gerado, uma mensagem é enviada aos sistemas colaboradores que previamente solicitaram serem avisados de um determinado tipo de

evento. Ambos SOA e EDA habilitam a EAI (*Enterprise Application Integration*). A EAI pode ser vista como um passo adiante na evolução das aplicações de integração (*middlewares*), pois estendem as capacidades desses para superar a integração de aplicações já existentes, em oposição ao desenvolvimento de uma nova lógica de aplicação [3].

Uma combinação desses estilos arquiteturais permite que mensagens sejam trocadas através de serviços disponibilizados pelos sistemas de uma determinada arquitetura. A seguir, será apresentada a *WS-Notification*, que habilita a utilização de arquiteturas dirigidas a eventos com os *Web Services*.

3.4.1 WS-Notification

A *WS-Notification* [24] é uma família de especificações que define uma abordagem padrão para a utilização de notificações nos *Web Services*, usando o padrão *Publisher-Subscriber*. Seu funcionamento se dá, basicamente, através da introdução de um *Broker*, chamado, aqui, de *NotificationBroker*, que recebe mensagens e as repassa para uma lista de assinantes.

O restante da seção descreve os principais componentes dessa especificação necessários à compreensão desse trabalho. Mais detalhes podem ser encontrados em [24].

1. *Situation* (Situação): Ocorrência ou modificação em um ambiente que é do interesse de outros sistemas. Uma situação gera, tipicamente, uma ou mais *NotificationMessages*.
2. *NotificationMessage* (Mensagem de notificação): Descrição de uma situação. Tipicamente, para cada situação será gerada uma *NotificationMessage*, porém, é possível que uma situação gere mais de uma *NotificationMessage* ou mesmo, que diversas situações gerem a mesma *NotificationMessage*.
3. *Notification* (Notificação): Ato de transferir uma *NotificationMessage* para as partes interessadas.
4. *Publisher* (Publicador): Entidade que gera *NotificationMessages*, baseada em mudanças do seu estado interno. Não é, necessariamente, um *Web Service*, porém, se desejar trocar mensagens associado a uma interface *NotificationProducer*, deverá implementar um *Web Service*. Se ele não estiver associado a um *NotificationProducer*, então não precisa suportar mensagens de criação de assinaturas e ter conhecimento sobre os *NotificationConsumer*, que são seus assinantes, pois um *NotificationBroker* realiza essa tarefa.
5. *NotificationProducer* (Produtor de Notificações): Distribui as *NotificationMessages* para os seus respectivos assinantes. Pode ser um *Publisher*, isto é, criar as *NotificationMessages* ele próprio, ou um *NotificationBroker*, distribuindo *NotificationMessages* que foram produzidas por uma entidade *Publisher* à parte.

6. *Topic* (Tópico): Um tópico define um tipo específico de *Notification* e está relacionado ao esquema do *NotificationMessage*. Cada *NotificationMessage* deve estar associado a um *Topic*.
7. *NotificationConsumer* (Consumidor de notificações): *Web Service* que recebe as *NotificationMessages* de um *NotificationProducer*. Pode ser implementado para receber mensagens *Notify* genéricas, ou habilitado para processar um ou mais tipos de *NotificationMessage* específicos do domínio.
8. *Subscription* (Assinatura): representa o relacionamento entre um *NotificationConsumer*, *NotificationProducer*, *Topic* e outros elementos opcionais, tais como expressões de filtragem, políticas e informação do contexto. Ele é criado quando um assinante manda uma mensagem de requisição de assinatura para um *NotificationProducer*.
9. *Subscriber* (Assinante): é uma entidade (normalmente um *Web Service*) que atua como um requerente de serviços, enviando uma mensagem de requisição para um *NotificationProducer*. Pode ser uma entidade diferente de *NotificationConsumer* que, de fato, vai receber as *NotificationMessages*.
10. *NotificationBroker* (Mediador de notificações): *Web Service* intermediário que desacopla os *NotificationConsumers* de seus *Publishers*. Implementa as interfaces *NotificationProducer* e a *NotificationConsumer*. Por ser um *Broker,* pode fornecer funções mais avançadas do que as de um *NotificationProducer*. Ele pode:
    10.1. Assumir o papel de *NotificationProducer* (distribuir as *NotificationMessages*) e gerenciar as assinaturas.
    10.2. Reduzir o número de conexões e referências entre serviços, solucionando problemas que podem ocorrer, caso existam muitos *Publishers* e muitos *NotificationConsumers*.
    10.3. Funcionar como um localizador de serviços. Potenciais *Publishers* e *Subscribers* podem procurar uns aos outros utilizando um *NotificationBroker* comum.
    10.4. Prover *Notifications* anônimas, de tal forma que *Publishers* e *NotificationConsumers* não tenham a necessidade de conhecer a identidade uns dos outros.
    10.5. Realizar funções complexas como, por exemplo, transformar internamente o conteúdo das *NotificationMessages*.

A Figura 6 ilustra o relacionamento dos principais elementos da *WS-Notification*.

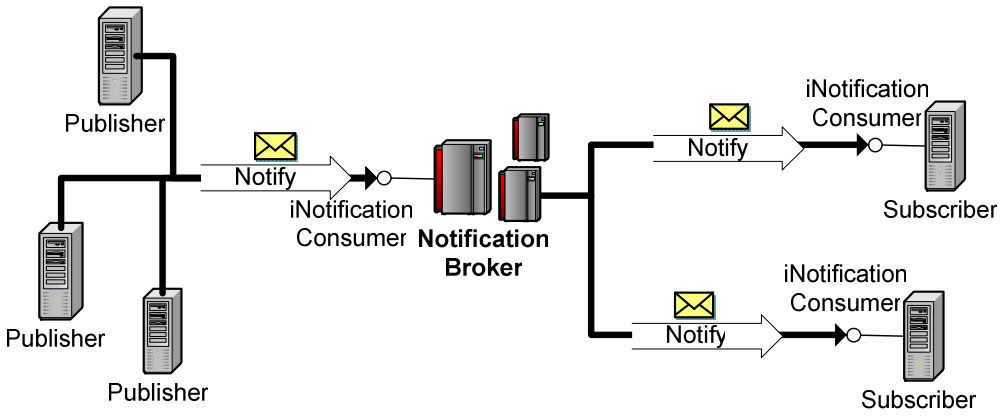

Figura 6 - Representação dos principais elementos do *WS-Notification*.

A padronização *WS-Notification* define, então, como as interfaces de *NotificationBrokers*, *NotificationConsumers* e *NotificationProducers* devem ser implementadas e os documentos que as descrevem definem elementos que são ou não obrigatórios.

A *WS-Notification* permite que um *NotificationProducer* envie uma *NotificationMessage* para um *NotificationConsumer* de duas formas diferentes:

1. Uma *NotificationMessage* de aviso, isto é, o conteúdo específico da aplicação.
2. Os dados do *NotificationMessage* usando uma mensagem *Notify*.

Uma mensagem *Notify* permite que o *NotificationProducer* forneça material adicional às informações definidas no *WS-Notification* (tais como o *Topic*), além do conteúdo do *NotificationMessage* específico da aplicação. Ele também permite que o *NotificationConsumer* possa receber uma grande quantidade de *NotificationMessages* sem ter que declarar o suporte a elas no seu WSDL *portType*. Essa forma de notificação permite que um lote de múltiplas *NotificationMessages* seja entregue utilizando uma única mensagem.

Quando requisitada a criação de uma assinatura em nome de um *NotificationConsumer*, o assinante precisa garantir que o *NotificationConsumer* encontra-se habilitado para tratar do tipo de notificação requisitada para aquela assinatura.

Para esse trabalho, por uma questão de simplificação, serão utilizados apenas notificações com mensagens do tipo *Notify*.

3.5 Resumo

Web Service refere-se a um conjunto de tecnologias emergentes que permitem expor serviços de uma forma transparente a clientes. Através dos *Web Services*, empresas podem encapsular os processos de negócios existentes, publicá-los como serviços, procurar por outros serviços e trocar informações com parceiros.

A utilização de *Web Services* traz diversos benefícios quando comparada às outras técnicas de integração de sistemas, principalmente pela facilidade de utilização e por estarem fundamentados em tecnologias amplamente difundidas. Atualmente, porém, alguns desafios podem ser identificados na adoção dessa tecnologia. A *WS-Notification*, como visto anteriormente, representa uma especificação para trabalhar com *Web Services* em uma arquitetura dirigida a eventos. Ela apresenta todos os elementos necessários para a construção de um *Broker* de notificações (*NotificationBroker*) e descreve como esses devem interoperar.

É possível relacionar o *NotificationBroker* com a solução parcial ou total dos problemas de projeto de *Web Services* apresentados:

1. Para a gerência da configuração dos *Web Services*, é possível perceber que o *NotificationBroker* pode, através de filtros, alterar o conteúdo de uma mensagem para que essa seja compatível com uma nova versão, mantendo ao mesmo tempo um controle centralizado de todas as versões disponíveis de um determinado serviço;
2. Algumas limitações importantes derivadas da Internet são possíveis de serem contornadas. Para a questão de performance, é possível realizar um balanceamento de carga no *NotificationBroker*, o que ajuda a amenizar este problema. Para a confiança frágil na rede, é possível implementar o *NotificationBroker* com um mecanismo que garanta a entrega da notificação, considerando uma infra-estrutura de hardware para a replicação de dados. Para a segurança, tem-se um controle de acessos a serviços através da centralização das chamadas no *NotificationBroker*. Quanto aos padrões de qualidade, esses podem ser monitorados no *NotificationBroker*, podendo ser utilizados para classificar os serviços cadastrados junto ao servidor;
3. A descoberta dinâmica de serviços torna-se automática, pois as mensagens são distribuídas diretamente aos assinantes e a sua localização é gerenciada pelo *NotificationBroker*, liberando o cliente dessa tarefa;
4. O projeto de interfaces é facilitado, pois se utiliza o padrão da *WS-Notification*, e a incompatibilidade de tecnologias é resolvida no *NotificationBroker*, ou seja, em um único ponto, independente do número de clientes e servidores.

É importente observar porém, que a utilização *NotificationBroker* do também implica em algumas conseqüências negativas para o sistema.

4 TRABALHOS RELACIONADOS

A integração de sistemas não é algo novo. A utilização de *Web Services* já possui diversos guias para a sua implementação. Este Capítulo busca os principais trabalhos relacionados a esse tema e que trazem alguma contribuição para solucionar alguns dos problemas já encontrados, além de posicionar o presente trabalho em relação a outros na mesma área, bem como posicionar os *Web Services* em relação a outras tecnologias.

4.1 Criação de *Web Services* Dirigida a Negócios

Para [25], as principais deficiências encontradas, atualmente, na criação de *Web Services* são:

1. As atuais arquiteturas cliente/servidor possuem alto acoplamento;
2. Os modelos de negócio propostos são complicados;
3. Ausência de um mapeamento entre os modelos de negócio e de *Web Services*;
4. Insuficiência de diretrizes para a criação de *Web Services*;

O autor propõe então uma metodologia para a criação de *Web Services*. A idéia central é criar um modelo dirigido a negócios através da separação em três camadas: espaço de negócio, espaço de serviço e espaço de computação. Além disso, cada camada é classificada de acordo com seu domínio arquitetural e semântico. O modelo fornece um mecanismo para realizar o mapeamento dos sistemas de negócios para os sistemas de serviços e, então, para os sistemas de computação, através de um processo consistente e um método comum de representação.

Primeiramente, será apresentado o modelo BSC (*Business-Service-Computing*) e, após, a metodologia proposta.

4.1.1 O Modelo BSC

Como mencionado anteriormente, o modelo BSC é dividido em três espaços, representados na Figura 7. Cada espaço consiste de um conjunto de instâncias que são chamadas de sistemas.

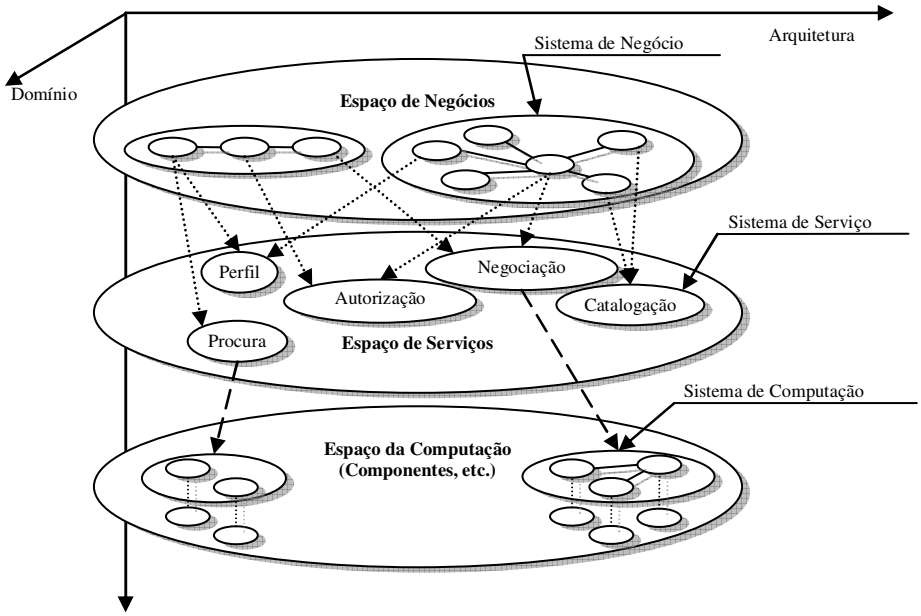

Figura 7 - Descrição do modelo [25].

Os serviços são vistos como um intermediário entre o espaço de negócio e o espaço de computação. O espaço de serviço encapsula os espaços da computação subjacentes, tal como os modelos de computação, modelos de interface e tecnologias de implementação. Dessa maneira, os sistemas de serviço podem trabalhar juntos em plataformas de computação diferentes. Por outro lado, sistemas de serviço podem ser reusados por múltiplos sistemas de negócio e a criação e evolução de sistemas complexos de negócios tornam-se ágeis e com um custo razoável.

Baseado no modelo BSC, os mapeamentos entre as camadas propostas são:

1. O mapeamento do espaço de negócio para o espaço de serviços indica que um sistema de negócios pode ser decomposto em um conjunto de serviços e descrito pela colaboração entre eles. O mapeamento inverso entre o espaço de serviços e o espaço de negócios indica que o serviço é reusado por vários sistemas de negócios.
2. O mapeamento do espaço de serviços para o espaço da computação indica que um serviço é implementado e executado por um conjunto de sistemas de computação. O mapeamento inverso do espaço de computação para o espaço de serviço indica que um sistema de computação é usado por um ou mais sistemas de serviços.

Para identificar o mapeamento entre os espaços BSC descritos na Figura 7, esses espaços são explorados em duas dimensões, a de arquitetura e a de domínio, as quais refletem a noção de sintaxe e semântica de cada modelo.

A noção de arquitetura de software define a estrutura global dos sistemas em termos de componentes e seus conectores. A arquitetura *Broker* consiste de clientes (*Requesters*), intermediários (*Brokers*) e fornecedores (*Providers*). O intermediário atua entre os clientes e os

fornecedores. A arquitetura *Broker* desacopla as interações entre clientes e fornecedores, o que é compatível com a natureza descentralizada da *Web*. O autor argumenta que este padrão arquitetural é comum para as três camadas da arquitetura BSC.

Pode-se perceber uma diferença de conceitos em relação à utilização da definição entre [24] e [25]. O *NotificationBroker* é um *Broker* com o objetivo de apenas repassar mensagens entre publicadores e assinantes, podendo ser classificado no sistema de componentes. O *Broker* apresentado por [25] está presente nas três camadas do modelo BSC, sendo que na camada de componente ele pode ser mapeado para o *NotificationBroker*, porém na de Serviços e de Negócios, não é possível realizar esse mapeamento. Por isso, neste trabalho utilizaremos o nome Intermediador para o *Broker* representado ou na camada de Serviços ou de Negócios, e apenas *Broker* caso ele esteja na camada de computação.

Modelos de negócio representam a estrutura lógica de uma unidade de negócios. A Figura 8 apresenta seis padrões de estrutura de sistemas de negócios descrita em [25], onde é possível observar a presença do modelo de Intermediador. Para cada um desses padrões, existe uma série de passos que devem ser seguidos. Por exemplo, para o padrão *Leilão*, primeiramente, os interessados em fazer a compra devem aderir ao leilão. Na seqüência, o vendedor cataloga todos os interessados que, em seguida, dão seus lances para, então, ser realizada a seleção de quem será o comprador.

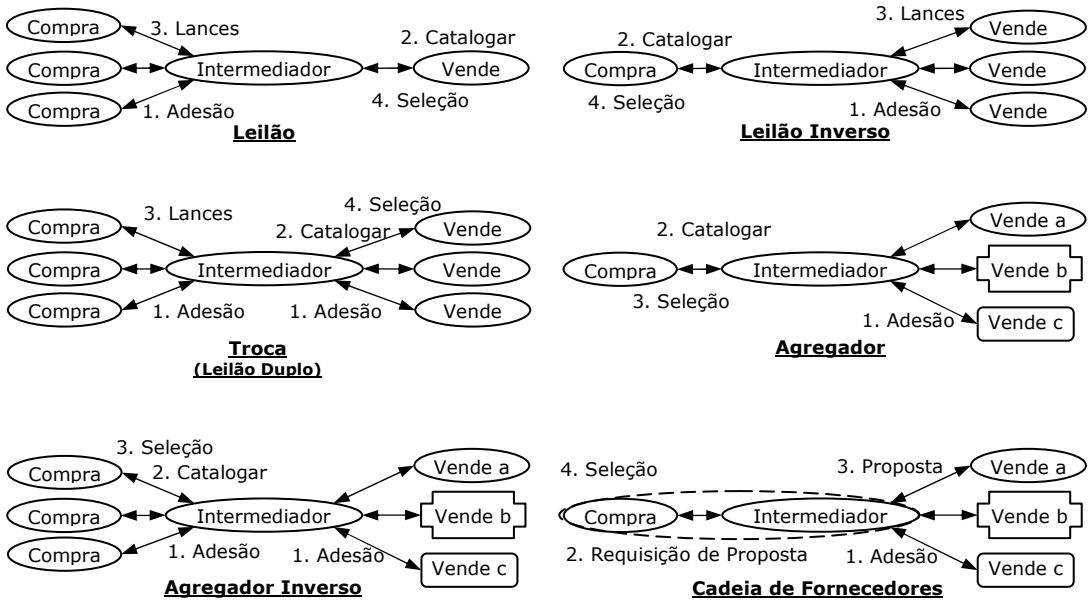

Figura 8 - Padrões estruturais da arquitetura *Broker* dos sistemas de negócios [25].

A dimensão *domínio* define uma área de aplicações de negócios similares. A Figura 9 apresenta um exemplo de aplicação com os domínios manufatura e vendas. O domínio é definido como uma unidade de um espaço semântico, o qual compartilha a mesma ontologia de negócios.

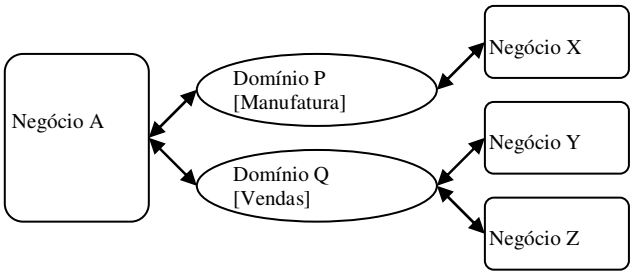

Figura 9 - Um exemplo de múltiplos domínios [25].

4.1.2 Metodologia de Criação de *Web Services* Dirigida a Negócios

Com base no modelo BSC, é apresentada uma metodologia para a criação de *Web Services* dirigida à negócios, composta, basicamente, de três etapas:

1. Modelagem de negócios: um sistema de negócios é criado através da análise do processo de negócio na empresa ou entre empresas. A ontologia de negócios para o seu domínio precisa ser criada através da classificação do vocabulário no domínio, no qual devem constar informações compartilhadas na empresa ou trocadas através de empresas.
2. Modelagem de serviços: a modelagem de um sistema de serviços pode ser feita através da decomposição dos sistemas de negócios, ou através da abstração do sistema de computação. Nas duas formas, o sistema de serviços precisa estar de acordo com os padrões dos *Web Services*, para que ele possa ser reusado e composto através da Internet.
3. Implementação e execução de serviços: implementar *Web Services* é particularmente direto se o sistema de serviço já estiver de acordo com o padrão dos *Web Services*. A interface do serviço é descrita com WSDL e o domínio é mapeado para um conjunto de XML Namespaces.

4.2 Integração de Sistemas com *Web Services*

Foram encontrados diversos trabalhos relacionados com a questão de integração de sistemas através de *Web Services*, cada qual apresentando uma abordagem diferente para a solução dos problemas encontrados.

Em [26] é apresentada uma série de padrões que dizem respeito à integração de sistemas com *Web Services*, além de um conjunto de tarefas para que esses padrões sejam aplicados. O foco do trabalho está no mapeamento entre os elementos da arquitetura e as tecnologias existentes no mercado. Para o trabalho com os elementos de implementação, é proposta uma técnica de mapeamento entre as interfaces dos componentes e as interfaces que devem ser providas para a arquitetura. Esse mapeamento entre os componentes pré-existentes e os *Web Services* não

considera se os elementos a serem integrados necessitam ou não de uma customização mais elaborada como, por exemplo, trabalhar com eventos.

Web Services são também aplicações distribuídas e, como tal, alguns dos padrões construídos para essa classe de aplicações também são válidos para *Web Services*. Padrões para sistemas distribuídos solucionam várias questões tal como segurança, configuração, paralelismo e sincronização. Schmidt et al [27] apresenta um conjunto de padrões de projeto para sistemas distribuídos que são utilizados, principalmente, na arquitetura CORBA (*Common Object Request Broker Architecture*) e podem ser utilizados diretamente com o elemento *Event Channel*, um tipo *Broker*.

Em [28] é apresentado um catálogo de padrões de projeto para serem utilizados com a tecnologia J2EE (*Java 2 Enterprise Edition*). Tais padrões são utilizados em diversas camadas das aplicações J2EE, inclusive o encapsulamento de sistemas legados e a utilização de sistemas que seguem o padrão *Publisher-Subscriber* [9]. Além disso, os autores também apresentam boas práticas relacionadas ao projeto e à arquitetura de aplicações na plataforma J2EE, com seu foco em quatro tecnologias Java: servlets, JSP (*JavaServer Pages*), EJB (*Enterprise JavaBeans*) components e JMS (*Java Message Service*).

Com relação a este trabalho, especificamente, o Capítulo 8 apresenta dois padrões importantes: *Service Activator* (Figura 10) e *Web Service Broker*. No primeiro, os autores abordam a utilização de chamadas assíncronas, apresentando uma solução com a tecnologia JMS. A idéia é permitir que um componente chame uma função remota de maneira assíncrona através desse sistema intermediário e forneça um ponto de retorno para que os dados requisitados sejam recebidos. O segundo padrão, *Web Service Broker*, apresenta uma estrutura e diretrizes para a criação de *Web Services* a partir de sistemas já existentes.

Para a integração, esses padrões apresentam uma limitação no que se refere ao lado cliente da solução, pois consideram que esse já está pronto para trabalhar de forma assíncrona.

Outras questões que o trabalho deixa em aberto são: (1) quais os requisitos necessários para que um sistema possa ser integrado ao modelo proposto pelos autores; (2) a inexistência de um tratamento para possíveis falhas devido a invocações remotas ou indisponibilidade de algum sistema participante da arquitetura; (3) o sistema proposto trabalha basicamente com chamadas assíncronas utilizando o modelo *pull*, ou seja, mantém o fluxo de eventos dentro do sistema consumidor-fornecedor, e acaba por não abordar o modelo *push*, ou seja, o envio de notificações do servidor para sistemas consumidores; (4) ele trabalha com a idéia de sistemas caixa-branca (*White-Box*) em que é possível realizar a alteração nas aplicações participantes da arquitetura, não

apresentando claramente como será o funcionamento dos padrões em sistemas caixa-preta (*Black-Box*), ou seja, quando não se tem acesso ao código fonte.

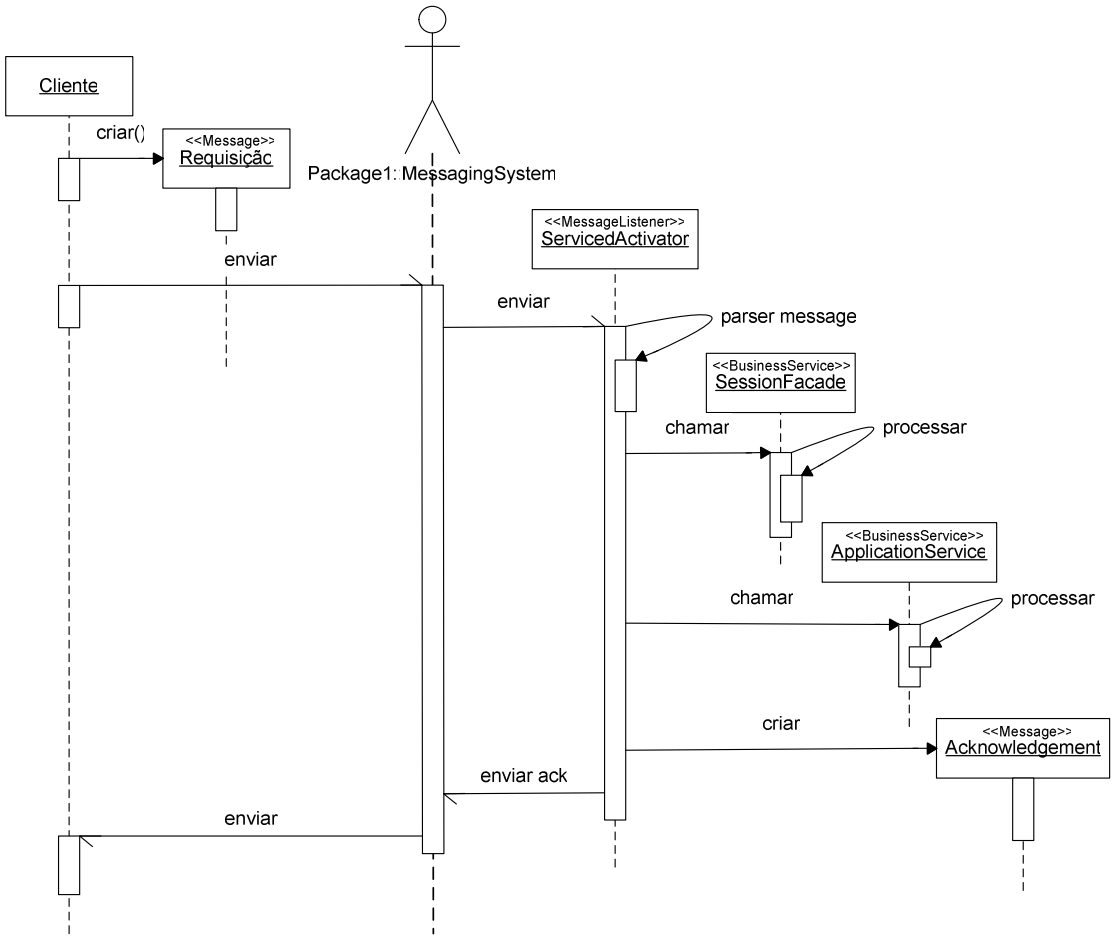

Figura 10 - Funcionamento do padrão *Service Activator* apresentado em [28].

4.3 *Web Services* e Componentes

É possível observar que existe uma evolução natural na tecnologia: novos conceitos e padronizações tendem a ser criados a partir da evolução de conceitos já aplicados. *Web Services* não são uma exceção, sendo possível encontrar, através da pesquisa bibliográfica, alguns dos seus precursores, como os componentes de software, os provedores de serviço de aplicações e aplicações de integração de sistemas [25].

Utilizando um foco em arquitetura e engenharia de software, é perceptível a aproximação entre *Web Services* e componentes [4]. Sobre essa ótica, *Web Services* são considerados como componentes de software reusáveis, os quais, utilizando uma interface, são publicados e podem ser utilizados através da Internet. A idéia é que, como componentes são precursores de *Web Services*, possuem propriedades em comum que podem ser tratadas de forma semelhante.

A seguir, serão apresentados conceitos fundamentais do desenvolvimento baseado em componentes e o relacionamento desses com os *Web Services*.

4.3.1 Conceitos de DBC

Brown [29] define componentes como um conjunto de funcionalidades reusáveis que podem ser distribuídos independentemente, implicando na possibilidade de outros componentes de acessarem essas funcionalidades oferecidas. Um princípio fundamental de DBC, para tal possibilidade, é que um componente deverá possuir uma especificação que descreve o que ele faz e como se comporta quando seus métodos são utilizados [29]. Distribuição independente significa que componentes são tipicamente inconscientes do contexto no qual são utilizados. Isso significa dizer que componentes não podem ser desenvolvidos com dependências incrustadas em outro componente, ou com a necessidade da presença de recursos externos.

Um dos principais elementos de um componente é a sua Interface. Ela resume como um cliente pode interagir com um componente, porém esconde detalhes fundamentais da implementação, fornecendo toda a informação da qual um cliente pode depender, permitindo que ele não tenha consciência de qualquer tipo de implementação, nem mesmo se uma implementação existe de fato [29]. Uma maneira útil de se ver especificações de interface é como contratos entre um cliente de uma interface e de um fornecedor de uma implementação de uma interface. O contrato indica o que o cliente necessita fazer para usar a relação. Indica, também, o que um fornecedor tem que implementar para se adequar aos serviços prometidos pela interface [30]. Para o mapeamento entre *Web Services* e componentes, o elemento WSDL estudado anteriormente corresponde à interface de um *Web Service*.

A tecnologia de componentes de software suporta a construção de um estilo particular que inclui componentes, modelos de componentes, e *frameworks* de componentes. O modelo de componentes impõe restrições de projeto no desenvolvimento de componentes. Já o *framework* de componentes reforça essas restrições em adição ao fornecimento de um conjunto de funcionalidades úteis [31]. Em outras palavras, são o modelo e *framework* de componentes que definem o uso e a comunicação dos componentes. Dois componentes podem comunicar-se somente se compartilham um mecanismo para se encontrarem e trocarem dados [32]. Para os *Web Services*, estes componentes corresponderiam ao conjunto dos elementos apresentados anteriormente, ou seja, o SOAP para troca de dados, WSDL para a descrição de interface, UDDI e WSIL para a publicação e descoberta. Tais conceitos são utilizados como um limitador entre as tecnologias que podem interagir com um *Web Service*, ou seja, se uma aplicação quiser interoperar com um *Web Service*, deve seguir determinadas normas do modelo proposto pelos *Web Services*.

4.4 Resumo

É possível encontrar muitos trabalhos relacionados à tecnologia dos *Web Services*, porém, boa parte deles está focada em alguma tecnologia, sendo perceptível a carência por metodologias para a aplicação dos mesmos, como apresentado na Seção 4.2. O modelo e a metodologia desenvolvidos por [25] são coesos e abstratos, porém, a questão de "como" a definição dos serviços vai ser realizada a partir da camada de negócios e de componentes não é detalhada. O trabalho apresentado em [25] praticamente não considera os sistemas e componentes pré-existentes e não apresenta como os *Web Services* são ligados entre si, apenas se baseia nas ligações modeladas na camada de negócio. Esses pontos acabam por inviabilizar a aplicação da metodologia proposta da forma com que ela é apresentada. Em [25] também é apresentado o conceito de intermediador de negócios, que permite a um elemento mediador concentrar parte da lógica de intermediação entre clientes e servidores.

Nesse ponto, a utilização da padronização *WS-Notification* pode vir a complementar o sistema, sendo utilizada na parte de comunicação dos componentes e operacionalização do Intermediador de negócios a ser desenvolvido. O conceito de componentes de software também vem ao encontro a esses objetivos, visto a sua proximidade com *Web Services* e a maturidade em que se encontram.

5 INTEGRAÇÃO DE SI COM O MODELO *WS-NOTIFICATION*

A integração de sistemas, como já mencionado anteriormente, é uma atividade essencial para as organizações e pode ser realizada de diversas maneiras. Muitos são os padrões estruturais que podem ser utilizados em um sistema de negócios para realizar uma integração.

É possível, então, visualizar duas formas distintas de interação entre sistemas de informação:

1. De forma direta, em que o cliente e o servidor mantêm toda a lógica do negócio e a interação é realizada sem intermediadores (Figura 11.a);
2. De forma indireta, em que um Intermediador atua entre o cliente e o servidor, gerenciando as requisições, suas respostas e a lógica de negócio que é realizada entre os elementos (Figura 11.b).

A utilização de um Intermediador pode trazer diversas vantagens:

1. Controle sobre regras de negócio;
2. Balanceamento de carga;
3. Gerenciamento de fluxo de dados;
4. Gerenciamento de configuração;
5. Controle de permissões e de segurança;
6. Localização dinâmica de sistemas;
7. Transparência de comunicação.

O *NotificationBroker* é um elemento roteador de mensagens utilizado para integrar outros sistemas. A sua utilização torna a arquitetura mais flexível e permite que os sistemas participantes sejam mantidos ou substituídos de forma transparente para os demais sistemas, tanto clientes como servidores. Esse tipo de comunicação resolve alguns dos problemas atualmente encontrados na integração de sistemas utilizando-se *Web Services*, apresentados na Seção 3.2.

O *NotificationBroker* trabalha, basicamente, com troca de mensagens, porém nem todos os sistemas de informação estão prontos para interoperarem neste estilo, sendo necessário que eles sejam adaptados para a comunicação com troca de mensagens. Além disso, como já discutido anteriormente, o *NotificationBroker* não é um sistema Intermediador, pois trabalha em um nível operacional.

Para tanto, é necessário desenvolver componentes que complementem o *NotificationBroker*, ou seja, viabilizem a utilização do *NotificationBroker* de forma transparente para os sistemas, além de permitirem que lógica de negócio seja colocada também nessa nova camada.

Para a realização dessa tarefa, é necessário, primeiramente, identificar quais as questões que estão envolvidas na integração de sistemas com o *NotificationBroker* e, em seguida, deve-se identificar soluções para essas questões.

a) Requisição direta

b) Requisição com Intermediador

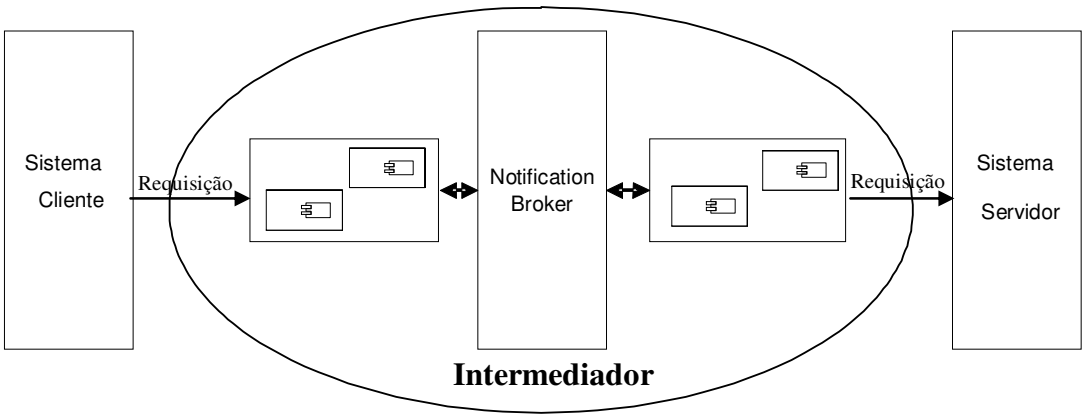

Figura 11 - Duas formas de interação entre cliente e servidor.

O presente capítulo discute os problemas envolvidos nesse tipo de interação e apresenta como um elemento intermediador pode ser construído para resolver tais problemas. Além disso, apresenta a idéia da Infra-Estrutura construída que representa o intermediador e permite a comunicação de sistemas de informação com o *NotificationBroker*.

5.1 Problemas identificados para integração

5.1.1 Compatibilidade dos sistemas com EDA

A primeira questão identificada é a incompatibilidade das interfaces disponibilizadas pelos sistemas a serem integrados ao estilo arquitetural. Essa é uma questão que aparece na integração de qualquer sistema, porém é agravada quando existe a necessidade de se adaptar sistemas para o padrão *Publisher/Subscriber* [9], como é o caso do *NotificationBroker*. Isso ocorre porque a troca de

mensagens no *NotificationBroker* é basicamente assíncrona, bastante diferente da forma síncrona, com o bloqueio de recursos por parte dos processos (Figura 12).

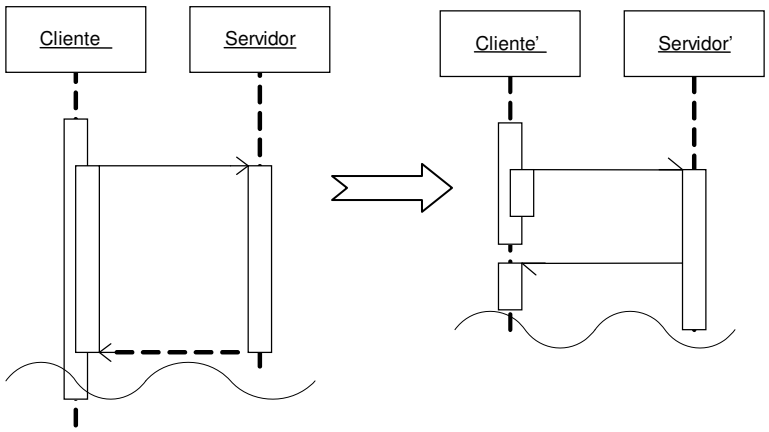

Figura 12 - Interação síncrona versus interação assíncrona.

Primeiramente, podemos identificar os sistemas a serem integrados em duas categorias: os já habilitados ao modelo de mensagens com interfaces prontas para o padrão *Publisher/Subscriber*, e os que se utilizam de chamadas síncronas, ou RPCs (*Remote Procedure Calls*), que correspondem à maioria dos sistemas existentes.

Na primeira categoria, a conversão/adaptação desses sistemas, para que funcionem com um *NotificationBroker*, restringe-se a aspectos sintáticos específicos à *WS-Notification*, que são apresentados na próxima seção. Na segunda categoria, é necessário, primeiramente, habilitar esses sistemas para que trabalhem com o modelo EDA. Para tanto, é necessário converter os clientes em *Publishers* e os servidores em *Subscribers*, ou seja, é realizada uma alteração no padrão de interação entre os sistemas clientes e servidores.

Em relação às interfaces providas pelos sistemas servidores, as quais são referidas pelos sistemas clientes, já devem ter sido convertidas para *Web Services* e estarem descritas em WSDL.

5.1.2 Encapsulamento e desencapsulamento de chamadas e mensagens

Como apresentado na Seção 3.4.1, a padronização *WS-Notification* utiliza mensagens do tipo *Notify* para a troca de informação. Assim, é necessário que as requisições dos clientes (*Publishers*) aos servidores (*Subscribers*) também estejam nesse formato. Para a categoria de sistemas que já trabalha com o padrão *Publisher/Subscriber*, é necessário o encapsulamento da mensagem para a padronização *WS-Notification*. A Figura 13 apresenta como exemplo o encapsulamento de uma mensagem JMS. Os novos elementos são os descritos na Seção 3.4.1 e o seu preenchimento é:

1. *Notify*: envelope externo utilizado por uma notificação;

2. *NotificationMessage*: é preenchido com todos os demais elementos, com o conteúdo da chamada e as informações necessárias para o roteamento da mensagem;
3. *Topic*: contém o tipo de mensagem gerada, de acordo com o método chamado, pelo sistema cliente. Para cada nova interface provida pela Infra-Estrutura, um novo *Topic* deverá ser cadastrado no *NotificationBroker* e os sistemas servidores que irão receber a notificação deverão ser cadastrados como assinantes desse novo *Topic*.
4. *Message Body*: elemento preenchido com os dados da chamada original mais um identificador único gerado para cada chamada.
5. *Identificador*: variável que identifica a chamada original. É e utilizado para que seja realizada a associação entre a chamada original e as respostas providas pelos servidores.

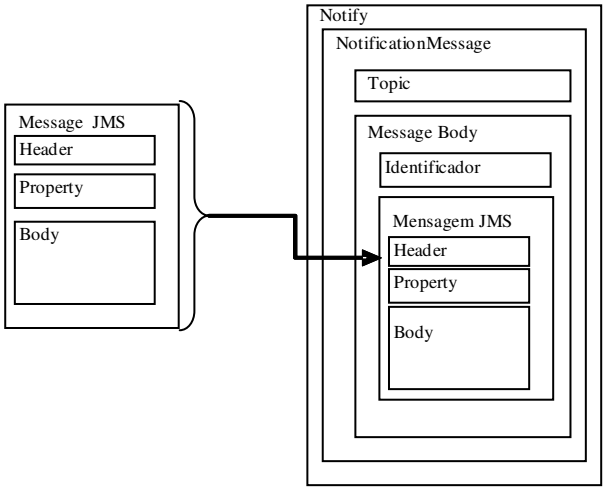

Figura 13 - Encapsulamento de uma mensagem JMS em uma mensagem *Notify*.

Caso a categoria seja a de sistemas que trabalham com chamadas RPC, deve ser realizado o encapsulamento da requisição, como é apresentado na Figura 14, armazenando todos os dados da chamada (HTTP *Header*, SOAP *Envelop* e SOAP *Header*, além do SOAP *Body*) para que possam ser repassados para o sistema assinante. Os elementos adicionados são os mesmos da Figura 13.

Da maneira inversa ao encapsulamento da mensagem, o desencapsulamento de uma mensagem deve ser realizado para permitir que o sistema servidor (agora visto como um *Subscriber*) receba a solicitação sem que necessite de alterações na sua interface. Ou seja, agora a mensagem está no formato *Notify* e deve ser transformada para o modelo original. A Figura 15 apresenta o desencapsulamento de uma chamada *Web Service*. O mesmo poderia, por exemplo, ser aplicado à mensagem JMS apresentada na Figura 13.

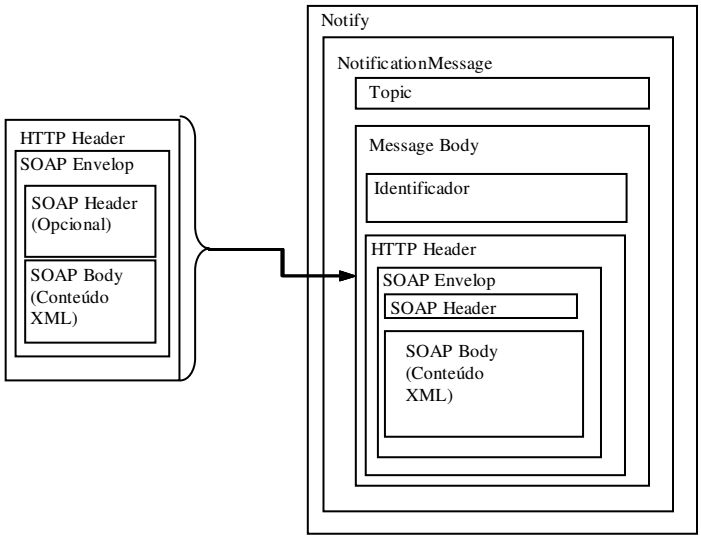

Figura 14 - Transformação de uma chamada RPC em uma mensagem *Notify*.

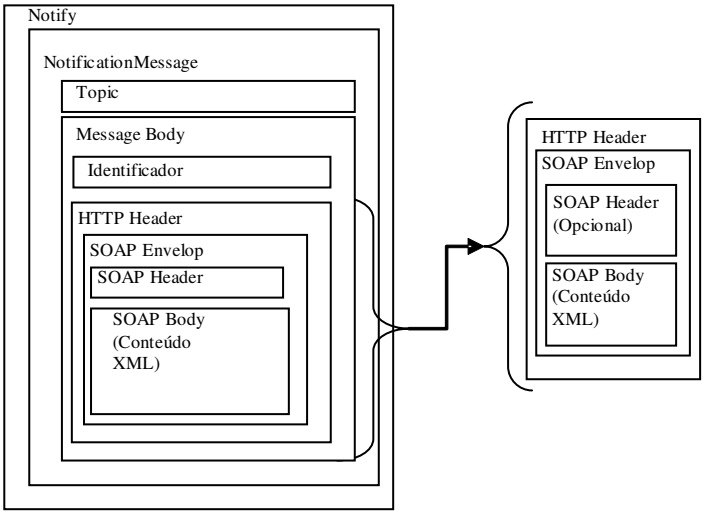

Figura 15 - Transformação de uma mensagem em uma chamada RPC.

Nessa atividade de desencapsulamento, devem ser mantidos todos os dados referentes à mensagem/chamada original, permitindo que o sistema assinante da mensagem receba as informações de forma transparente. Tal processo deve incluir além dos valores de variáveis de chamada a métodos, as informações como o HTTP *Header*, SOAP *Envelop* e SOAP *Header* (caso exista) que foram inicialmente enviadas pelo sistema cliente.

5.1.3 Suporte a respostas

Em uma EDA, sistemas publicadores geram eventos que são enviados para sistemas assinantes. Para sistemas que utilizam chamadas síncronas, é comum esperar uma resposta a sua solicitação. Como em EDA a comunicação é realizada basicamente através de notificações, essa resposta também será uma mensagem *Notify*. Assim, existe a necessidade de se criar algum ponto de retorno para essa mensagem de resposta.

É possível, para tal tarefa, utilizar uma "dual-interface" que é apresentada em [8]. Uma dual-interface é uma interface que realiza tanto uma solicitação a um sistema externo como recebe uma solicitação de um sistema com a resposta à solicitação inicial. Dessa forma, permite receber resposta de maneira assíncrona. Contudo, um dos requisitos para a utilização dessa solução é que o sistema que se está adaptando seja caixa branca, ou seja, o código fonte deve estar disponível para alteração. Para sistemas do tipo caixa preta (que não possuem seu código disponível para alteração), tal solução não é aplicável.

Neste trabalho, estão sendo considerados sistemas altamente decomponíveis ou semi-decomponíveis de programas, como apresentado no Capítulo 2, ou seja, basta que as interfaces da camada de aplicação estejam disponíveis para que o sistema possa ser reusado, não dependendo da disponibilidade do seu código fonte. Assim, se identifica um problema de retorno de mensagens para sistemas que trabalham com RPC e são do tipo caixa preta.

5.1.4 Extensão do *NotificationBroker* para suportar lógica de mediação do negócio

Para [25], intermediadores no nível de sistemas de negócio incorporam uma lógica específica a um padrão de negócio, tal como Leilão, Leilão Inverso, Troca, Agregador, Agregador Inverso e Cadeia de Fornecedores. Cada um desses padrões possui uma finalidade própria e pode ser trocado conforme a necessidade do sistema de negócios que se está a integrar.

Assim, é possível que um sistema mediador de negócios não apenas repasse mensagens entre *Publishers* e *Subscribers*, como o *NotificationBroker* o faz, mas, também, que contenha algum nível de conhecimento e poder de decisão para realizar a seleção e negociação de respostas. Essa extensão retira parte da lógica do cliente e/ou do servidor e a centraliza no elemento Intermediador, permitindo, assim, mais flexibilidade para evoluir a lógica de negociação entre clientes e servidores.

5.1.5 Referência a sistemas remotos

A integração de sistemas tem, necessariamente, como premissa, que um sistema conheça a localização do outro. A utilização de uma referência *hardcoded*[§] não permite um acesso transparente ao sistema parceiro, forçando, necessariamente, a uma nova adaptação do sistema por completo para cada alteração no sistema acessado, além de dificultar o desenvolvimento dos sistemas.

[§] Referência a um sistema remoto que não pode ser configurada, sendo possível apenas alterá-la recopilando todo o sistema.

Como se trata de uma arquitetura distribuída, a utilização de uma referência a um objeto local como ligação a um objeto remoto é extremamente útil [10]. A transparência de localização serve para facilitar a utilização de chamadas de objetos remotos de forma mais natural.

5.1.6 Tolerância a falhas de sistemas externos

Sistemas distribuídos possuem problemas de consistência que não são encontrados com freqüência em sistemas centralizados. Um sistema distribuído consiste em número variado de computadores conectados que estão sujeitos a falhas independentes em qualquer um de seus componentes como, por exemplo, ligação com a rede, sistema operacional ou aplicação individual [33].

É possível realizar uma implementação de uma *NotificationBroker* que seja tolerante a falhas, garantindo a entrega das notificações publicadas a seus assinantes, porém, isso não garante que os sistemas participantes não falhem ou que estejam sempre disponíveis. Se um cenário com o padrão de mediação Leilão em que muitos podem ser os sistemas envolvidos, por exemplo, tem um de seus elementos inacessível, não necessariamente o Leilão vai parar de funcionar, pois outros sistemas estarão disponíveis, o que seria suficiente.

Assim, a descentralização permite que partes do sistema falhem enquanto outras partes continuem funcionando, o que pode causar um comportamento anormal na execução de uma aplicação. Dessa forma, é necessária uma solução capaz de gerar resposta satisfatória mesmo quando uma parte dos sistemas externos falhe.

5.2 Proposta de uma Infra-Estrutura

A adaptação de sistemas em ambientes heterogêneos pode ser problemática e custosa. Para a solução das questões encontradas, é proposta a criação de uma Infra-Estrutura de integração, que permita guiar a adaptação de sistemas para integração dos mesmos via um *NotificationBroker*. Essa Infra-Estrutura deve ser genérica para permitir que, de maneira sistemática, os sistemas de informação possam ser adaptados de forma simples.

Além disso, é importante que parte da lógica de negociação, que coordena a comunicação entre clientes e servidores, seja implementada em um elemento central flexível que facilite o controle sobre as regras de negócio, permitindo que a construção de um intermediador de mais alto nível, como sugerido por [25], seja viabilizada.

A utilização de uma Infra-Estrutura serve como um guia para a adaptação de aplicações de forma sistêmica, permitindo a replicação da solução em vários domínios diferentes. Com base nas questões identificadas na Seção 5.1, um conjunto de componentes pôde ser identificado,

baseado no papel que cada um deles deve desempenhar para solucionar uma ou mais questões. A Figura 25 esboça os componentes que definem a Infra-Estrutura de integração proposta.

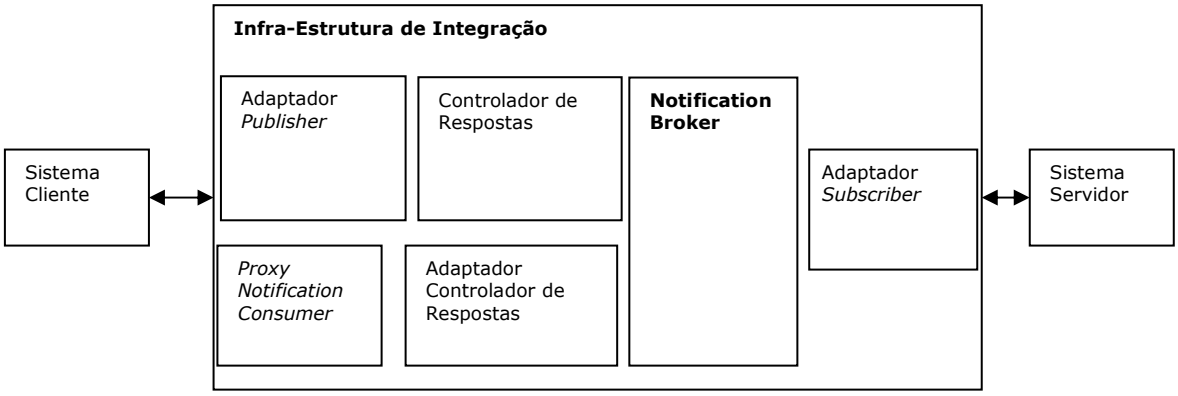

Figura 16 - Componentes da Infra-Estrutura de Integração.

A Tabela 1 apresenta o mapeamento entre as questões identificadas e os componentes da Infra-Estrutura. Cada componente pode atuar para resolver uma ou mais questões e a solução para cada questão pode estar distribuída em mais de um componente.

Com as questões resolvidas, supõe-se que integrações de sistemas com o *NotificationBroker* fiquem simplificadas e, por conseqüência, mais rápidas de serem realizadas. Espera-se dessa Infra-Estrutura que a sua implementação seja viável considerando requisitos não funcionais como desempenho.

A utilização de padrões de projeto e arquiteturais permite que problemas conhecidos da engenharia de software sejam solucionados de uma maneira eficiente, visto que esses já foram utilizados em outros sistemas com sucesso.

Tabela 1 - Problemas de adaptação vs. Componentes da Infra-Estrutura.

Questões \ **Componentes**	Adaptador *Publisher*	Controlador de Respostas	*Proxy NotificationConsumer*	Adaptador *Subscriber*	Adaptador Controlador de Respostas
Compatibilidade dos sistemas com EDA	✓	✓		✓	
Encapsulamento e desencapsulamento de chamadas e mensagens	✓			✓	✓
Suporte a respostas		✓			✓
Extensão do *NotificationBroker* para suportar lógica de mediação do negócio		✓			
Referência a sistemas remotos			✓		
Tolerância a falhas de sistemas externos		✓			

A Tabela 2 mostra os padrões de projeto/arquiteturais que foram empregados para o desenvolvimento dos componentes da Infra-Estrutura. O mapeamento foi realizado de acordo com as questões que cada componente soluciona, considerando as forças de cada padrão.

Tabela 2 - Padrões de Projeto/Arquiteturais vs. Componentes da Infra-Estrutura.

Padrões \ **Componentes**	Adaptador *Publisher*	Controlador de Respostas	*Proxy NotificationConsumer*	Adaptador *Subscriber*	Adaptador Controlador de Respostas
Adapter	✓			✓	✓
Façade	✓				
Blackboard		✓			
Proxy			✓		

6 INFRA-ESTRUTURA DE INTEGRAÇÃO

Este capítulo detalha os componentes da Infra-Estrutura de integração desenvolvida para resolver os problemas de conexão de sistemas de informação com o *NotificationBroker*.

Para cada componente, são discutidas as questões de integração abordadas, os padrões escolhidos para a sua implementação, bem como a estrutura e componentes resultantes. Além disso, este capítulo mostra como utilizar a Infra-Estrutura e discute algumas variações possíveis.

6.1 Componentes identificados para a construção da Infra-Estrutura

6.1.1 Adaptador *Publisher*

Esse componente tem por objetivo tratar de duas questões envolvidas na integração de sistemas com o *NotificationBroker*:

- Transformação do cliente para que ele trabalhe com o padrão *Publisher-Subscriber*, necessário quando este utiliza RPC (Seção 5.1.1);
- Encapsulamento de mensagens e chamadas em uma mensagem *Notify*, necessário às duas categorias (Seção 5.1.2).

Durante a revisão bibliográfica, foi estudado o padrão de projeto *Adapter* [10] o qual possui como objetivo converter as interfaces de um sistema em outras interfaces esperadas. Ou seja, é possível utilizar-se do padrão *Adapter* para converter sistemas para funcionarem tanto com troca de mensagens quanto para converter a mensagem para a padronização *Notify*. Sua utilização cabe à solução dessas questões de adaptação de interfaces, sendo, tal padrão, o ponto de acesso dos clientes à Infra-Estrutura.

Para permitir que a Infra-Estrutura seja o mais flexível possível, é importante garantir que a criação desse *Adapter* não seja onerosa para o projeto. Como a Infra-Estrutura possui outros elementos, é importante que o seu acesso seja o mais transparente possível para o cliente, sendo outra responsabilidade desse componente esconder tal complexidade.

O padrão *Façade* [10] tem por objetivo simplificar o acesso a um conjunto de classes ou *framework*, apresentando um ponto de acesso único e diminuindo a dependência entre elementos. Assim, o *Façade* torna-se interessante para ser aplicado junto com o *Adapter*, incrementando a flexibilidade e reusabilidade da Infra-Estrutura.

A Figura 17 apresenta a utilização desses padrões em conjunto para a realização de uma interface requerida pelo sistema cliente, permitindo acesso a outros elementos internos da Infra-Estrutura de integração de forma transparente. A Tabela 3 descreve os elementos da Figura 17.

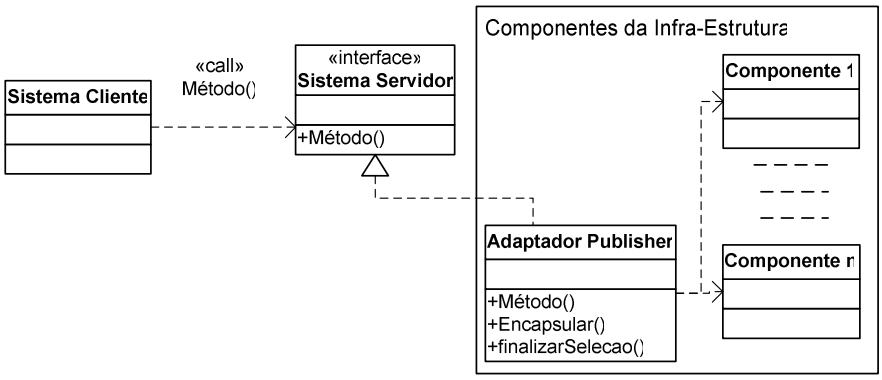

Figura 17 - Adaptador *Publisher*.

Tabela 3 - Descrição das classes participantes do Adaptador *Publisher*.

Elemento	Função
Sistema Cliente	Responsável por realizar a requisição.
<<interface>> Sistema Servidor	Provê a interface requerida pelo Sistema Cliente e que deverá ser adaptada.
Método()	Representa a assinatura do método do sistema servidor chamado pelo cliente.
Adaptador *Publisher*	Define a classe concreta a ser invocada pelo sistema cliente, provendo uma interface única para o conjunto de componentes da Infra-Estrutura.
Método()	Método concreto a ser chamado.
Encapsular()	Método utilizado para encapsular a mensagem para o formato *Notify*.
finalizarSelecao()	Método utilizado para finalizar a seleção de respostas. É ativado de acordo com um critério de parada escolhido para cada implementação da Infra-Estrutura.
Componente 1 .. n	Demais componentes internos da Infra-Estrutura, que devem ser invocados, detalhados no restante do Capítulo.

O Adaptador *Publisher* funciona recebendo as chamadas dos clientes e ativando os componentes internos responsáveis por trabalhar com cada requisição. Ele converte a mensagem para a padronização *Notify* através do seu método encapsular() e repassa para o componente interno responsável por tratá-la. O Adaptador *Publisher* também é responsável por coletar a resposta final selecionada e retorná-la ao sistema cliente. Seu funcionamento está no diagrama de seqüência da Figura 18. Os componentes Controlador de Respostas (Controle Mediação) e *Proxy NotificationConsumer*, bem como as mensagens enviadas a estes, serão detalhadas nas seções 6.1.4 e 6.1.2, respectivamente.

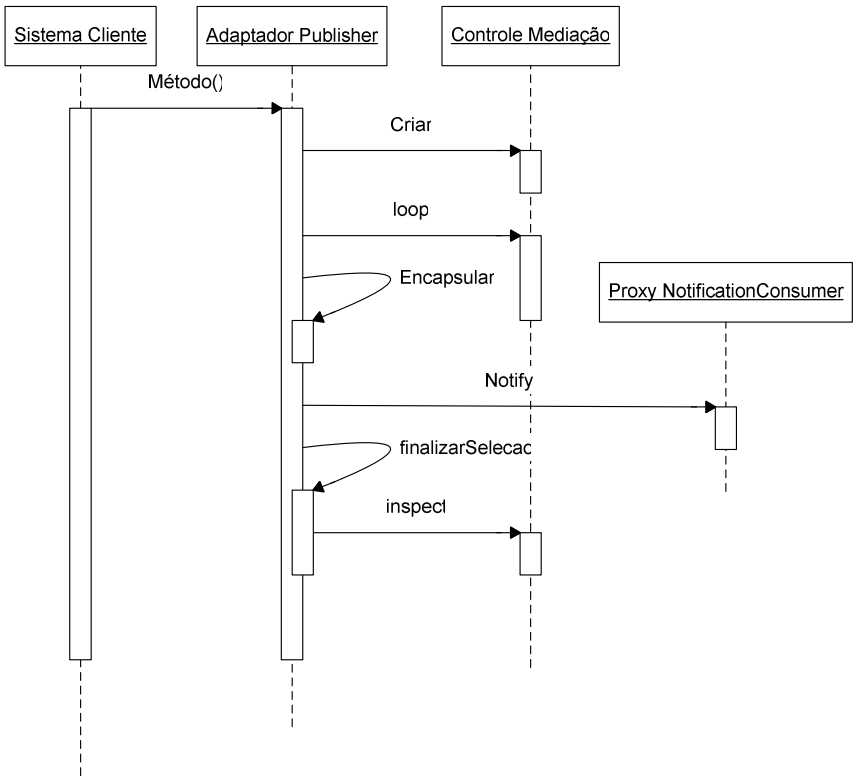

Figura 18 - Funcionamento do componente Adaptador *Publisher*.

6.1.2 *Proxy NotificationConsumer*

Esse componente tem por objetivo permitir uma referência local ao *NotificationBroker* que soluciona a questão apresentada na Seção 5.1.5, destacando a necessidade de transparência na invocação remota de sistemas.

O padrão *Proxy* [9] permite essa referência a sistemas remotos, habilitando que a localização seja alterada sem que o sistema tenha que ser modificado. Assim, esse componente auxilia a implementação da Infra-Estrutura, provendo transparência para a comunicação com o *NotificationBroker*.

Para tal, o *Proxy NotificationConsumer* também deve implementar a mesma interface *NotificationConsumer* já implementada pelo *NotificationBroker* (vista na Seção 3.4.1). O *Proxy NotificationConsumer* está apresentado na Figura 19, a qual é detalhada na Tabela 4.

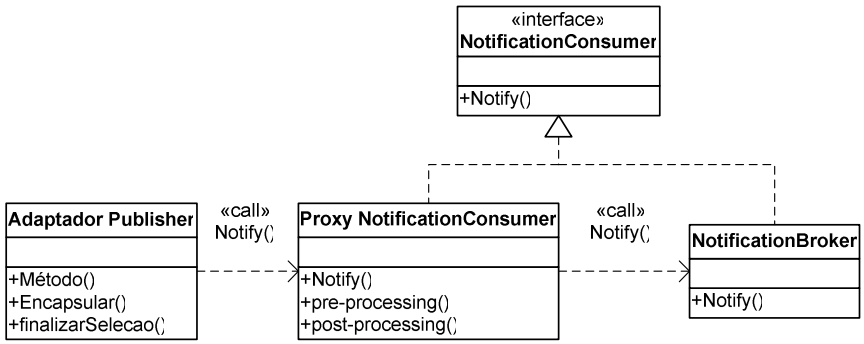

Figura 19 - *Proxy NotificationConsumer.*

Tabela 4 - Descrição das classes participantes do *Proxy NotificationConsumer.*

Elemento	Função
Adaptador Publisher	Elemento que solicita o envio da mensagem para o *NotificationBroker*.
Método()	Método chamado por um sistema cliente (conforme Tabela 3)
Encapsular()	Método utilizado para encapsular a mensagem para o formato *Notify*.
finalizarSelecao()	Método utilizado para finalizar a seleção de respostas. É ativado de acordo com um critério de parada determinado para cada implementação da Infra-Estrutura.
NotificationConsumer	Interface que será implementada pelo *Proxy NotificationConsumer* e pelo *NotificationBroker*.
Notify()	Assinatura do método utilizado para o envio de uma mensagem *Notify*.
Proxy NotificationConsumer	Esconde a real localização física do *NotificationBroker* repassando a este as mensagens *Notify* recebidas.
Notify()	Método utilizado para o envio de mensagens ao *NotificationBroker*.
pre-processing()	Método executado antes da invocação do método remoto, utilizado para ler as configurações que serão utilizadas na invocação.
post-processing()	Método executado após a invocação do método remoto, utilizado para liberar as estruturas de dados não mais utilizadas.
NotificationBroker	Sistema responsável por enviar a mensagem *Notify* para todos os assinantes registrados (descrito na Seção 3.4.1.)
Notify()	Método utilizado para o envio de mensagens *Notify*.

Para a Infra-Estrutura, o componente *Proxy NotificationConsumer* é utilizado em dois momentos:

a) No envio ao *NotificationBroker* da mensagem *Notify* originada de uma requisição de um cliente (Figura 20), representado na Infra-Estrutura pelo Adaptador *Publisher*;

b) No envio ao *NotificationBroker* da mensagem *Notify* originada da resposta de um servidor (Figura 21), representado na Infra-Estrutura pelo Adaptador *Subscriber*. Esse aspecto será mais bem detalhado na Seção 6.1.3.

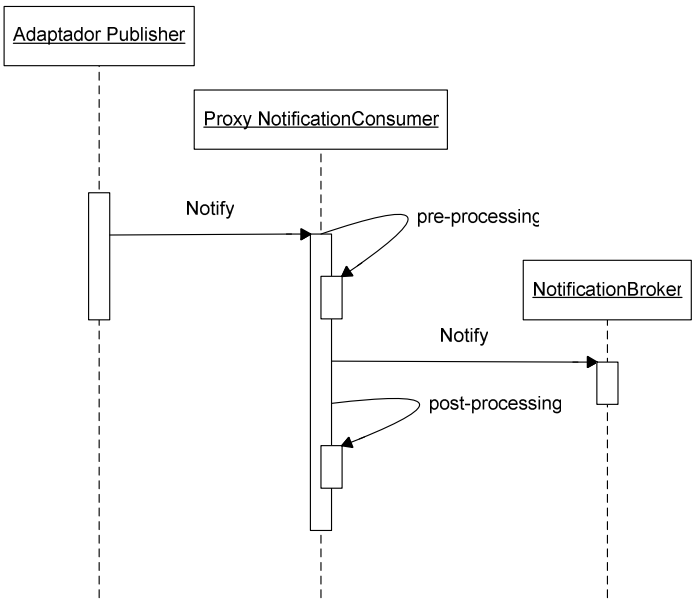

Figura 20 - Mensagens do Adaptador *Publisher* para o *Proxy NotificationConsumer*.

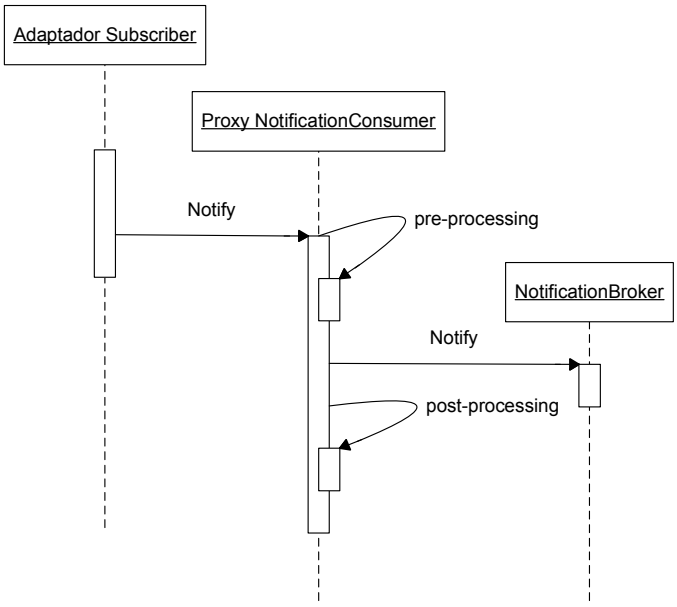

Figura 21 - Mensagens do Adaptador *Subscriber* para o *Proxy NotificationConsumer*.

6.1.3 Adaptador *Subscriber*

Esse componente tem por objetivo tratar de duas questões envolvidas na integração de sistemas com o *NotificationBroker*:

- Transformação do servidor em assinante (*Subscriber*), necessário quando este utiliza RPC (Seção 5.1.1);
- Desencapsulamento de mensagens e chamadas de uma mensagem *Notify* (Seção 5.1.2), de forma transformá-las novamente para seu formato original, para que o sistema servidor não perceba a diferença entre a chamada realizada pela Infra-Estrutura e o Sistema Cliente original.

Como na Seção 6.1.1, também o padrão *Adapter* é capaz de solucionar essa questão, porém, não é necessária a utilização em conjunto com o *Façade*, visto que se trata de apenas um ponto de acesso (um sistema servidor) para cada chamada. A Figura 22 apresenta o padrão *Adapter* aplicado para o desenvolvimento desse componente.

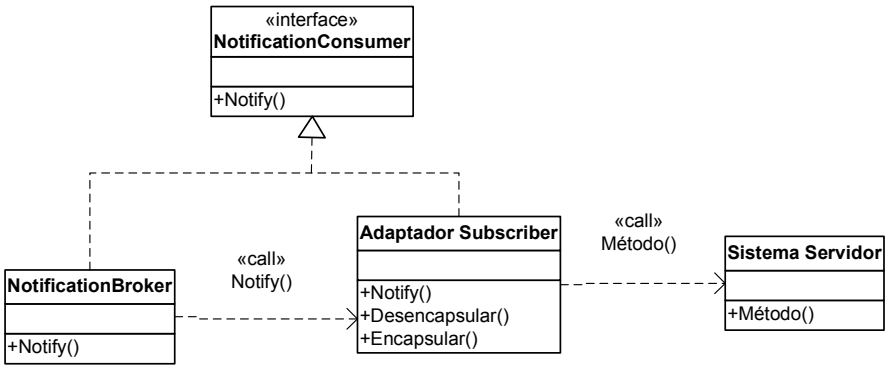

Figura 22 -Adaptador *Subscriber*.

Tabela 5 - Descrição das classes participantes do Adaptador *Subscriber*.

Elemento	Função
NotificationBroker	Sistema responsável por enviar uma mensagem *Notify* para todos os assinantes registrados (descrito na Seção 3.4.1.)
Notify()	Método concreto, utilizado para o envio de mensagens *Notify*.
NotificationConsumer	Interface que será implementada pelo Adaptador *Subscriber* e pelo *NotificationBroker*.
Notify()	Assinatura do método utilizado para o envio de uma mensagem *Notify*.
Adaptador *Subscriber*	Elemento responsável por invocar o método final do Sistema Servidor, convertendo uma mensagem *Notify* na Mensagem/Chamada original.
Notify()	Método que recebe a notificação, converte na mensagem/chamado original, envia a seu destinatário, retornando a resposta ao *NotificationBroker* após o encapsulamento.
Desencapsular()	Método responsável por desencapsular a mensagem *Notify* para a padronização esperada pelo Sistema Servidor
Encapsular()	Método utilizado para encapsular a mensagem de resposta, caso exista, para o formato *Notify*.
Sistema Servidor	Sistema que contém as operações necessárias para o Sistema Cliente
Método()	Método original que o Sistema Cliente deseja invocar.

O funcionamento desse componente está descrito na Figura 23. O Adaptador *Subscriber* recebe uma mensagem do *NotificationBroker* e o referido componente desencapsula a mensagem e realiza a chamada ao método original do sistema servidor.

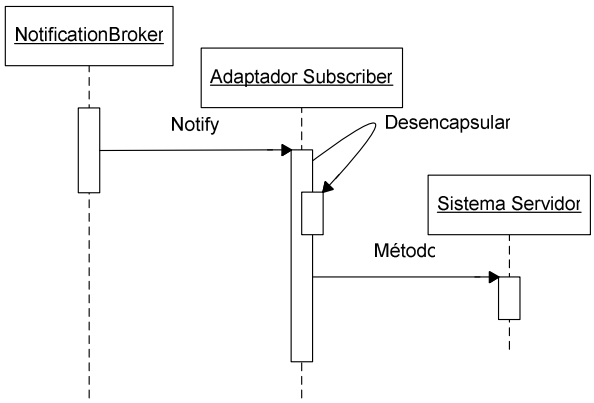

Figura 23 - Funcionamento do Adaptador *Subscriber* para o recebimento de mensagens.

É possível que o Sistema Servidor forneça uma resposta à requisição. Nesse caso, o componente encapsula a resposta em uma nova mensagem *Notify* e coloca o mesmo identificador da mensagem originalmente recebida. O *Topic* dessa nova mensagem deve estar de acordo com a assinatura do Adaptador Controlador de Respostas (componente visto a seguir) para que este receba a notificação.

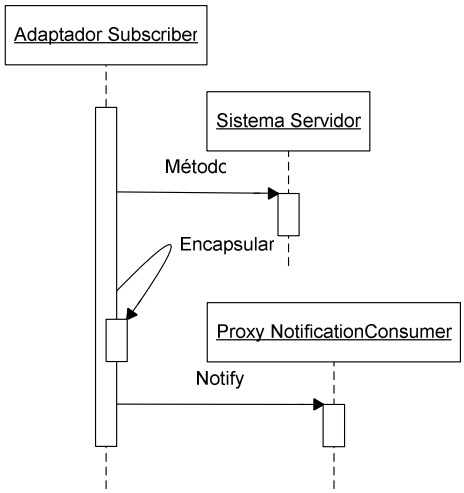

Figura 24 - Funcionamento do Adaptador *Subscriber* para sistemas servidores com resposta.

6.1.4 Controlador de Respostas

Componente que tem por objetivo resolver total ou parcialmente as seguintes questões de integração:

- Suporte a respostas, necessárias para sistemas RPC, como apresentado na Seção 5.1.3, viabilizando a elas a chegada aos seus destinatários (i.e. sistemas clientes);
- Suporte à lógica de mediação do negócio, apresentado na Seção 5.1.4, permitindo que parte da lógica do negócio seja transferida para a Infra-Estrutura;

- Suporte a falhas de sistemas externos, apresentado na Seção 5.1.6, permitindo que respostas sejam entregues mesmo que algum sistema participante da interação não esteja disponível.

Assim, é proposto que esse componente armazene as respostas às notificações de tal forma que elas possam:

- Ser repassadas aos sistemas clientes mesmo em situações de falha;
- Ser manipuladas por um componente de acordo com alguma lógica de intermediação.

Esse componente está baseado no padrão arquitetural *Blackboard* [9], pois:

1. Trabalha com problemas que não possuem solução determinística como, por exemplo, o padrão de mediação Leilão, em que o número de respostas a uma solicitação pode variar;
2. É preparado para trabalhar com diversos sistemas heterogêneos de forma flexível;
3. Pode ser utilizado como um repositório para as respostas recebidas;
4. Permite o paralelismo;
5. Permite a utilização de um controle inteligente para a seleção de respostas, habilitando o trabalho da Infra-Estrutura como um intermediador em operações entre sistemas como descrito por [25], contendo parte das regras de negócio.

O padrão *Blackboard* é útil para problemas onde uma estratégia determinística não pode ser aplicada e funciona como um quadro de respostas em que, a partir de diversas fontes de conhecimento, uma solução final é escolhida. É um padrão utilizado na área de inteligência artificial, sendo bastante flexível e tolerante a falhas. Sua aplicação é bastante ampla e pode possuir diversas variações, já tendo sido utilizado para a integração de sistemas de informação como em [34]. A Figura 25 apresenta a estrutura do *Blackboard* aplicada à Infra-Estrutura de integração para a criação do componente Controlador de Respostas.

A forma original de trabalho do *Blackboard* consiste em diversas fontes de conhecimento (*Knowledge Source*) funcionando em uma estrutura de dados comum. Cada fonte de conhecimento é responsável por resolver uma parte do problema. O padrão *Blackboard* também possui uma classe controladora (*Control*) responsável por gerenciar as demais classes e servir de interface para outros sistemas. O padrão, ainda, possui a classe *Blackboard*, que armazena a resposta selecionada.

Para a aplicação na Infra-Estrutura, foi considerado que cada fonte de conhecimento do *Blackboard* é uma Resposta *Subscriber*, ou seja, um elemento que contém a resposta dada por um assinante a uma solicitação. Quem faz esse gerenciamento é o Controle Mediação (o *Control* do padrão *Blackboard*), que recebe as mensagens de retorno e cria instâncias do Resposta *Subscriber*. Conforme as respostas chegam, a melhor resposta será selecionada e disponibilizada no elemento

Estado Mediação (equivalente à classe *Blackboard* do padrão). A Tabela 6 apresenta o mapeamento entre os elementos.

Tabela 6 – Relação entre os elementos do Padrão *Blackboard* e do Controlador de Mediação.

Componentes do Padrão *Blackboard*	Componentes do Controlador de Mediação
Knowledge Source	Resposta Subscriber
Control	Controle de Mediação
Blackboard	Estado Mediação

O critério para a finalização da escolha da resposta pode ser definido conforme cada caso, podendo ser, por exemplo, um tempo pré-determinado ou um número máximo de respostas.

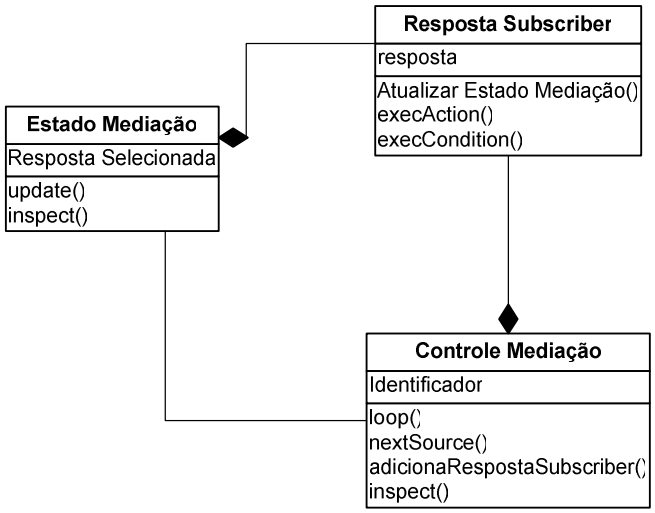

Figura 25 - Controlador de Respostas.

Os passos do funcionamento desse componente, detalhados na Figura 26, são baseados no funcionamento do padrão arquitetural *Blackboard* e estão descritos abaixo:

1. O componente Adaptador *Publisher* cria um Controle Mediação para cada nova invocação (Figura 18), permitindo, assim, um local para o armazenamento dos dados de resposta de cada requisição realizada aos servidores. A identificação desse Controle Mediação é realizada através de um identificador único, enviado juntamente com a mensagem e que deverá ser retornado com as mensagens-resposta;
2. Quando um Controle Mediação é criado, este instancia um novo objeto Estado Mediação;
3. O Adaptador *Publisher*, então, dispara o método loop() do Controle Mediação (Figura 18), que fica gerenciando as respostas recebidas;

4. Para cada novo objeto Resposta *Subscriber*, o método *execCondition* é chamado com o objetivo de inspecionar o Mediador Respostas para encontrar a solução mais adequada (a forma como os objetos Resposta *Subscriber* são criados é discutida na Seção 6.1.5);
5. Caso seja selecionado um objeto Resposta *Subscriber*, *execAction* é chamada com o objetivo de atualizar o Estado Mediação;
6. A execução do loop() termina após alguma condição de chamada ser alcançada. A condição de parada deve ser implementada de acordo com as necessidades do negócio;
7. O Adaptador *Publisher* chama o método *inspect*() do Controle Mediação para ler a resposta selecionada (Figura 18).

Tabela 7 - Descrição das classes participantes do Controlador de Respostas.

Elemento	Função
Estado Mediação *(Blackboard)*	Elemento responsável por armazenar a resposta selecionada.
Resposta Selecionada	Atributo que contém a melhor resposta selecionada através do algoritmo de seleção de respostas implementado.
update()	Método para atualizar a Resposta Selecionada.
inspect()	Método para ler a Resposta Selecionada.
Resposta *Subscriber* *(Knowledge Source)*	Elemento que contém a resposta postada por cada *Subscriber* (Sistema Servidor).
Resposta	Atributo com a mensagem respondida.
Atualizar Estado Mediação()	Método utilizado para atualizar o estado da Mediação, após ser verificado que a Resposta *Subscriber* é melhor que a resposta previamente selecionada.
execCondition()	Método que compara a resposta do *Subscriber* com a resposta previamente selecionada do Estado Mediação.
execAction()	Realiza o processamento sobre a resposta caso seja necessário.
Controle Mediação (Control)	Elemento responsável por gerenciar os demais elementos do Mediador de Respostas.
Identificador	Identificador único, que permite que uma resposta seja corretamente associada a uma requisição.
loop()	Método principal que chama os demais métodos da seleção.
nextSource()	Método que busca um elemento Resposta *Subscriber* e chama o seu método *execCondition*().
adicionaResposta Subscriber()	Método chamado quando uma nova resposta é recebida, utilizado para a criação de uma nova instância de Resposta *Subscriber*.
inspect()	Método utilizado para verificar a resposta selecionada e armazenada no Estado Mediação.

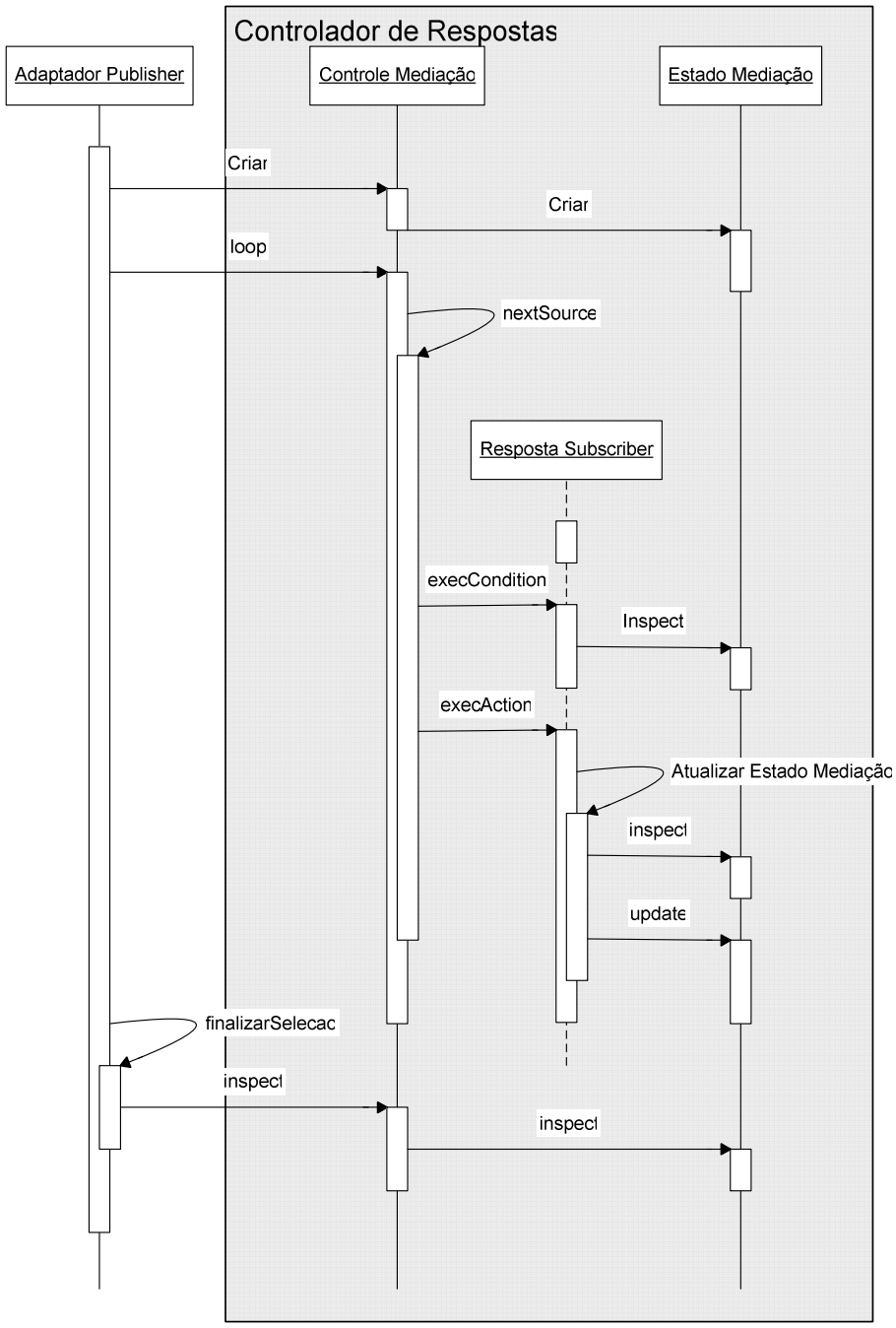

Figura 26 - Funcionamento do Controlador de Respostas.

6.1.5 Adaptador Controlador de Respostas

Componente que tem por objetivo repassar as respostas geradas por servidores, em resposta às requisições dos clientes e ao Controlador de Respostas. Complementa a solução à questão de suporte a respostas (Seção 5.1.3).

Para tanto, é necessário que a Infra-Estrutura transforme o componente Controlador de Respostas também em *Subscriber*, porque as mensagens retornadas pelos servidores serão enviadas ao *NotificationBroker* utilizando-se o formato *Notify*.

Para essa tarefa, o padrão *Adapter* pode ser novamente utilizado, como apresentado na Figura 27 e detalhado na Tabela 8. Esse componente deve estar registrado como assinante do *Topic* gerado pelo Adaptador *Subscriber* para que receba as mensagens de resposta.

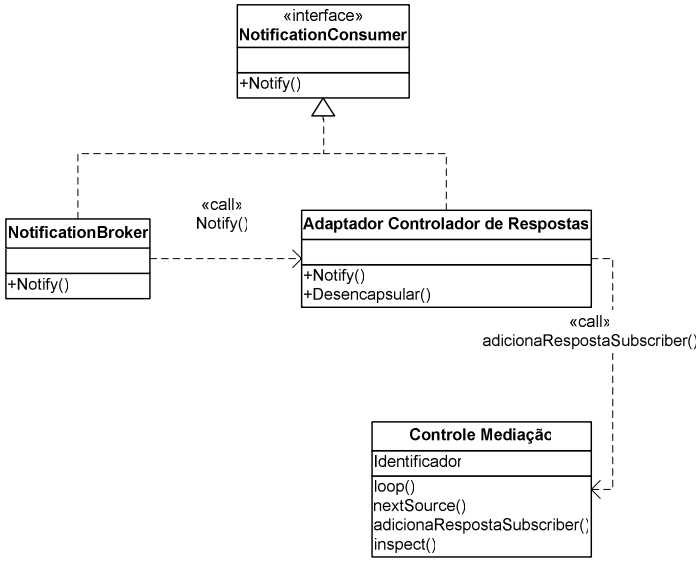

Figura 27 - Adaptador Controlador de Respostas.

Tabela 8 - Descrição das classes participantes do Adaptador Controlador de Respostas.

Elemento	Função
NotificationBroker	Sistema responsável por enviar a mensagem *Notify* para todos os assinantes registrados (descrito na Seção 3.4.1.)
Notify()	Método concreto, utilizado para o envio de mensagens *Notify*.
NotificationConsumer	Classe que fornece a interface que será utilizada pelo *Proxy* e pelo *NotificationBroker* (descrito na Seção 3.4.1.).
Notify()	Assinatura do método utilizado para o envio de uma mensagem *Notify*.
Adaptador Controlador de Respostas	Elemento responsável por receber e desencapsular a mensagem antes desta ser repassada para o Controlador de Respostas.
Notify()	Método que recebe a notificação e, então, chama o método Desencapsular e repassa a informação para o Controle Mediação (classe principal do elemento Controlador de Respostas).
Desencapsular()	Método responsável por desencapsular à mensagem *Notify*.
Controle Mediação	Elemento responsável por gerenciar os demais elementos do componente Controlador de Respostas, servindo de ponto de acesso do componente.
Identificador	Identificador único, que permite que uma resposta seja corretamente associada a uma requisição.
loop()	Método principal que chama os demais métodos da seleção.
nextSource()	Método que busca um elemento Resposta *Subscriber* e chama o seu método *execCondition*().
adicionaResposta Subscriber()	Método chamado quando uma nova resposta é recebida, utilizado para a criação de uma nova instância de Resposta *Subscriber*.
inspect()	Método utilizado para verificar a resposta selecionada e armazenada no Estado Mediação.

O comportamento desse componente na Infra-Estrutura é apresentado na Figura 28. Tal elemento recebe a notificação de resposta enviada pelo Adaptador *Subscriber* através do *NotificationBroker*, desencapsula a mensagem e chama o método do Controle Mediação para que uma nova resposta seja adicionada.

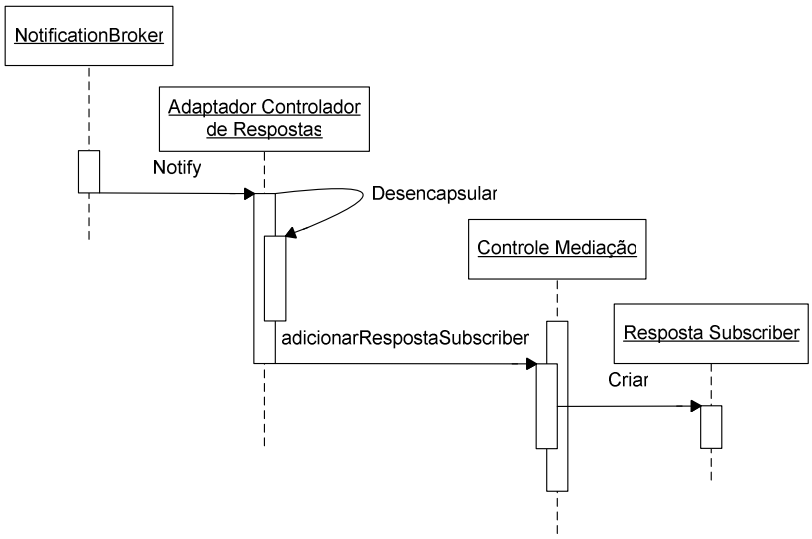

Figura 28 - Funcionamento do Adaptador Controlador de Respostas.

6.1.6 Detalhamento da Infra-Estrutura

A Infra-Estrutura tem como objetivo o auxílio à integração de sistemas no modelo EDA utilizando-se de um *NotificationBroker* para a comunicação. Os componentes apresentados têm por objetivo tornar isso possível. A partir da identificação desses componentes, é realizada a sua composição, formando a Infra-Estrutura apresentada na Figura 29.

Apenas para legibilidade da Figura 29, foi apresentada, diversas vezes, a interface *NotificationConsumer*, tratando-se, porém, da mesma interface. As diferentes classes da Infra-Estrutura que a implementam estão detalhadas na Figura 30.

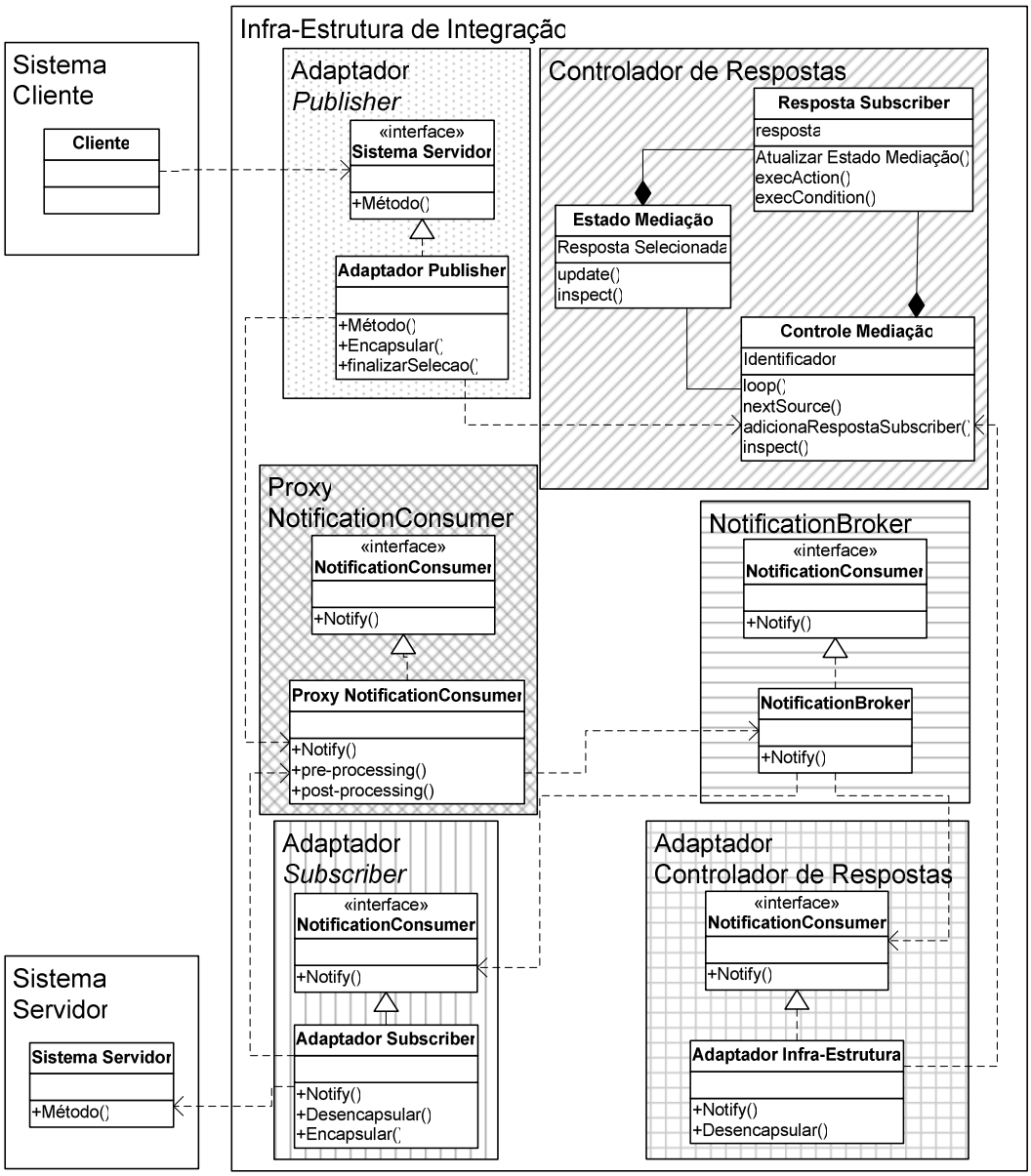

Figura 29 - Infra-Estrutura de Integração.

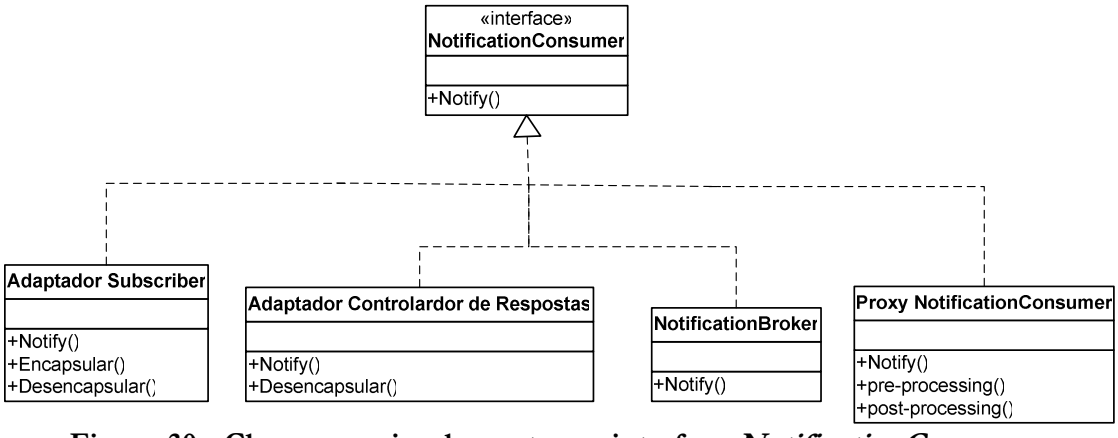

Figura 30 - Classes que implementam a interface *NotificationConsumer*.

6.2 Diretrizes para a utilização da Infra-Estrutura

6.2.1 Análise

Para a utilização da Infra-Estrutura, deve-se, primeiramente, extrair do contexto da integração algumas informações. São elas:

1. Formato das requisições efetuadas pelo Sistema Cliente;
2. Formato das respostas efetuadas pelos Sistemas Servidores, podendo ser diferentes, dependendo das interfaces dos servidores;
3. Interface(s) do Sistema Servidor que será(ão) utilizada(s) pelo Adaptador *Publisher*;
4. Lógica de intermediação (como se seleciona uma resposta);
5. Definição de um critério de parada para a seleção de respostas (quando se deve parar de selecionar respostas);
6. Definição dos *Topic* que serão criados, tanto para as chamadas dos Sistemas Clientes quanto para as respostas dos Sistemas Servidores;

6.2.2 Projeto

Com essas informações, os seguintes passos devem ser seguidos para que a Infra-Estrutura fique pronta para intermediar as interações entre Sistemas Clientes e Sistemas Servidores, de acordo com a Infra-Estrutura de Integração proposta:

1. Implementação do Adaptador *Publisher*:
    a. Criação desse componente a partir das interfaces requeridas do Sistema Servidor. Caso exista mais de um Sistema Cliente, é possível que seja necessário implementar mais de uma interface ou até mesmo um Adaptador *Publisher* para cada cliente.
    b. Encapsulamento dos dados da requisição no formato *Notify*. Todos os dados necessários para realizar a chamada ao sistema servidor devem ser encapsulados.
    c. Definição de um critério de parada para a implementação do método *finalizarSelecao*. Deve-se selecionar como será realizada a interrupção do *loop* do Controle Mediação. Para cada caso em que a Infra-Estrutura for aplicada, é possível escolher um novo critério de parada para a seleção de respostas.
2. Implementação do Adaptador *Subscriber*:
    a. Diferentes Sistemas Servidores podem possuir diferentes Interfaces. Nessa linha de raciocínio, pode ser necessário criar mais de um Adaptador *Subscriber*. Cada um deverá implementar um método *Notify* para receber a notificação e um método Desencapsular, para converter os dados da notificação na chamada ao Sistema Servidor;

   b. Pode ser importante que a resposta à notificação seja retornada para o Sistema Cliente. Nesse caso, esses dados devem ser encapsulados e enviados para o *NotificationBroker*. Isso é implementado no método Encapsular de tal componente;
3. Implementação do Controlador de Respostas:
   a. A estrutura de dados trabalhada é a mesma dos dados retornados do Sistema Servidor (e armazenada nos Resposta *Subscriber*). Caso existam diferentes servidores, ou todas as respostas devem ser convertidas para alguma padronização antes de serem adicionadas, ou o Controlador de Respostas deve suportar os diferentes tipos de respostas dos servidores.
   b. A lógica de intermediação deve ser colocada no método *execCondition* de cada Resposta *Subscriber*.

6.2.3 Configuração

Após a implementação, é necessário que sejam configurados:

1. A chamada do Sistema Cliente para a Infra-Estrutura, e não mais diretamente para o Sistema Servidor;
2. O *Proxy NotificationConsumer* deve ser autorizado a enviar mensagens do tipo *Notify* (verificação de permissões de acesso);
3. Criação de um *Topic* no *NotificationBroker* para as mensagens geradas a partir das requisições do Sistema Cliente;
4. Realização da assinatura dessas mensagens pelo Adaptador *Subscriber* (deve ser realizada uma assinatura para cada Adaptador *Subscriber*);
5. Criação de um *Topic* no *NotificationBroker* para as mensagens de repostas do Sistema Servidor (deve ser criado um *Topic* para cada tipo de respostas diferente, caso existam);
6. Realização da assinatura dessas mensagens pelo Adaptador Controlador de Respostas;
7. Criação de regras e filtros no *NotificationBroker* para transformar mensagens, caso as estruturas de resposta sejam diferentes para Sistemas Servidores diferentes.

6.3 Considerações e Variações

6.3.1 Pontos de acesso e adaptação

Para os clientes, os pontos de acesso representam os servidores de serviços que estão acessando. Uma arquitetura pode possuir diversos clientes e diversos fornecedores e, com isso, uma interação de ligação de n para m.

A utilização de um *NotificationBroker* em uma arquitetura tende a condensar as chamadas de maneira que todas as interações entre sistemas passem por ele. Nesse contexto, existem duas alternativas para a implementação dos Adaptadores *Publisher*, variando seu número necessário para a integração de sistemas com o *NotificationBroker*.

A primeira é a colocação de um Adaptador *Publisher* para cada sistema, mantendo os demais mecanismos de integração compartilhados. Essa alternativa possibilita a utilização de interfaces mais genéricas, aumentando a flexibilidade, apesar de sua implementação e manutenção serem mais trabalhosas. Por essa solução ser mais custosa, deve-se verificar a sua real necessidade.

A segunda alternativa é concentrar todas as interfaces em um único Adaptador *Publisher*. É uma alternativa menos flexível e pode ser utilizada caso a interface requerida por vários sistemas seja a mesma, ou seja possível realizar a "fusão" das interfaces requeridas em um único Adaptador *Publisher*. A vantagem da utilização dessa abordagem é a diminuição da quantidade de código necessária, diminuindo também custos de manutenção.

Como a segunda alternativa concentra várias chamadas em um único ponto de acesso, é necessário que algum mecanismo de identificação do cliente que gerou o evento seja criado.

Para resolver essa questão, existem três alternativas possíveis:

1. Passagem da identificação através da chamada do serviço, utilizando o método GET: utiliza-se a URL na chamada do serviço para passar uma variável extra de identificação, permitindo de forma simples a identificação do cliente;
2. Identificação através de mecanismos de autenticação das aplicações clientes na rede: permite uma maior segurança, mas pode ser inviável dependo do tipo de sistema que está interagindo;
3. Utilização de *portTypes* diferentes, um para cada cliente: permite uma flexibilidade um pouco maior que a opção anterior, pois o método e os parâmetros chamados não necessitam ser os mesmos para cada cliente. Essa alternativa é inviável caso as interfaces possuam grandes diferenças como, por exemplo, quando um sistema utiliza mensagens sobre *Web Services* e outro utiliza RPC.

6.3.2 Resposta direta ao Controlador de Respostas

Quando não existe a necessidade da resposta a uma solicitação ser enviada para mais de um recipiente, é possível retornar os dados diretamente para o Controlador de Respostas, evitando passar novamente pelo *NotificationBroker* e melhorando o desempenho na comunicação.

A Figura 31 apresenta como a Infra-Estrutura fica nesse caso. É possível observar que não existe o componente Adaptador Controlador de Respostas, já que ele se torna desnecessário.

Outra modificação consiste no fato de que o elemento Encapsular do componente Adaptador *Subscriber* não é mais utilizado, sendo que os dados retornados pelo Sistema Servidor são repassados diretamente ao Controlador de Respostas.

A utilização dessa variação impede que outros sistemas, além do Sistema Cliente, utilizem à resposta do Sistema Servidor.

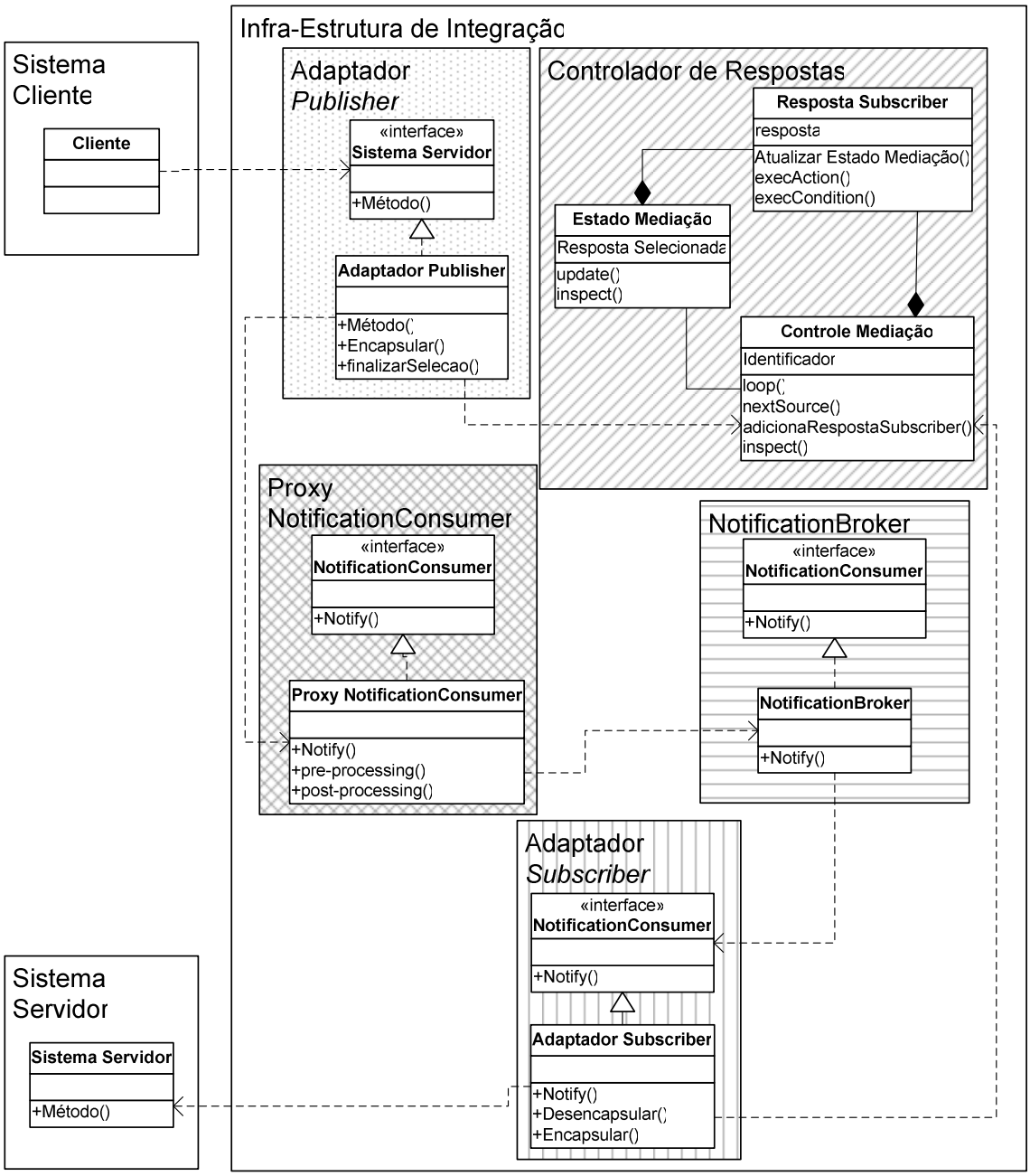

Figura 31 - Infra-Estrutura com utilização de resposta direta ao Controlador de Respostas.

6.3.3 *Proxy* do Servidor

O padrão de projeto *Proxy* também pode ser utilizado, além da comunicação com o *NotificationBroker,* através do *Proxy NotificationConsumer* para a comunicação com os Sistemas Servidores que serão adaptados. Assim, é possível criar um *Proxy* Sistema Servidor para a comunicação com este. A estrutura do padrão fica como apresentado na Figura 32.

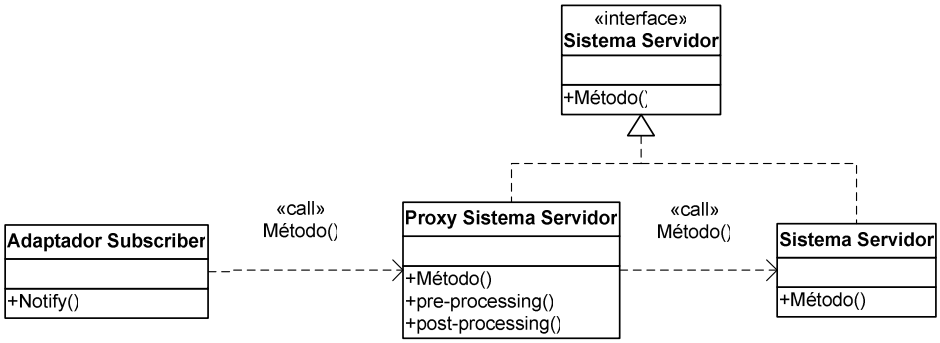

Figura 32 - Estrutura do padrão *Proxy* aplicado para o acesso a um Sistema Servidor.

6.4 Vantagens da Infra-Estrutura

A utilização efetiva dessa Infra-Estrutura está ligada a três pontos principais:

1. Adaptação das interfaces dos Sistemas Clientes;
2. Adaptação das interfaces dos Sistemas Servidores;
3. Criação de um mediador com uma lógica de seleção de respostas.

Aplicando os três pontos, é possível sistematizar a aplicação de Infra-Estrutura na adaptação de sistemas para o modelo EDA.

A criação da lógica para a seleção de respostas pode variar a sua complexidade, conforme regras de negócio mais complexas são adicionadas a ela. Como apresentado anteriormente, o componente Controlador de Respostas é bastante flexível, permitindo uma ampla utilização de padrões de intermediação de negócios.

A vantagem de utilizar essa Infra-Estrutura está em evitar ou diminuir a complexidade de todas as questões de integração já apresentadas anteriormente, podendo ser adaptada a vários domínios, por ser genérica, e permitir que a lógica de mediação seja alterada conforme a necessidade de cada adaptação. Assim, auxilia na adaptação de sistemas de maneira sistemática, com menor esforço e custo, de forma não intrusiva.

7 APLICAÇÃO DA INFRA-ESTRUTURA

A aplicação da Infra-Estrutura em estudos de caso visa possibilitar a sua avaliação em uma ambiente de testes para que seu comportamento possa ser comparado à utilização de uma aplicação sem ela. Como esta pesquisa se caracteriza por ser de natureza exploratória, a utilização de estudo de caso para avaliação da proposta é válida [35].

Além disso, a Infra-Estrutura atua no *background* das aplicações de forma transparente aos usuários, sendo o desempenho da aplicação a única diferença perceptível entre a comunicação através da Infra-Estrutura e sem ela. Para verificar a influência que a utilização da Infra-Estrutura causa no funcionamento da aplicação, testes de desempenho foram realizados, a fim de determinar se seu emprego é ou não viável.

Este capítulo apresenta os componentes e configurações dos estudos de casos aplicados. Primeiramente, é apresentada a especificação WS-I SCM que contém os sistemas a serem utilizados em todos os estudos de casos executados. Em seguida, são apresentadas as considerações gerais sobre os desenvolvimentos realizados, de forma a caracterizar as tecnologias e arquiteturas utilizadas para que os testes possam ser repetidos.

Para tanto, são apresentados os projetos dos três estudos de casos desenvolvidos, bem como a configuração utilizada em cada um deles. Ao final do capítulo, é apresentado o resultado dos testes de desempenho, bem como uma discussão sobre os resultados encontrados.

7.1 Especificação WS-I SCM

Para a aplicação de exemplos práticos, será utilizado o sistema desenvolvido por empresas para a especificação WS-I[**]. A especificação WS-I é um conjunto de requisitos para que os *Web Services* garantam a interoperabilidade entre diversas plataformas. Esse sistema simula uma cadeia de suprimentos com clientes, estoque e manufatura.

Para que as empresas que fornecem plataformas de desenvolvimento de *Web Services* comprovem que seus produtos estão de acordo com a especificação WS-I, elas devem desenvolver um protótipo que esteja de acordo com os requisitos da WS-I. Esse protótipo é chamado WS-I SCM (*WS-I Supply Chain Management*) [36] [37]. O WS-I SCM foi escolhido por estar de acordo com alguns requisitos para o desenvolvimento do estudo de caso:

[**] www.ws-i.org

1. Possuir acesso direto às interfaces de aplicação, pois é classificado como um sistema Altamente Decomponível;
2. Ser um sistema distribuído;
3. Possuir uma especificação de requisitos completa;
4. Possuir um projeto bem definido;
5. Possuir implementação em diversas plataformas;
6. Simular uma aplicação real;
7. Possuir seu código fonte disponível, para que *counters* possam ser instalados e seu desempenho monitorado.

Entre as empresas que desenvolveram esse sistema, destacam-se BEA Systems, IBM, Microsoft, Novell, Oracle, SAP e Sun Microsystems. Tal sistema utiliza-se, basicamente, de chamadas síncronas a *Web Services*, com o total acoplamento dos elementos participantes.

O sistema é dividido em três partes, como ilustrado na Figura 33: a) *WebClient* e *Retailer*, b) *Retailer* e *Warehouse* e c) *Warehouse* e *Manufacturer*. O sistema *LoggingFacility* também faz parte da WS-I SCM, mas não é explorado neste estudo de caso. O *WebClient* é uma aplicação *Web* que se comunica através de *Web Services* com o sistema *Retailer*. O *Retailer* comunica-se com as *Warehouse* para preencher os pedidos. Quando uma *Warehouse* não pode preencher um pedido, ela envia uma nova requisição de produtos para as *Manufacturer*.

A comunicação é realizada utilizando-se apenas *Web Services*. Com a introdução do *NotificationBroker*, diversas modificações devem ser realizadas nos aplicativos que compõem o sistema, tornando perceptíveis os problemas de adaptação de sistemas legados para o funcionamento na nova arquitetura.

Na arquitetura original da WS-I SCM, os endereços dos sistemas parceiros estão armazenados em um componente de configuração, não sendo possível a alteração em tempo de execução. Além disso, o sistema não permite que novos componentes sejam adicionados à arquitetura, sendo que o número de *Warehouses* e *Manufacturers* é fixo. Por não possuir nenhum Intermediador, a lógica está toda nos sistemas e não pode ser alterada sem que os diversos elementos sejam reprojetados. Os estudos de casos houveram de ser desenvolvidos dentro dessas restrições.

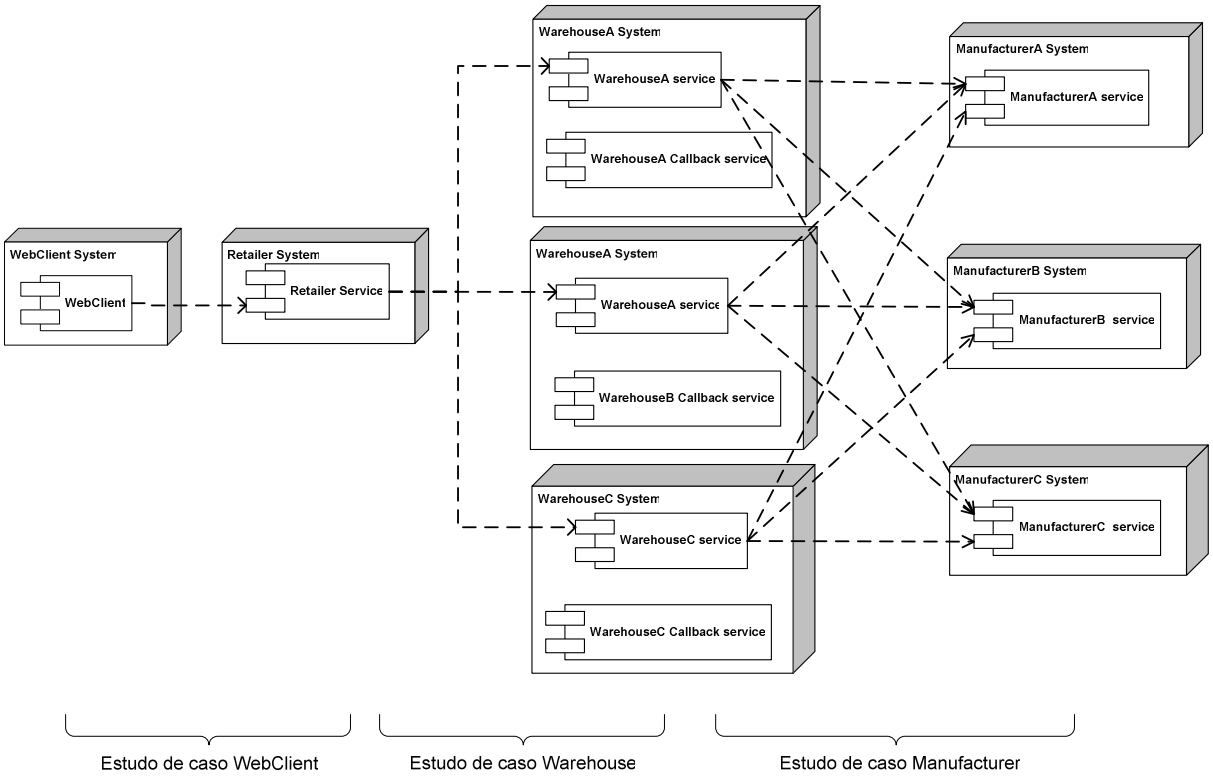

Figura 33 - Estrutura da WS-I SCM.

7.2 Considerações Gerais

7.2.1 Componentes comuns

A Infra-Estrutura possui um conjunto de componentes fixos que são sempre utilizados. As customizações que devem ser realizadas para a aplicação da Infra-Estrutura geralmente são parecidas e variam conforme cada caso. Ainda existem alguns componentes que são totalmente reusados. A Tabela 9 apresenta o reuso dos componentes nos estudos de caso aplicados.

O *Proxy NotificationConsumer* é exatamente o mesmo, pois trabalha apenas com mensagens do tipo *Notify*. O mesmo vale para o Adaptador Controlador de Respostas que, apesar de desencapsular as mensagens, não necessita manipular o seu conteúdo, logo não existindo a necessidade do seu entendimento.

Tabela 9 - Estudos de caso e os componentes reusáveis da Infra-Estrutura.

Componentes/Métodos	Estudos de caso: *WebClient*	*Warehouse*	*Manufacturer*
Adaptador *Publisher*			
Proxy NotificationConsumer	✓	✓	✓
Adaptador *Subscriber*			
Controlador de Respostas			
Adaptador do Controlador de Respostas	✓	✓	✓

7.2.2 Plataforma de desenvolvimento

O sistema foi desenvolvido sobre plataforma Microsoft .NET. O sistema operacional utilizado foi o Windows XP, o banco de dados o MSDE 2000 e o servidor Web o IIS 5.1.

7.2.3 Estrutura do *NotificationBroker* construído

Para os estudos de caso, foi desenvolvido um *NotificationBroker* simples, contendo apenas as funcionalidades básicas para o seu funcionamento, cuja estrutura é apresentada na Figura 34.

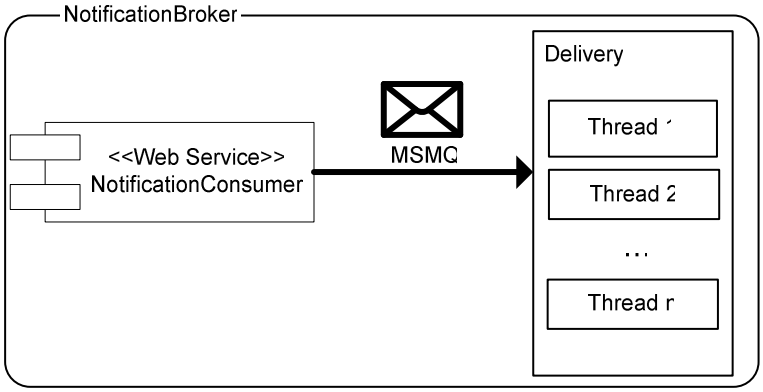

Figura 34 - Estrutura do *NotificationBroker* utilizado.

As mensagens *Notify* são recebidas através de um Web Service que implementa a interface *NotificationConsumer*. Esse encapsula a mensagem e gera uma mensagem MSMQ (*Microsoft Message Queuing* - recurso disponível em sistemas operacionais da Microsoft) que é recebida por um sistema assinante denominado *Delivery*.

O *Delivery* foi desenvolvido como sendo uma aplicação Windows, que possui uma listagem de assinaturas do *NotificationBroker*. Como uma mensagem pode possuir diversos assinantes, o *Delivery* gera uma *Thread* (linha de execução paralela) para cada mensagem a ser

entregue a um assinante. Assim, mesmo que um assinante esteja sobrecarregado ou não esteja disponível, outros assinantes não serão afetados no recebimento das mensagens.

7.2.4 Implementação dos Componentes

Por questões de desempenho e facilidade de desenvolvimento, sempre que possível foram utilizados componentes de software para a implementação da Infra-Estrutura. A utilização de *Web Services* se deu apenas em pontos onde as mensagens trocadas eram no formato *Notify* da *WS-Notification*, que exige que a interface provida seja um Web Service.

Assim, o *NotificationBroker*, os Adaptadores *Subscriber* e os Adaptadores do Controlador de Respostas foram implementados utilizando-se Web Services.

Nos estudos de caso, também os Adaptadores *Publisher* foram implementados, sendo Web Services, pois os sistemas clientes requisitam esse tipo de interface.

7.3 Estudo de Caso: WS-I SCM *Warehouses*

7.3.1 Análise

A primeira aplicação da Infra-Estrutura foi realizada na intermediação entre o sistema *Retailer* e as *Warehouse*. A Figura 35 representa o domínio dessa integração. Conforme a Seção 6.2.1, para a utilização da Infra-Estrutura deve-se, primeiramente, extrair do contexto da integração algumas informações, que estão descritas na Tabela 10.

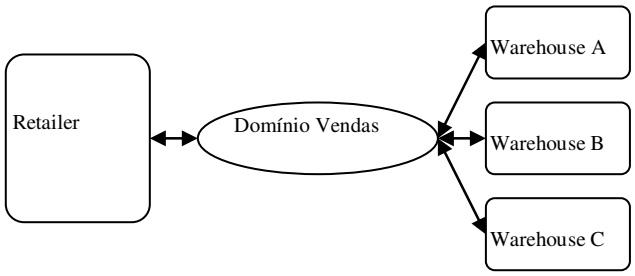

Figura 35 - Domínio do estudo de caso WS-I SCM *Warehouse*.

Tabela 10 - Elementos identificados para a adaptação.

Informação	Descrição
Formato das requisições do *Retailer*	**1.** *Soap Header*: cabeçalho enviado em uma requisição. Possui os elementos: a. *Configuration*: configurações diversas do ambiente WS-I SCM; b. *TimestampHeader*: tempo de validade de uma requisição, verificado pela *Warehouse*; **2.** *SOAP Body*: corpo da chamada. O elemento é: a. *ShipGoods***:** estrutura de dados contendo uma lista dos itens a serem comprados com a sua quantidade desejada.
Formato da resposta da *Warehouse*	*ShipGoodsResponse*: estrutura de dados contendo o nome da *Warehouse* e uma lista com os códigos dos itens que estão disponíveis em estoque.
Interface da *Warehouse*	*ShipGoods*: método utilizado para enviar o pedido ao sistema.
Lógica de intermediação selecionada	Para o estudo de caso, foi escolhida uma lógica simples para a seleção de respostas. Essa escolha foi aleatória e apenas para efeitos de avaliação da Infra-Estrutura. A seleção é a da resposta que melhor atendeu à solicitação (mais itens solicitados em estoque), no menor tempo.
Critério de parada	O critério de parada foi escolhido, da mesma forma que a Lógica de intermediação, de maneira aleatória e tem caráter apenas de exemplo. O critério de parada foi o tempo limite de um segundo para que uma resposta seja selecionada.
Topics **Criados**	Os *Topic* representam o tipo de mensagem que é repassada através do *NotificationBroker*. Foi definido que cada *Warehouse* retornará um *Topic* diferente para as suas mensagens, aumentando a flexibilidade na medida em que também permite que outros sistemas recebam notificações de acordo com a *Warehouse* do seu interesse. As mensagens são: 1. *Warehouse.ShipGoods*: *Topic* para uma nova requisição; 2. *Warehouse.ShipGoodsResponseA*: *Topic* para uma resposta à *Warehouse* A; 3. *Warehouse.ShipGoodsResponseB*: *Topic* para uma resposta à *Warehouse* B. 4. *Warehouse.ShipGoodsResponseC*: *Topic* para uma resposta à *Warehouse* C. Teria sido possível utilizar apenas um *Topic* para todas as *Warehouses*, porém tornando a solução menos flexível.

7.3.2 Implementação

7.3.2.1 Adaptador *Publisher*

Esse componente foi implementado com um *Web Service* para estar de acordo com a interface das *Warehouse*. De acordo com a comportamento padrão descrito para esse tipo de componente (Seção 6.1.1) o diagrama da Figura 36 mostra como ele se relaciona com os demais elementos da aplicação e da Infra-Estrutura.

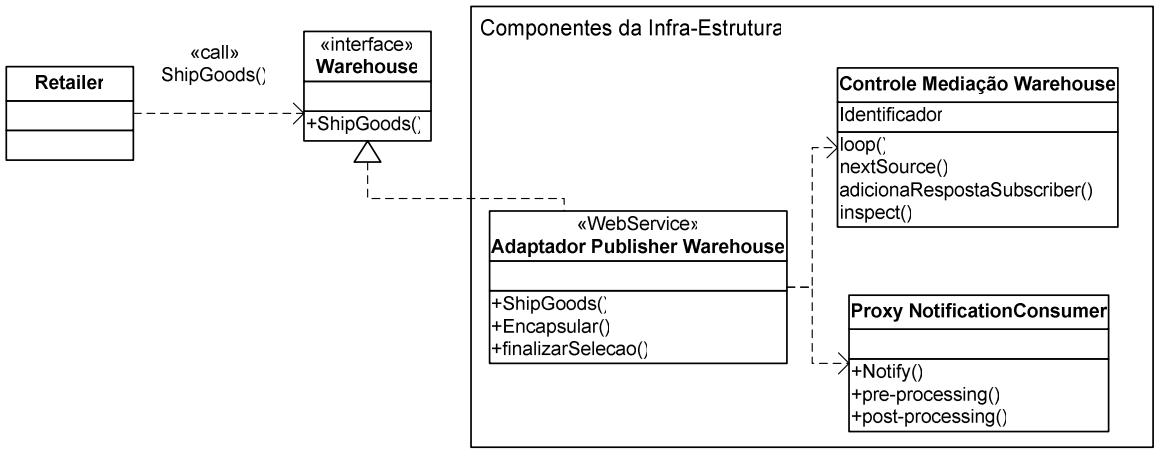

Figura 36 - Estrutura do Adaptador *Publisher* aplicado ao WS-I SCM Vendas.

O método *ShipGoods* do Adaptador *Publisher Warehouse* recebe a solicitação de uma nova ordem de compra, cria um novo Controle Mediação *Warehouse* e armazena esse objeto em uma variável de aplicação para ser posteriormente lido, além acionar o método *Encapsular* responsável por armazenar todas as informações relevantes da chamada e enviar a notificação para o *Proxy NotificationConsumer*. Após o envio da notificação, um marcador de tempo é acionado para a chamada do método *finalizarSelecao* que irá ler a resposta selecionada.

Para este estudo de caso, optou-se por criar apenas um Adaptador *Publisher* para todas as *Warehouses*, uma vez que as interfaces de todas as *Warehouses* são iguais e a diferenciação entre as *Warehouse* chamadas pode ser feita pela URL utilizada em cada chamada.

7.3.2.2 Adaptador *Subscriber*

A implementação do Adaptador *Subscriber* foi realizada para cada *Warehouse* individualmente, como mostra a Figura 38. Cada um desses adaptadores segue a estrutura do diagrama da Figura 38. A decisão ou não de ser desenvolvido um ou mais *Web Services* é do projetista e deve ser realizada de acordo com cada situação.

Para este estudo de caso, a decisão foi tomada apenas para efeitos de exemplificação. Porém, poderia ter sido considerada a possibilidade de um único Adaptador *Subscriber* para o conjunto de *Warehouse*, pois as suas interfaces são iguais. Contudo, tal solução seria menos flexível, visto que pressupõe que as interfaces de todos os sistemas servidores estejam disponíveis para o acesso direto, ou seja, sem problemas com restrições de ambiente como, por exemplo, *firewalls*.

O comportamento desse componente segue o padrão definido na Seção 6.1.3.

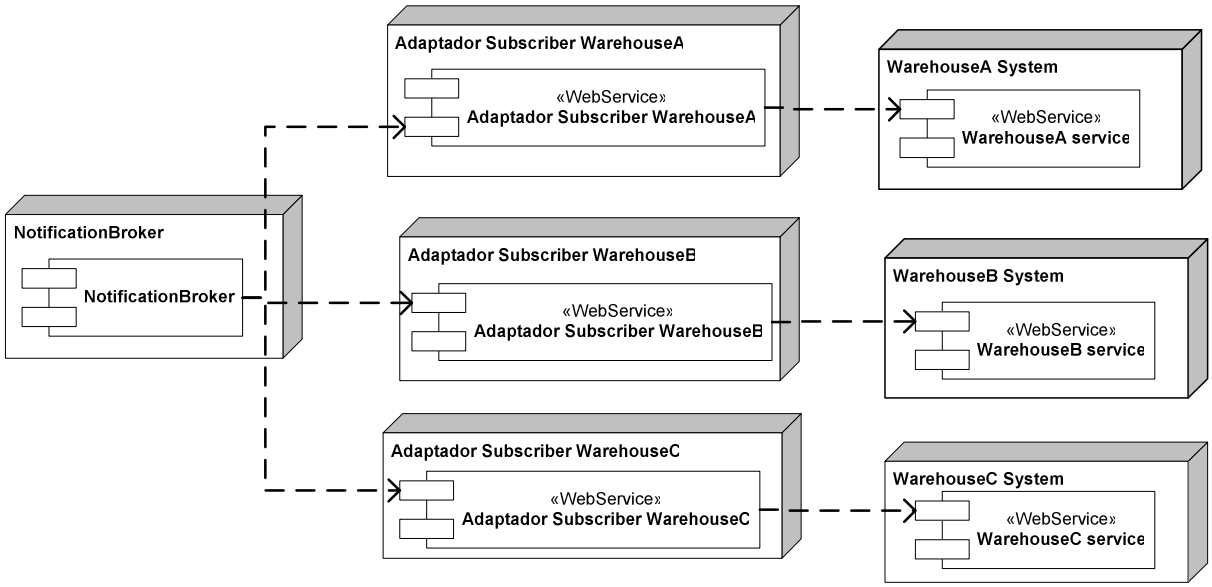

Figura 37 - Estrutura dos Adaptadores *Subscriber* criados para as *Warehouse*.

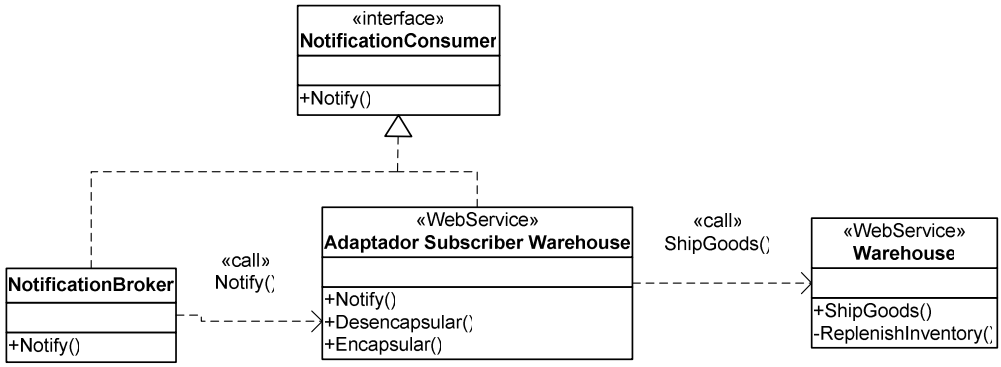

Figura 38 - Estrutura do componente Adaptador *Subscriber* aplicado ao WS-I SCM Vendas.

7.3.2.3 Controlador de Respostas

Para a implementação do Controlador de Respostas, foi aplicada a lógica de seleção de respostas descrita anteriormente. Como já mencionado, a criação de uma instância desse componente é realizada pelo Adaptador *Publisher Warehouse*, a qual é armazenada em uma área de aplicação da Infra-Estrutura para que possa ser acessada pelos diversos processos concorrentes. É o Adaptador Controlador de Respostas que repassa as mensagens das respostas recebidas para o Controlador de Respostas.

A lógica de seleção de respostas é realizada dentro do método *execCondition* de cada Resposta *Subscriber*, conforme detalhado na Seção 6.1.5.

7.3.3 Configuração

As configurações necessárias estão descritas na Tabela 11, conforme os itens descritos na Seção 6.2.3.

Tabela 11 - Configurações necessárias para o estudo de caso WS-I SCM *Warehouse*.

<table>
<tr><th>Item</th><th>Descrição da configuração realizada</th></tr>
<tr><td>1</td><td>O sistema possui um arquivo de configuração, que pode ser facilmente alterado para que as chamadas sejam redirecionadas para a Infra-Estrutura.</td></tr>
<tr><td>2</td><td>Os testes foram executados localmente utilizando-se o usuário administrador do sistema, não existindo qualquer problema de permissão.</td></tr>
<tr><td>3</td><td>Foi criado o Topic "Warehouse.ShipGoods" para as mensagens geradas a partir de uma chamada ao método ShipGoods de uma Warehouse.</td></tr>
<tr><td>4</td><td>As assinaturas criadas foram:

<table>
<tr><td>Topic</td><td>Assinante</td></tr>
<tr><td>Warehouse.ShipGoods</td><td>Adaptador_Subscriber_WarehouseA</td></tr>
<tr><td>Warehouse.ShipGoods</td><td>Adaptador_Subscriber_WarehouseB</td></tr>
<tr><td>Warehouse.ShipGoods</td><td>Adaptador_Subscriber_WarehouseC</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>Os Topic criados para as mensagens de respostas foram:
1. Warehouse.ShipGoodsResponseA
2. Warehouse.ShipGoodsResponseB
3. Warehouse.ShipGoodsResponseC</td></tr>
<tr><td>6</td><td>As assinaturas criadas para as respostas foram:

<table>
<tr><td>Topic</td><td>Assinante</td></tr>
<tr><td>Warehouse.ShipGoodsResponseA</td><td>Adaptador_Controlador_Respostas</td></tr>
<tr><td>Warehouse.ShipGoodsResponseB</td><td>Adaptador_Controlador_Respostas</td></tr>
<tr><td>Warehouse.ShipGoodsResponseC</td><td>Adaptador_Controlador_Respostas</td></tr>
</table>
</td></tr>
<tr><td>7</td><td>Não foi necessário criar nenhum filtro, pois o formato das mensagens é o mesmo.</td></tr>
</table>

Além dessas configurações, os estoques das *Warehouse* foram colocados de forma que apenas a *Warehouse* C possuísse produtos em estoque para satisfazer a requisição do sistema de varejo, assim os testes executados sem a Infra-Estrutura acabam por se comunicar com as três *Warehouse*.

7.4 Estudo de Caso: WS-I SCM *Manufacturer*

7.4.1 Análise

As manufaturas são chamadas sempre que uma *Warehouse* está com um nível baixo de estoque. As chamadas apenas enviam uma ordem de produção de algum equipamento, porém não esperam resposta. Assim, a Infra-Estrutura é utilizada apenas como um sistema de preparação de interfaces para o *NotificationBroker*, não havendo a necessidade de se utilizar o Controlador de Respostas.

Para este estudo de caso, os estoques das *Warehouse* foram colocados no mínimo, de maneira que uma ordem de produção é enviada a cada nova compra. Os elementos identificados estão descritos na Tabela 12 conforme os itens descritos na Seção 6.2.1.

Tabela 12 - Elementos identificados para a adaptação.

Informação	Descrição
Formato das requisições das *Warehouses*	**1.** *Soap Header*: cabeçalho enviado em uma requisição. Possui os elementos: a. *Configuration*: configurações diversas do ambiente WS-I SCM; b. *StartHeader*: possui um endereço de resposta e um identificador da requisição; **2.** *SOAP Body*: corpo da chamada. Os elementos são: a. *PurchOrdType*: estrutura de dados contendo: i. *orderNum*: identificador da ordem de fabricação; ii. *customerRef*: referência da *Warehouse* que solicitou a fabricação dos itens; iii. *itens*: lista de itens a serem fabricados;
Formato da resposta da *Manufacturer*	Não existe resposta à requisição.
Interface da *Manufacturer*	*submitPO*: método utilizado para enviar o pedido ao sistema.
Lógica de intermediação selecionada	Para este estudo de caso, não é necessária a utilização de uma lógica de intermediação.
Critério de parada	Não é necessário critério de parada, pois não existem respostas a serem selecionadas.
Topics **Criados**	Os *Topic* foram criados de acordo com a manufatura à qual mensagens são enviadas. Isso porque, pelas regras de negócio, cada *Manufacturer* produz apenas a sua marca de produto. Essa situação é diferente do estudo de caso anterior, onde as *Warehouse* armazenam todas as marcas e, por isso possuíam apenas um *Topic* para as requisições. As mensagens são: 1. *Manufacturer.submitPOA*: *Topic* para uma nova requisição à manufatura A; 2. *Manufacturer.submitPOB*: *Topic* para uma nova requisição à manufatura B; 3. *Manufacturer.submitPOC*: *Topic* para uma nova requisição à manufatura C;

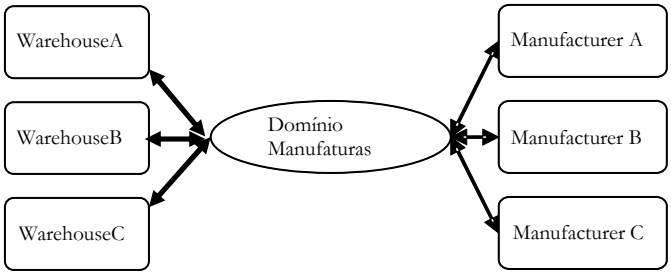

Figura 39 - Domínio de funcionamento do estudo de caso WS-I SCM *Manufacturer*.

7.4.2 Implementação

7.4.2.1 Adaptador *Publisher*

Esse componente foi implementado com um *Web Service* para estar de acordo com a interface das *Manufacturer*. A estratégia aplicada é a mesma já utilizada no estudo de caso anterior, ou seja, a utilização de apenas um Adaptador *Publisher* para todas as chamadas das *Warehouses*, identificando o chamado através de uma variável passada na URL.

7.4.2.2 Adaptador *Subscriber*

A implementação do Adaptador *Subscriber* foi realizada apenas uma vez, seguindo a mesma estratégia já utilizada para a criação do Adaptador *Publisher* deste estudo de caso (i.e. um único adaptador) como apresentado na Figura 40. Essa, portanto, é uma variação da estratégia usada para esse tipo de componente no estudo de caso anterior.

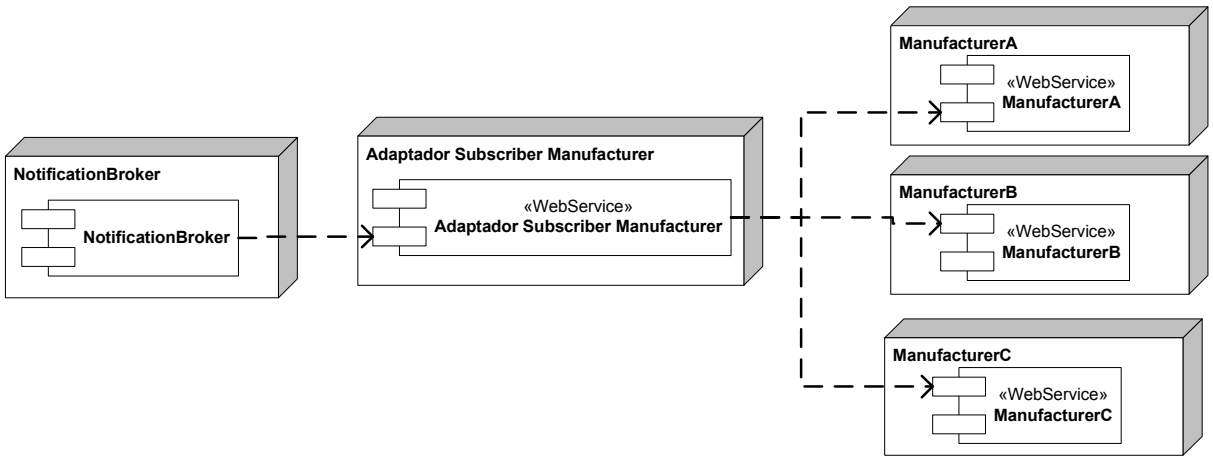

Figura 40 - Adaptador *Subscriber* implementado para as *Manufacturer*.

Como já mencionado, a decisão ou não de utilizar um ou vários Adaptador *Subscriber* deve ser do projetista, de acordo com as restrições arquiteturais impostas. Problemas com *firewalls* ou questões de desempenho podem sugerir que sejam criados vários Adaptadores *Subscriber*. Ainda é possível aplicar uma solução mista, variando de acordo com o Sistema Servidor que deve receber a mensagem.

7.4.2.3 Controlador de Respostas

Por não possuir mensagem de retorno, não existe a necessidade da criação de um Controlador de Respostas.

7.4.3 Configuração

As configurações necessárias estão descritas na Tabela 11, conforme os itens descritos na Seção 6.2.3.

Tabela 13 - Configurações necessárias para o estudo de caso WS-I SCM *Manufacturer*.

Item	Descrição da configuração realizada
1	O sistema possui um arquivo de configuração, que pode ser facilmente alterado para que as chamadas sejam redirecionadas para a Infra-Estrutura.
2	Os testes foram executados localmente utilizando-se o usuário administrador do sistema, não existindo qualquer problema de permissão.
3	Foram criados os *Topics*: 1. *Manufacturer.submitPOA* 2. *Manufacturer.submitPOB* 3. *Manufacturer.submitPOC* Os *Topics* são gerados respectivamente para as *Manufacturer* A, B e C, e representam uma chamada ao método *submitPO* que seria realizada a cada uma delas.
4	As assinaturas criadas foram: *Topic* | **Assinante** *Manufacturer.submitPOA* | *Adaptador_Subscriber_Manufacturer* *Manufacturer.submitPOA* | *Adaptador_Subscriber_Manufacturer* *Manufacturer.submitPOA* | *Adaptador_Subscriber_Manufacturer* Todas as mensagens são enviadas para o mesmo sistema assinante, porém, para cada assinatura a URL utilizada é diferente, contendo o código da *Manufacturer* que se quer endereçar, para que o Adaptador*_Subscriber_Manufacturer* possa realizar a diferenciação entre as chamadas.
5	Não existe mensagem de resposta, então não existe a necessidade de se ter um *Topic* para retorno de dados.
6	Não há assinaturas a serem criadas, pois não existe mensagem de retorno.
7	Não foi necessário criar nenhum filtro, pois o formato das mensagens é o mesmo.

7.5 Estudo de Caso: WS-I SCM *Retailer*

7.5.1 Análise

Este estudo de caso mostra que é possível aplicar a Infra-Estrutura também na comunicação entre o *WebClient* e o *Retailer*. A Figura 41 representa o domínio dessa integração e a Tabela 10 apresenta as configurações necessárias segundo a Seção 6.2.1.

Tabela 14 - Elementos identificados para a adaptação.

Informação	Descrição
Formato da requisição *submitOrder* e da requisição *GetCatalog* do *WebClient*	**1.** *Soap Header*: cabeçalho enviado em uma requisição. Possui os elementos: a. *Configuration*: configurações diversas do ambiente WS-I SCM; **2.** *SOAP Body*: corpo da chamada. Os elementos são: a. *SubmitOrderRequestType***:** estrutura de dados contendo uma lista dos itens a serem comprados e os detalhes de quem está comprando. i. *PartsOrderItem[]*: lista de itens a serem comprados. (*productNumber*: código do produto; *quantity*: quantidade; *price*: preço do produto para o cliente). ii. *CustomerDetailsType*: detalhes do cliente. (*custnbr***:** código do cliente; *name*: nome do cliente; *street1*: endereço; *street2*: endereço; *city*: cidade; *state*: estado; *zip*: CEP; *country*: país). O método ***GetCatalog*** também foi implementado, pois o *WebClient* utiliza a mesma URL de referência para os dois métodos. Porém, não foram realizados testes de desempenho para ele, já que nenhuma operação é realizada (apenas leitura de dados). Por essa razão, a estrutura de sua chamada não será detalhada aqui.
Formato da resposta do *Retailer*	*submitOrderResponse*: estrutura de resposta a uma solicitação *submitOrder* contendo o status dos itens pedidos.
Interface do *Retailer*	*submitOrder*: método utilizado para enviar o pedido ao sistema.
Lógica de intermediação selecionada	Para o estudo de caso, como apenas uma resposta é retornada, não existe nenhuma lógica de intermediação implementada, sendo que o Controlador de Respostas apenas é utilizado como repositório temporário para a resposta recebida.
Critério de parada	O critério de parada foi um tempo limite de um segundo para que a resposta seja lida.
Topics **Criados**	Os *Topics* criados são: 1. *Retailer.SubmitOrder: Topic* solicitando uma nova compra; 2. *Retailer.submitOrderResponse: Topic* para a resposta a uma solicitação;

Figura 41 - Domínio do estudo de caso WS-I SCM *Retailer*.

7.5.2 Implementação

7.5.2.1 Adaptador *Publisher*

Esse componente foi implementado com um *Web Service* para ficar de acordo com a interface do *Retailer*. Como há apenas um *Retailer* para cada *WebClient*, não existe a decisão de ter mais de um ponto de acesso. Para simplificação, ele foi desenvolvido como sendo um serviço dentro da própria aplicação da Infra-Estrutura.

7.5.2.2 Adaptador *Subscriber*

A implementação do Adaptador *Subscriber* foi realizada apenas uma vez, pois existe apenas um *Retailer* a ser chamado. O Adaptador *Subscriber* também foi desenvolvido como sendo um serviço dentro da própria aplicação da Infra-Estrutura.

7.5.2.3 Controlador de Respostas

O controlador de respostas é bastante simples, contendo apenas um local para que os dados retornados sejam armazenados até o momento de serem enviados para o *WebClient*.

7.5.3 Configuração

As configurações necessárias estão descritas na Tabela 11, conforme os itens descritos na Seção 6.2.3.

Tabela 15 - Configurações necessárias para o estudo de caso WS-I SCM *Retailer*.

<table>
<tr><th>Item</th><th>Descrição da configuração realizada</th></tr>
<tr><td>1</td><td>O sistema possui um arquivo de configuração, que pode ser facilmente alterado para que as chamadas sejam redirecionadas para a Infra-Estrutura.</td></tr>
<tr><td>2</td><td>Os testes foram executados localmente utilizando o usuário administrador do sistema, sem qualquer problema de permissão.</td></tr>
<tr><td>3</td><td>Foi criado o Topic "Retailer.SubmitOrder" para as mensagens geradas a partir de uma chamada ao método submitOrder do WebClient.</td></tr>
<tr><td>4</td><td>A assinatura criada foi:
<table>
<tr><td>Topic</td><td>Assinante</td></tr>
<tr><td>Retailer.SubmitOrder</td><td>Adaptador Subscriber Retailer</td></tr>
</table></td></tr>
<tr><td>5</td><td>O Topic criado para as mensagens de resposta foi o "Retailer.submitOrderResponse"</td></tr>
<tr><td>6</td><td>A assinatura criada para a resposta foi:
<table>
<tr><td>Topic</td><td>Assinante</td></tr>
<tr><td>Retailer.submitOrderResponse</td><td>Adaptador Controlador Respostas Retailer</td></tr>
</table></td></tr>
<tr><td>7</td><td>Não foi necessário criar nenhum filtro, pois o formato das mensagens é o mesmo.</td></tr>
</table>

7.6 Testes de Desempenho

Para avaliar a Infra-Estrutura em funcionamento, foram executados alguns testes simulando a utilização do sistema. A ferramenta utilizada foi a ACT [††] (*Microsoft Application Center Test*). A ACT é uma ferramenta para testes de performance em aplicações Web, capaz de simular diversos usuários interagindo com a aplicação simultaneamente. Durante a execução de um teste, essa ferramenta é capaz de capturar diversas informações da aplicação e do ambiente, apresentando, ao final dos testes, o resumo das informações coletadas. Ela possui, também, componentes que guiam à criação dos *scripts* de testes, facilitando o seu uso.

7.6.1 Contadores

Os contadores são variáveis registradas no sistema operacional, utilizadas para monitorar uma aplicação. Podem fornecer diversas informações como, por exemplo, medidas de desempenho da aplicação. Para este trabalho o padrão de contador utilizado fornece basicamente três informações:

1. Total de transações;
2. Tempo médio de cada transação;
3. Número de transações por segundo.

Serão medidas as performances das requisições exatamente no ponto onde as chamadas para os sistemas servidores são realizadas, e que serão direcionadas para a Infra-Estrutura. Isso é feito para diminuir os efeitos de fatores externos nos resultados dos testes. Os *counters* foram instalados para monitorar as seguintes transações:

1. ***WebClient submitOrder***: a transação começa quando o *WebClient* chama o método *submitOrder* do *Retailer* e finaliza quando os dados são retornados;
2. ***Retailer ShipGoods***: a transação começa quando o *Retailer* chama a *WarehouseA* e finaliza quando recebe a resposta da *WarehouseC*;
3. ***Warehouse submitPO***: a transação começa quando uma *Warehouse* chama o método *submitPO* de uma *Manufacturer* e termina quando o recebimento é confirmado;

7.6.2 Hardware

As configurações de Hardware foram:

1. Pentium 4;
2. 256Mb RAMBus;

[††] http://msdn.microsoft.com/library/en-us/act/htm/actml_main.asp

3. HD de 40 Gb.

É importante destacar a limitação do ambiente de testes, ou seja, a possibilidade da utilização de uma única máquina influenciar no resultado final do experimento.

7.6.3 Descrição dos testes

Um script que simula usuários realizando compras ao sistema *WebClient* foi criado e executado com configurações diferentes. As configurações que distinguem os testes são:

1. Infra-Estrutura: utilizada ou não;
2. Conexões simultâneas: informam quantos utilizadores foram simulados em paralelo;
3. Número de operações total: representa o número total de chamadas.

Devido a limitações do número de conexões em simultâneo que o IIS (*Internet Information Services*) possui para a licença que acompanha o Windows XP, o número máximo de usuários utilizados para os testes foi três.

Variando essas configurações, os testes de cada estudo de caso foram agrupados em dois conjuntos: o primeiro com apenas uma conexão, realizando um total de 100 operações, e o segundo com três conexões em simultâneo, realizando 100 operações cada e totalizando 300 operações. A apresentação dos testes também foi agrupada conforme o número de operações e o número de conexões em simultâneo utilizadas.

7.6.4 Estudo de Caso: WS-I SCM Warehouse

Para este estudo de caso, as *Warehouse* foram configuradas de maneira que apenas a *WarehouseC* tivesse condições de satisfazer os pedidos dos clientes. Logo, como o sistema *Retailer* chama em seqüência as *Warehouse*, as três requisições sempre são realizadas para os testes sem a Infra-Estrutura. A Infra-Estrutura, por trabalhar com eventos, sempre chama todas as *Warehouses* para, depois, selecionar qual a melhor resposta. A Tabela 16 apresenta as configurações e o tempo total revelado nos testes.

Tabela 16 - Configurações do ACT para cada teste realizado.

Teste	**1**	**2**	**3**	**4**
Infra-Estrutura	Não	Sim	Não	Sim
Conexões simultâneas	1	1	3	3
Número total de operações	100	100	300	300
Duração do Teste	**00:00:57**	**00:01:39**	**00:01:54**	**00:02:15**

A Figura 42 compara os resultados dos testes de desempenho 1 e 2, isto é, compara, para apenas uma conexão, a diferença de desempenho produzido pelo uso ou não da Infra-Estrutura. A Média de Requisições da Figura 42 (eixo vertical) representa a variação da média de chamadas

por segundo ao longo dos testes e o Tempo (eixo horizontal) representa o tempo decorrido dos testes (em segundos).

Pode-se verificar que o Teste 1 (sem a Infra-Estrutura) possui um número de requisições por segundo maior que o Teste 2, porém a utilização da Infra-Estrutura faz com que o sistema possua um desempenho mais constante, sofrendo menos oscilações.

A Figura 43 compara da mesma forma os teste 3 e 4. Os resultados são semelhantes à comparação entre os testes 1 e 2, ou seja, o desempenho em média é melhor sem utilizar a Infra-Estrutura, porém o número de requisições por segundo acaba oscilando mais, sendo o desempenho da aplicação mais inconstante.

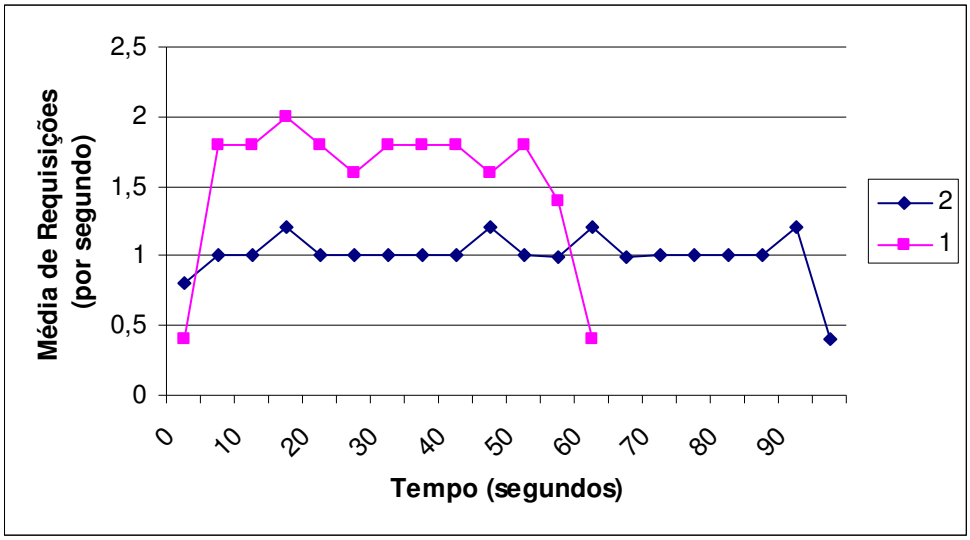

Figura 42 - Comparação dos testes 1 e 2 (uma conexão).

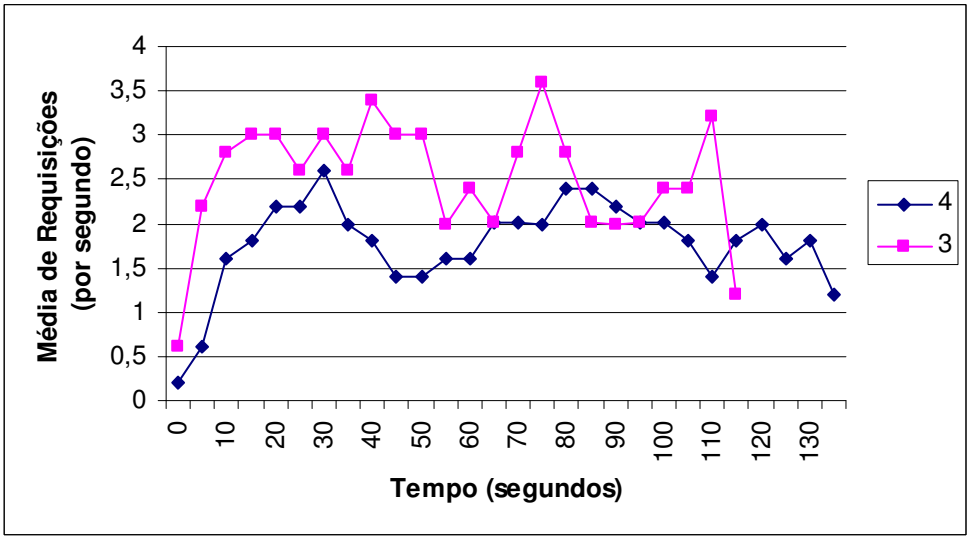

Figura 43 - Comparação dos testes 3 e 4 (três conexões).

O fato de os testes com a Infra-Estrutura serem mais constates é justificado pelo tempo de parada para a seleção da resposta na comunicação entre o *Retailer* e a *Warehouse*. É possível

observar isso, pois o valor de requisições por segundo do Teste 1 ficou próximo a uma requisição por segundo, ou seja, próximo do tempo escolhido no critério de parada para a seleção de respostas.

A Tabela 17 apresenta as taxas de desempenho entre os testes agrupados pelo número de conexões utilizadas e pelo número de operações realizadas. A taxa é calculada com base na proporção da diferença dos tempos coletados, sendo considerado o tempo de execução com a Infra-Estrutura como 100%.

Tabela 17 - Diferença de tempo utilizado para finalizar os testes.

Conexões Simultâneas	**Fórmula**		**Cálculo**		
1 conexão (100 operações)	100*	Teste 2 - Teste1 / Teste2	Taxa =100*	00:01:39 - 00:00:57 / 00:01:39	=42,42%
3 conexões (300 operações)	100*	Teste 4 - Teste3 / Teste4	Taxa =100*	00:02:15 - 00:01:54 / 00:02:15	=15,55%

O Teste 1 demorou 42,42% a menos que o Teste 2. O Teste 3 demorou 15,55% a menos que o Teste 4, ou seja, existem evidências de que há uma aproximação do desempenho com e sem a Infra-Estrutura provocada pelo aumento da demanda por recursos.

Vale salientar que todos os sistemas estavam instalados em um único servidor. Logo, quando a demanda por recursos aumentou, o desempenho de todos os sistemas foi prejudicado.

7.6.5 Estudo de Caso: WS-I SCM Manufacturer

Para este estudo de caso, os estoques das *Warehouse* foram ajustados para que a cada requisição de compra seja verificada a necessidade de se repor os estoques. Assim ordens de fabricação são geradas e enviadas as *Manufacturer*.

Foram realizados quatro testes variando a utilização ou não da Infra-Estrutura, total de interações e o número de conexões concorrentes, como apresentados na Tabela 18.

Tabela 18 - Configurações do ACT para cada teste realizado.

Teste	**1**	**2**	**3**	**4**
Infra-Estrutura	Não	Sim	Não	Sim
Conexões Simultâneas	1	1	3	3
Número total de operações	100	100	300	300
Duração do Teste	**00:03:16**	**00:03:39**	**00:08:14**	**00:08:43**

Os testes foram aplicados de forma semelhante ao estudo de caso WS-I SCM Warehouse. A Figura 44 compara os resultados dos testes de desempenho 1 e 2 (apenas uma conexão). O eixo vertical apresenta a variação da média de chamadas por segundo ao longo dos testes, e o eixo horizontal apresenta o tempo (em segundos). A Figura 45, da mesma forma, apresenta os resultados para os testes 3 e 4.

Pode-se observar que, em ambas as comparações, os tempos de execução e o desempenho médio são bastante semelhantes, tanto nos testes com a Infra-Estrutura como sem ela.

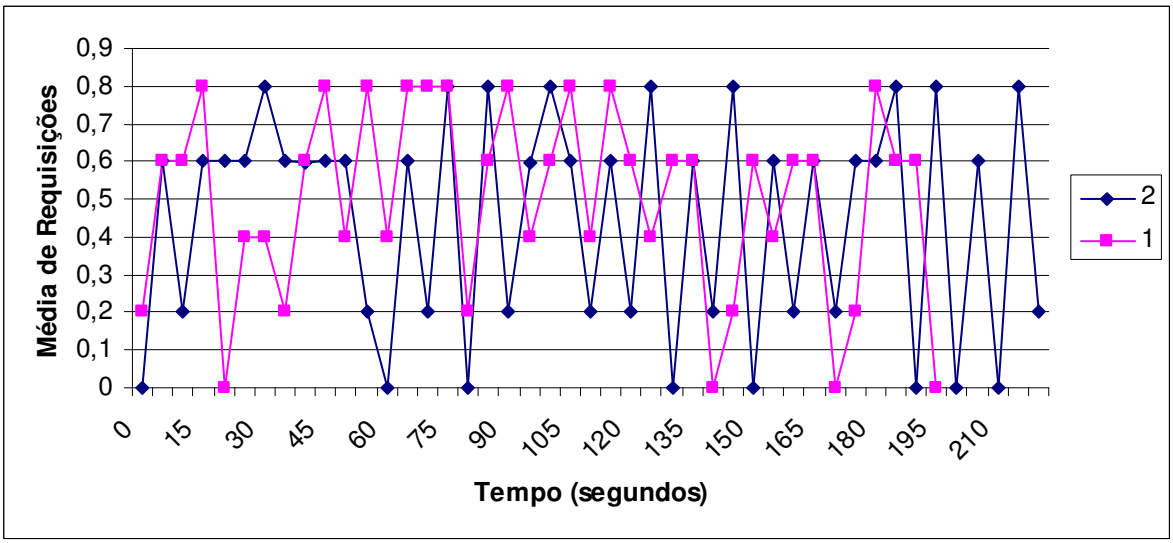

Figura 44 - Comparação dos testes 1 e 2 (uma conexão).

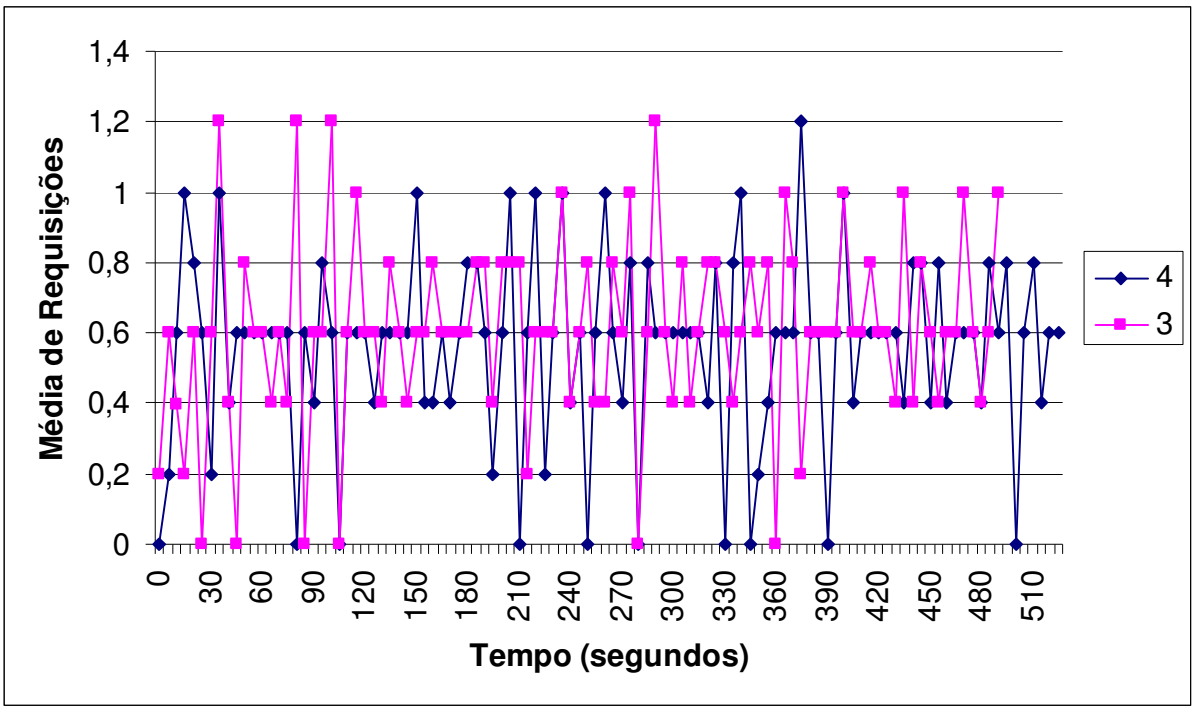

Figura 45 - Comparação dos testes 3 e 4 (três conexões).

Os valores com e sem a Infra-Estrutura são semelhantes, pois não existem mensagens de retorno e, assim, não existe a necessidade de esperar ou selecionar uma resposta.

Essa é a configuração que mais consome recursos da máquina, pois envolve o maior número de sistemas concorrentemente. Essa demanda por recursos acaba por aproximar o tempo

total de processamento dos testes, semelhante ao que já foi discutido com a utilização de várias conexões simultaneamente.

A Tabela 19 apresenta as taxas de desempenho entre os testes agrupados pelo número de conexões utilizadas e pelo número de operações realizadas, seguindo o mesmo formato da Tabela 17. A taxa é calculada com base na diferença dos tempos coletados, sendo considerado o tempo de execução com a Infra-Estrutura como 100%.

Tabela 19 - Diferença de tempo utilizado para finalizar os testes.

Conexões Simultâneas	**Fórmula**		**Cálculo**		
1 conexão (100 operações)	100*	Teste 2 - Teste1 / Teste2	Taxa = 100*	00:03:39- 00:03:16 / 00:03:39	=10,50%
3 conexões (300 operações)	100*	Teste 4 - Teste3 / Teste4	Taxa = 100*	00:08:43 - 00:08:14 / 00:08:43	=5,54%

O Teste 1 demorou 10,50% a menos que o Teste 2, sendo que essa relação diminuiu para 5,54% no caso dos testes 3 e 4, ou seja, pode-se observar novamente que existe uma aproximação do desempenho com e sem a Infra-Estrutura quando ocorre um aumento na demanda por recursos. Este resultado novamente pode ter a influência do uso de um único servidor.

7.6.6 Estudo de Caso: WS-I SCM Retailer

Para este estudo de caso, as *Warehouse* foram novamente configuradas como no estudo de caso WS-I SCM *Warehouse*, descrito na Seção 7.3, ou seja, não são enviados pedidos para as *Manufacturer*. Por ser a mesma configuração, os valores utilizados nos testes realizados sem a Infra-Estrutura foram os mesmos. Os dados estão apresentados na Tabela 20.

Tabela 20 - Configurações do ACT para cada teste realizado.

Teste	**1**	**2**	**3**	**4**
Infra-Estrutura	Não	Sim	Não	Sim
Conexões Simultâneas	1	1	3	3
Número total de operações	100	100	300	300
Duração do Teste	**00:00:57**	**00:02:03**	**00:01:54**	**00:02:12**

A Figura 46 apresenta a comparação entre os testes 1 e 2. Como nos casos anteriores, o eixo vertical apresenta o número médio de requisições por segundo e o eixo horizontal apresenta o tempo (em segundos). Pode-se observar como a utilização da Infra-Estrutura influenciou o desempenho dos sistemas. É possível visualizar que o tempo médio para realizar as operações no Teste 2 tende a ser constante no decorrer do tempo e o desempenho, sempre inferior ao desempenho sem a utilização da Infra-Estrutura (Teste 1).

A Figura 47 apresenta a comparação entre os testes 3 e 4. A sua descrição e o seu comportamento são semelhantes ao gráfico dos testes 1 e 2. A Tabela 21 mostra as taxas de desempenho entre os testes agrupados pelo número de conexões utilizadas e pelo número de operações realizadas. O Teste 1 demorou 53,65% a menos que o Teste 2. O Teste 3 demorou 15,78% a menos que o Teste 4. Novamente, existe uma aproximação do desempenho com e sem a Infra-Estrutura quando a demanda por recursos aumenta. Ressaltando-se a influência possível do uso de um único servidor.

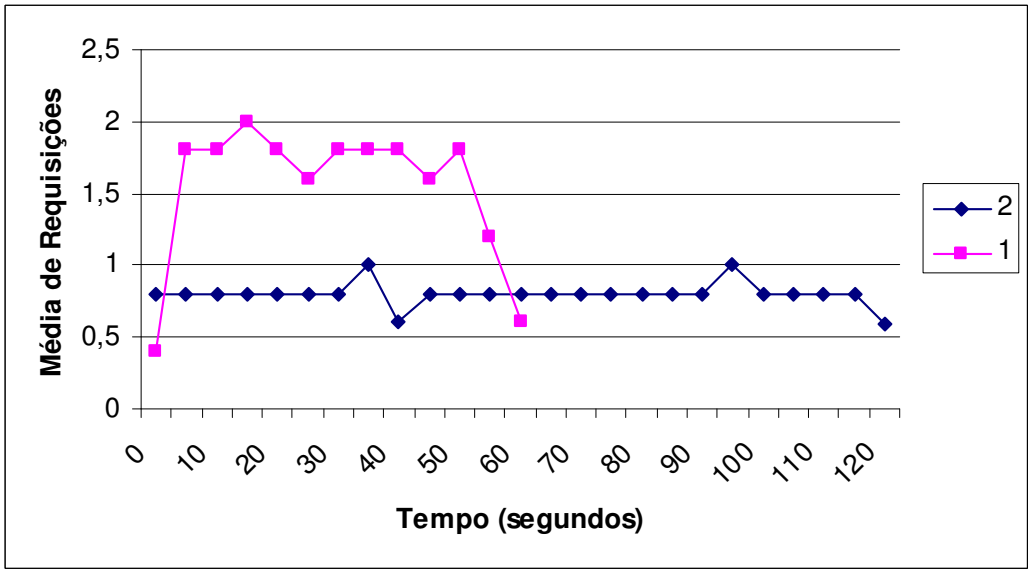

Figura 46 - Comparação dos testes 1 e 2 (uma conexão).

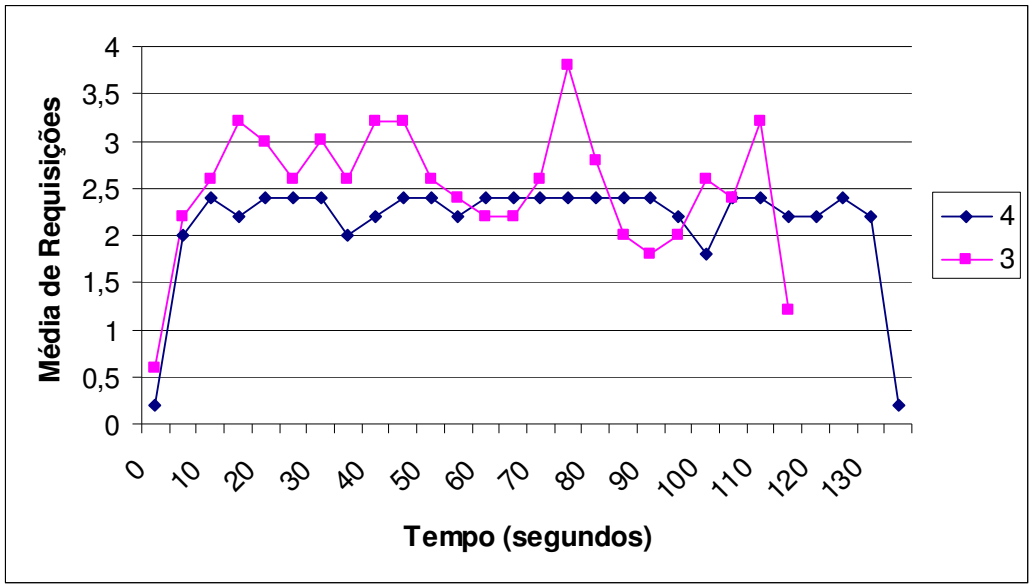

Figura 47 - Comparação dos testes 3 e 4 (três conexões).

Tabela 21 - Diferença de tempo utilizado para finalizar os testes.

Conexões Simultâneas	Fórmula		Cálculo		
1 conexão (100 operações)	100*	Teste 2 - Teste1 / Teste2	Taxa = 100*	00:02:03- 00:00:57 / 00:02:03	= 53,65 %
3 conexões (300 operações)	100*	Teste 4 - Teste3 / Teste4	Taxa = 100*	00:02:12 - 00:01:54 / 00:02:12	= 15,78 %

7.7 Considerações Finais

A aplicação da Infra-Estrutura em diversos estudos de caso provou ser viável, permitindo um grande conjunto de diferentes configurações para cada caso. Ainda com relação à implementação, foi confirmada a hipótese de ser possível retirar parte da lógica do negócio dos sistemas clientes e servidores para colocá-la em um intermediador sem causar impacto no funcionamento dos sistemas envolvidos.

Por conter diversos sistemas executando em paralelo, e estes conterem diversas linhas de execução, existe uma grande dificuldade para se saber o estado interno da Infra-Estrutura. Isso está de acordo com a documentação do padrão *Blackboard* [9] que sugere, como uma limitação do padrão, a dificuldade em realizar testes, visto que a resposta não é determinística.

Com relação aos testes de desempenho, foi possível observar que, conforme aumentam as solicitações, as taxas (diferenças da utilização ou não da Infra-Estrutura) diminuem. O tempo de espera por uma seleção de resposta é baixo e totalmente aceitável a uma aplicação Web para o usuário, demonstrando que é possível utilizar a Infra-Estrutura para a comunicação on-line sem maiores implicações para os sistemas adaptados. É importante ressaltar que os testes foram realizados com apenas um servidor.

Novos testes deveriam ser realizados para testar a hipótese de que instalando os sistemas em computadores diferentes e aumentando a carga dos testes, seria possível que o desempenho com a utilização da Infra-Estrutura se aproxime muito do desempenho sem ela.

8 CONCLUSÃO

Este trabalho buscou identificar e solucionar problemas comumente encontrados na integração de sistemas. O surgimento de padronizações com os *Web Services* vem ao encontro ao objetivo de auxiliar na integração. Porém, conforme o nível de abstração aumenta, é possível perceber novas limitações na comunicação entre sistemas heterogêneos.

O foco deste trabalho está na sistematização da integração de sistemas utilizando-se notificações. Assim, como demonstrado, a Infra-Estrutura de integração proposta atua como um intermediador, resolvendo diversos dos problemas existentes na integração de sistemas com Web *Services*, e possibilitando a utilização e gerência da lógica de negócio de uma maneira mais flexível, sendo esta retirada de dentro dos sistemas clientes e servidores.

Os padrões de projeto foram utilizados para a solução dos problemas de integração encontrados. A integração desses padrões forma a Infra-Estrutura, que é capaz de sistematizar a adaptação de sistemas solucionando os problemas apresentados na 5.1.

As funcionalidades providas pela Infra-Estrutura são:

1. Sistematização da integração através do padrão *Adapter*;
2. Gerência das regras de negócio;
3. Controle do fluxo de mensagens;
4. Gerência de configuração dos sistemas participantes.

A Infra-Estrutura permite que o projetista, além da solução dos problemas já mencionados, também substitua sistemas participantes das operações de forma transparente, tornando as integrações totalmente desacopladas. Como já mencionado anteriormente, a sua utilização está na definição e construção de três pontos, tornando simples a sua introdução em uma arquitetura:

1. Adaptação das interfaces dos Sistemas Clientes;
2. Adaptação das interfaces dos Sistemas Servidores;
3. Criação de um mediador com uma lógica de seleção de respostas.

Os estudos de caso mostraram que a Infra-Estrutura é genérica o suficiente para ser aplicada em mais de um domínio. Além disso, graças ao *NotificationBroker*, o controle da informação fica centralizado, facilitando a gerência dos dados trocados. A questão de escalabilidade fica relacionada à estrutura física em que a arquitetura está instalada, pois a utilização de mensagens permite que o fluxo de informação seja direcionado de forma a balancear melhor a carga nos servidores de aplicação.

A utilização do *framework* Microsoft .NET para implementação e testes dos casos de uso ajudou a minimizar o esforço realizado, pois ele já disponibiliza diversos serviços de monitoramento e testes, bem como um amplo suporte para o trabalho com XML, auxiliando na integração dos sistemas. Porém, a Infra-Estrutura pode ser desenvolvida em qualquer plataforma, pois não está ligada diretamente a nenhum serviço provido pelo *framework* Microsoft .NET.

8.1 Limitações

O trabalho desenvolvido apresenta as seguintes limitações:

1. A aplicação da Infra-Estrutura está limitada aos sistemas decomponíveis de programas e sistemas altamente decomponíveis;
2. As regras de negócio controladas estão restritas às que fazem parte da interação entre parceiros de negócio, não permitindo que regras internas dos sistemas sejam alteradas;
3. A Infra-Estrutura não possui suporte a transações, ou seja, a partir do momento em que uma mensagem é gerada, não se tem certeza de que um servidor irá recebê-la. Isso pode comprometer a seleção de respostas, caso não exista nenhuma a ser selecionada;
4. Não foi implementado nenhum mecanismo de segurança, sendo que os dados dos sistemas clientes e servidores ficam em uma área comum, podendo ser acessados por qualquer um dos sistemas que interagem com a Infra-Estrutura;
5. Os testes foram realizados em apenas um servidor, o que pode ter influenciado os resultados obtidos.

8.2 Trabalhos Futuros

Para a complementação do trabalho, podem ser considerados como possíveis trabalhos futuros:

1. Criação de uma ferramenta para o auxílio na aplicação da Infra-Estrutura;
2. Aplicação em sistemas de informação reais;
3. Testes de desempenho mais robustos, simulando um maior número de utilizadores;
4. Aplicação em sistemas de informação que não possuem interfaces do tipo WSDL;
5. Aplicação da Infra-Estrutura com outros padrões de negócios;
6. Mecanismos de suporte à transação e segurança.

9 REFERÊNCIAS

[1] ROY, J.; RAMANUJAN, A. "Understanding Web Services", IT Pro, vol. 3, issue 6, Nov. 2001, pp. 69-73.

[2] NEWCOMER, E. "Understanding Web Services: XML, WSDL, SOAP, and UDDI.", Addison-Wesley, 2002, 1st. ed., 368p.

[3] ALONSO, G.; CASATI, F.; KUNO, H.; MACHIRAJU, V. "Web services - Concepts, Architectures and Applications", Springer-Verlag, 2004, 374p.

[4] DEWES, E. S. R. "Um Estudo Comparativo entre Componentes e Web Services", Trabalho Individual I, Programa de Pós-Graduação em Ciência da Computação, PUCRS, 2003, 83p.

[5] BASS, L.; CLEMENTS, P.; KAZMAN, R. "Software Architecture in Practice", Addison Wesley, 1998, 560p.

[6] GARLAN, D.; SHAW, M. "An Introduction to Software Architecture". In: Advances in Software Engineering and Knowledge Engineering, World Scientific Publishing Company, v.1, Jan. 1994. Capturado em: "http://www-2.cs.cmu.edu/afs/cs/project /able/ftp/intro_softarch/intro_softarch.pdf" Oct. 2003, p.1-39.

[7] PERRY, D. E.; WOLF, A. L. "Foundations for the study of software architecture". ACM SIGSOFT Software Engineering Notes, vol. 17-4, Oct. 1992, pp.40-52,

[8] BROWN, A.; JOHNSTON, S.; KELLY, K. "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications", Tech. Rep., IBM, Oct. 2002. Capturado em: "www.rational.com/media/whitepapers/TP032.pdf", Aug. 2003, 16p.

[9] BUSCHMANN, F. et al. "Pattern-Oriented Software Architecture - a System of Patterns", John-Wiley and Sons, 1996, 476p.

[10] GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. "Design Patterns, Elements of Reusable Object-Oriented Software", Addison-Wesley, 1994, 395p.

[11] UMAR, A. "Application (Re)Engineering: Building Web-Based Applications and Dealing With Legacies". Prentice Hall Press, 1997, 624p.

[12] BRODIE, M. L.; STONEBRAKER, M. "Migrating Legacy Systems: Gateways, Interfaces & The Incremental Approach", Morgan Kaufmann, 1995, 207p.

[13] AALST, W. M. P. van der. "Don't go With The Flow: Web Services Composition Standards Exposed". IEEE Intelligent Systems, vol. 18, no. 1, Jan./Feb. 2003. Capturado em: "http://citeseer.nj.nec.com/vanderaalst03dont.html", Sept. 2003, pp. 72-76.

[14] ROY, J.; RAMANUJAN, A. "Xml Schema Language: Taking XML to The Next Level", IT Pro, vol. 3, no. 2, Mar./Apr. 2001, pp. 37-40.

[15] BOX, D. et al. "Simple Object Access Protocol (SOAP) 1.1". Tech. Rep., World Wide Web Consortium, May 2000.. Capturado em: "http://www.w3.org/TR/2000/NOTE-SOAP-20000508/", Sept. 2003. 30p.

[16] JEPSEN, T. "SOAP Cleans Up Interoperability Problems on The Web", IT Professional, vol. 3, no. 1, Jan./Feb. 2001, pp. 52-55.

[17] CURBERA, F.; DUFTLER, M.; KHALAF, R.; NAGY, W.; MUKHI, N.; WEERAWARANA, S. "Unraveling The Web Services Web. An Introduction to SOAP, WSDL and UDDI", IEEE Internet Computer, vol. 6, no. 2, Mar. 2002, pp 86-93.

[18] CHRISTENSEN, E.; CURBERA, F.; MEREDITH, G.; WEERAWARANA, S. "Web Services Description Language (WSDL) 1.1**"**, Tech. Rep., World Wide Web Consortium, Mar. 2001. Capturado em: "http://www.w3.org/TR/wsdl", Sept. 2003, 31p.

[19] CLÉMENT, L.; AL. et. "UDDI Technical White Paper", Tech. Rep., by Accenture, Ariba, Inc., Commerce One, Inc., Fujitsu Limited, Hewlett-Packard Company, i2 Technologies, Inc., Intel Corporation, International Business Machines Corporation, Microsoft Corporation, Oracle Corporation, SAP AG, Sun Microsystems, Inc., VeriSign, Inc., Sept. 2000. Capturado em: "http://www.uddi.org/pubs/ Iru_UDDI_Technical_White_Paper.pdf", Sept. 2003, 13p.

[20] FREMANTLE, P.; S.WEERAWARANA; KHALAF, R. "Enterprise Services", Communications of the ACM, vol. 45, no. 10, Oct. 2002, pp. 77-82.

[21] HAMMOND, J. "Introducing Web Services Into the Software Development Lifecycle", Tech. Rep., IBM**,** Oct. 2003. Capturado em: "http://www3.software.ibm.com /ibmdl/pub/software/rational/web/whitepapers/2003/TP033.pdf", Oct. 2003, 9p.

[22] CONALLEN, J. "Developing Applications For The Web Services Era", Tech. Rep., IBM, Oct. 2003. Capturado em: "http://www3.software.ibm.com/ibmdl/pub /software/rational/web/whitepapers/2003/TP031.pdf", Oct. 2003, 11p.

[23] BENATALLAH, B.; DUMAS, M.; FAUVET, M.-C.; RABHI, F. A.; SHENG, Q. Z. "Overview of Some Patterns For Architecting and Managing Composite Web Services", ACM SIGecom Exch. , vol. 3, no. 3, Aug. 2002, pp. 9-16.

[24] GRAHAM, S. et al. "Web Services Notification v1.0", Tech. Rep., Hewlett-Packard, IBM and SAP, Mar. 2004. Capturado em: "http://devresource.hp.com/drc/ specifications/wsrf/index.jsp", Apr. 2004, 68p.

[25] AOYAMA, M. "A Business-Driven Web Service Creation Methodology". In: IEEE Symposium on Applications and The Internet (Saint) Workshops, 2002. Capturado em: "http://www.seto.nanzan-u.ac.jp/~amikio/NISE/jp/papers/2002-01-WebSE-ServiceCreation.pdf ", Oct. 2003, 5p.

[26] ENDREI, M. et al. "Patterns: Service-Oriented Architecture and Web Services", IBM IBM Redbook, 2004. Capturado em: "http://www.redbooks.ibm.com/ abstracts/sg246303.html", May. 2004, 364p.

[27] SCHMIDT, D.; STAL, M.; ROHNERT, H.; BUSCHMANN, F. "Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects**"**, John Wiley Sons, 2000, vol. 2, 666p.

[28] ALUR, D.; CRUPI, J.; MALKS, D. "Core J2EE™ Patterns: Best Practices and Design Strategies**"**, Prentice Hall PTR, Jun. 2003, 2nd ed., 496p.

[29] BROWN, A. W.; SHORT, K. "On Components and Objects: The Foundations of Component-Based Development". In: Proceedings of the 5th International Symposium on Assessment of Software Tools (SAST '97), Jun. 1997. Capturado em: "http://www.cs.hut.fi/~saridaki/pdf/Brow97.pdf", Oct. 2003, pp 01-12.

[30] SZYPERSKI, C. "Component Software: Beyond Object-Oriented Programming", Addison-Wesley, 1999, 411p.

[31] BACHMAN, F. et al. "Volume II: Technical Concepts of Component-Based Software Engineering**"**, Tech. Rep., Carnegie Mellon University, 2000. Capturado em:

"http://www.sei.cmu.edu/publications/documents/00.reports/00tr008.html", Jun. 2003, 65p.

[32] HOPKINS, J. "Component Primer", Communications of the ACM, vol. 43-10, Oct. 2000, pp. 27-30.

[33] CUMMINS, Fred A. "Integração de Sistemas: EAI", Campus, 2002, 496p.

[34] SADEH-KONIECPOL, N.; HILDUM, D.; KJENSTAD, D. "Agent-Based e-Supply Chain Decision Support". In: Journal of Organizational Computing and Electronic Commerce, vol. 13, no. 3, Mar. 2003, 32p.

[35] WOHLIN, C.; RUNESON, P.; HOST, M.; OHLSSON, M. C.; REGNELL, B.; WESSLEN, A. "Experimentation in Software Engineering: an Introduction". Kluwer Academic Publishers, 2000, 147p.

[36] CHAPMAN, M.; GOODNER, M.; LUND, B.; MCKEE, B.; REKASIUS, R. "Supply Chain Management Sample Application Architecture**"**, WS-I, 2003. Capturado em: "http://www.ws-i.org/deliverables/Default.aspx", Dec. 2004, 37p.

[37] ANDERSON, S.; CHAPMAN, M.; GOODNER, M.; MACKINAW, P.; REKASIUS, R. "Supply Chain Management Use Case Model Document Status: Final Specification Version: 1.0", WS-I, 2003. Capturado em: "http://www.ws-i.org /deliverables/Default.aspx", Dec. 2004, 28p.