



FUNDAÇÃO EDSON QUEIROZ
UNIVERSIDADE DE FORTALEZA – UNIFOR

MISW – Uma Metodologia de Integração com Serviços Web

José Alécio Carvalho

MONOGRAFIA APRESENTADA AO CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DE FORTALEZA COMO PARTE DOS REQUISITOS PARA A
OBTENÇÃO DO GRAU DE BACHAREL EM INFORMÁTICA.

Aprovada por:

Profa Orientador. Elizabeth Sucupira Furtado, D.Sc. (UNIFOR)

Prof. Nabor das Chagas Mendonça, Ph.D. (UNIFOR)

Fortaleza, CE - Brasil

Dezembro / 2003

CARVALHO, JOSÉ ALÉCIO

MISW – Uma Metodologia de Integração com Serviços Web
[Fortaleza] 2003

vi, 63p. 29,7 cm (INFORMÁTICA/UNIFOR, Engenharia de Software, 2003)

Monografia - Universidade de Fortaleza, Informática

1. Serviços Web
2. Engenharia de Software
3. Padrões de Projeto
4. RUP – Rational Unified Process

I. INFORMÁTICA/UNIFOR II. TÍTULO (série)

“Uma viagem de mil milhas começa
com o primeiro passo.”

Lao-Tsé

Dedico este trabalho aos meus pais, José Alénisio
Carvalho e Francisca Rodrigues de Carvalho.

AGRADECIMENTOS

Pela oportunidade de crescimento pessoal, profissional e pelo desenvolvimento deste trabalho, meus sinceros agradecimentos a todas as pessoas que de certa forma colaboraram para a finalização desta etapa de minha vida profissional, em especial a:

Aos meus pais, minha irmã e irmãos, e familiares pelo apoio e ajuda que colaboraram com o meu crescimento pessoal e pelo suporte nos momentos difíceis.

À minha namorada, Sara, pela paciência e carinho que tem me dado e por estar sempre ao meu lado, nos momentos felizes e difíceis.

Aos meus amigos de colégio, Melo, Rodrigo, Tiago, Fernando, Júlio César, Ary e Gustavus, pelas lições de vida aprendidas.

Aos meus antigos e atuais colegas de trabalho, Franzé, Vinicius, Felipe, Bezerra, Yrleyjânder, Eric, Leo, pelo aprendizado e experiência profissional repassada.

À professora Elizabeth, orientadora desta monografia, pela confiança e pelas oportunidades que deram o “ponta-pé” inicial na minha carreira profissional.

RESUMO

O desenvolvimento para integrar sistemas com a utilização de serviços web tem tido bastante incentivo diante dos avanços tecnológicos das ferramentas e tecnologias que apóiam os serviços web. Contudo, projetar um sistema para integração com outros, requer bastante atenção na arquitetura projetada, e nos processos a serem executados. Portanto neste trabalho serão vistos conceitos usados ao se projetar sistemas para integração com serviços web, tais como padrões de projeto, em seguida, será descrita uma metodologia que guiará desenvolvedores nos diversos aspectos de uma integração. Por fim, será apresentado um estudo de caso para validar a metodologia proposta.

SUMÁRIO

1. Introdução	10
1.1. Visão Geral	10
1.2. Conceitos iniciais	11
1.2.1. O que são Serviços Web?	11
1.2.2. Interoperabilidade	12
1.3. Motivação	12
1.4. Objetivos	13
1.5. Estudo de caso	13
1.6. Estrutura da monografia	14
2. Serviços Web.....	15
2.1. Introdução.....	15
2.2. Arquitetura.....	15
2.3. Conceitos	18
2.3.1. SOAP	18
2.3.2. WSDL	20
2.3.3. UDDL.....	21
2.4. Java Web Services	22
2.4.1. Ferramenta: Apache Axis.....	23
2.5. Conclusão	25
3. Padrões de Projeto.....	26
3.1. Introdução.....	26
3.2. O que são Padrões de Projeto?	26
3.3. Padrões utilizados no estudo de caso.....	27

3.3.1. Data Access Object (DAO)	27
3.3.2. Façade	28
3.3.3. Service Locator	29
3.3.4. Model View Controller (MVC)	31
3.4. Conclusão	32
4. Uma Metodologia de Integração com Serviços Web.....	33
4.1. Introdução.....	33
4.2. MISW – Metodologia de Integração com Serviços Web.....	34
4.2.1. Definição	34
4.2.2. Principais conceitos	35
4.2.2.1. Atividades	35
4.2.2.2. Artefatos	36
4.2.2.3. Papéis	36
4.2.2.4. Fases	37
4.2.3. Estudo de Caso: Serviços Web de um sistema de EAD.....	42
4.2.3.1. Apresentação do Sistema de EAD: Fornecedor de serviços.....	42
4.2.3.2. Apresentação do Sistema: Utilizador de Serviços.....	42
4.2.3.3. Aplicando a MISW no estudo de caso.....	42
4.3. Conclusão	60
5. Conclusão	61
6. Bibliografia	62
6.1. Referências On-line	63

LISTA DE FIGURAS

Figura 1 - Modelo de funcionamento de um serviço web.	16
Figura 2 - Pilha básica de funcionamento de um serviço web.	17
Figura 3 - Como se dá a troca de mensagens utilizando o protocolo SOAP.....	20
Figura 4 – Funcionamento dos <i>Stubs</i> e <i>Skeletons</i>	24
Figura 5 - Funcionamento do DAO	28
Figura 6 - Funcionamento do Façade	29
Figura 7 - Funcionamento do Service Locator	30
Figura 8 - Funcionamento do MVC	31
Figura 9 - Fases do Fornecedor de Serviços	38
Figura 10 - Fases do Utilizador de Serviços	40
Figura 11 - Diagrama de Classes dos objetos de negócio.	43
Figura 12 - Diagrama de Classes das classes DAO.....	44
Figura 13 - Utilização do <i>Façade</i> : SistemaFacade implementa Interface Sistema.	45
Figura 14 - Diagrama de seqüência para o método getListaCursos.....	46
Figura 15 - Esboço da arquitetura de funcionamento dos elementos e padrões envolvidos.....	46
Figura 16 - Implementação do método getListaCursos no SistemaFacade.	47
Figura 17 - Implementação da interface InterfaceSistema.java	48
Figura 18 - Implementação da classe Curso.java	48
Figura 19 - Implementação da classe CursoDAO.java	49
Figura 20 – Trecho de código da classe SistemaFacade.java	50
Figura 21 – Trecho do WSDL gerado pelo Axis para os serviços.....	51
Figura 22 - Serviços publicados no Registro de Serviços do Axis.....	52
Figura 23 - Esboço da arquitetura de integração.....	54
Figura 24 - Diagrama de classes do sistema utilizador	55
Figura 25 - Diagrama de seqüência para recuperar lista de cursos.....	56
Figura 26 – Trecho de código da classe ServicosCadinetFacade	57
Figura 27 – Trecho de código da classe FacadeUtilizador	58
Figura 28 – Visão Web da aplicação utilizadora.....	59
Figura 29 – Visão Swing (desktop) da aplicação utilizadora.	60

Capítulo 1

1. Introdução

1.1. Visão Geral

As propostas de integração de sistemas distribuídos têm amadurecido bastante nos últimos anos para suprir a necessidade de integração de serviços das mais diversas corporações. Com o advento dos Serviços Web (*Web Services*), a integração de sistemas tornou-se mais prática, havendo também a diminuição de esforço e de custos para viabilizá-la.

Grandes empresas da indústria de Tecnologia da Informação (T.I.) vêm desenvolvendo pesquisas e aplicando técnicas para melhor usufruir dos recursos oferecidos pelos Serviços Web, dentre estas técnicas está a aplicação de padrões para apoiar no desenvolvimento de software.

Ferramentas como o Axis, o ETTK, o Java WSDP, dão grande auxílio no desenvolvimento de serviços web e são frutos de pesquisas desenvolvidas por instituições como Apache, IBM e SUN, respectivamente. Pode-se citar também, a utilização de tecnologias como WSDL (*Web Service Description Language*), UDDI (*Universal Discovery Description & Integration*) e SOAP (*Simple Object Access Protocol*), as quais possibilitam o funcionamento dos Serviços Web.

Com o objetivo de propor uma forma para analisar e projetar a integração de sistemas com a utilização de serviços web, esta monografia apresentará uma estratégia metodológica para melhor seguir os passos do desenvolvimento de integração. Antes de descrevê-la, neste capítulo serão apresentados os Serviços Web, o escopo deste trabalho e conceitos necessários para a compreensão adequada.

1.2. Conceitos iniciais

1.2.1. O que são Serviços Web?

“Serviços web, genericamente falando, são serviços ofertados via web.”
[TJWST,2003].

Diversos autores divergem quanto às definições de Serviços Web, muitas delas contraditórias e somente revelam uma parte do que realmente vem a ser Serviços Web. De forma simplificada, pode-se dizer que Serviços Web é um padrão para integrar sistemas através da utilização de protocolos de Internet como HTTP. Esta definição é muito simples, não sendo específica o bastante, já que CORBA, por exemplo, também se encaixaria nessa definição. Um fato que diferencia os Serviços Web dos demais padrões de integração, é que estes se caracterizam por serem baseados em tecnologias neutras como os protocolos de Internet e da tecnologia XML.

A utilização de um padrão neutro, como XML, para descrever e utilizar os Serviços Web faz com que esta modalidade para integração de sistemas se torne mais prática, fácil e viável, pois é independente de qualquer linguagem ou plataforma. Enquanto os demais padrões para integração necessitam de uma representação específica para sua plataforma ou linguagem, ou seja, em código compilado, ou binário.

Investigando mais detalhes nos Serviços Web encontram-se outras tecnologias que em conjunto os constituem, como **SOAP**, **WSDL** e **UDDI**. Algumas destas tecnologias serão úteis no desenrolar desta monografia, portanto uma breve introdução a estas será feita no próximo capítulo.

1.2.2. Interoperabilidade

Em um cenário ideal, seria formidável que todos os sistemas fossem escritos em uma só linguagem, para uma única plataforma de software e para uma única plataforma de hardware. Mas durante vários anos, a tecnologia da informação conheceu a diversidade de tecnologias concorrentes, que no fundo cada uma delas trazem benefícios específicos para seus utilizadores.

Diante desta diversidade de sistemas e plataformas, o desafio é integrá-los. Para isto, nos últimos anos surgiram tecnologias capazes de integrar sistemas diferentes, tornando-os assim interoperáveis. Mesmo assim, com o surgimento de CORBA, DCOM, RMI e etc, suas interoperabilidades tinham imposições e restrições, como por exemplo, para utilizar CORBA, a plataforma teria que oferecer uma implementação CORBA.

Com o surgimento da proposta de Serviços Web, o conceito de interoperabilidade tornou-se mais notável e aplicável entre sistemas. Pois a interação agora seria feita baseada em XML, um padrão aberto em que todas as plataformas de software e de hardware seriam capazes de interpretá-lo, por ser basicamente um arquivo texto comum.

1.3. Motivação

A proposta dos Serviços Web tem a oferecer muitos benefícios, tanto na rapidez do desenvolvimento, como na redução de custos e prazos na integração de sistemas. Ela possibilita integração de sistemas de plataformas diferentes (.NET, J2EE, CORBA), o que a torna mais interoperável.

Tendo em vista o grande caminho que a tecnologia dos serviços web tem a percorrer e a crescer, é importante propor um método para que o desenvolvimento desses serviços seja seguido de uma forma padrão por seus desenvolvedores. Este método

encoraja a utilização de padrões de projeto na arquitetura do software de integração, o que pode ajudar a obter sucesso e a qualidade esperada.

1.4. Objetivos

Neste contexto de desenvolvimento de sistema, uma metodologia é um roteiro a ser seguido durante todo um processo de desenvolvimento de qualquer tipo de software e portanto fundamental para garantir o sucesso e a qualidade do processo e do produto final.

Tendo em vista a necessidade de organizar o processo de desenvolvimento de software para integração de sistemas com a utilização de serviços web, esta monografia tem como objetivos principais descrever uma metodologia a ser seguida para desenvolver sistemas com essa tecnologia e aplicá-la em um estudo de caso, implementando em JAVA. A metodologia proposta é chamada Metodologia de Integração com Serviços Web (MISW).

Durante o estudo de caso pretende-se mostrar o uso desta metodologia durante o processo de desenvolvimento de uma integração entre dois sistemas com a utilização dos serviços web. Como consequência, serão investigadas as problemáticas encontradas e soluções aplicadas durante este processo de integração.

1.5. Estudo de caso

No estudo de caso deste trabalho será apresentada a arquitetura de um sistema que provê serviços web (chamado fornecedor de serviços), bem como a arquitetura de um sistema que irá utilizá-los (chamado utilizador de serviços).

Para o sistema fornecedor de serviços, foi escolhido o sistema de Educação a Distância (EAD) produzido na UNIFOR, chama do Cadinet. Este sistema provê cursos em um ambiente virtual online através da Internet e será descrito em mais detalhes no capítulo 4. Como sistema utilizador destes serviços, foi criado um sistema qualquer, que

possui duas visões: uma visão web, e uma visão de aplicação desktop. Portanto, o desafio é modelar uma arquitetura que possibilite da melhor forma a flexibilidade na utilização dos serviços.

As etapas de desenvolvimento deste estudo serão guiadas pela MISW, a qual será apresentada e discutida no decorrer deste trabalho.

1.6. Estrutura da monografia

Esta monografia está dividida em 4 (quatro) capítulos que definem conceitos necessários para sua total compreensão.

No segundo capítulo, serão apresentados o conceito de Serviços Web, e as tecnologias que o compõem. Serão também apresentadas uma breve introdução sobre *Java Web Services* e ferramentas de apoio ao desenvolvimento de Serviços Web.

O terceiro capítulo se refere aos Padrões de Projeto, onde será fornecida uma introdução aos mesmos, bem como fornece uma explicação de alguns destes.

No quarto capítulo, serão apresentados a metodologia MISW, seus conceitos principais, e a sua aplicabilidade em um estudo de caso.

Capítulo 2

2. Serviços Web

2.1. Introdução

Essencialmente os Serviços Web necessitam para seu funcionamento, que algumas tarefas sejam executadas. Basicamente estas tarefas se resumem em:

- Criação do Serviço Web, com suas interfaces e métodos a serem invocados;
- Publicação do Serviço Web;
- Localização do Serviço Web;

Para melhor compreender a seqüência e obter um entendimento conceitual sobre o funcionamento e comportamento dos Serviços Web, serão apresentados neste capítulo os principais aspectos arquiteturais e componentes que constituem um Serviço Web, bem como as ferramentas de apoio a todo o processo de desenvolvimento de aplicações com essa tecnologia.

2.2. Arquitetura

O modelo de funcionamento dos Serviços Web é simples, das principais atividades descritas anteriormente, pode-se identificar neste modelo 3 (três) entidades básicas no processo, bem como suas respectivas operações. Na Figura 1, pode-se ver um modelo superficial de seu funcionamento destacando as entidades com seus relacionamentos.

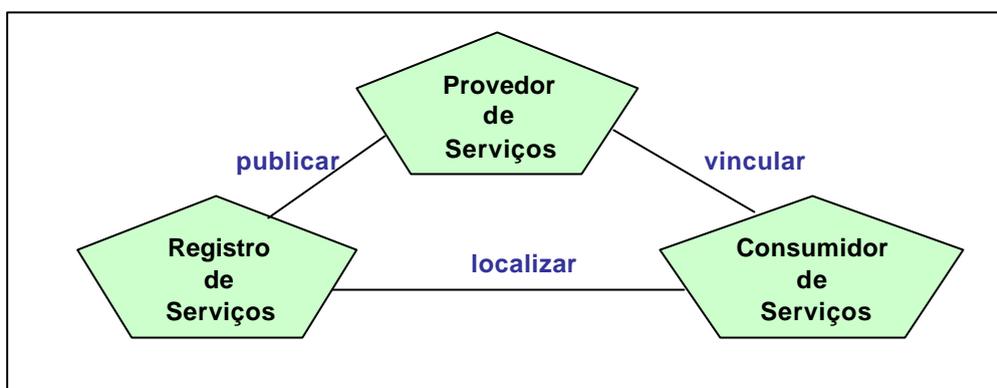


Figura 1 - Modelo de funcionamento de um serviço web.

Todo o processo inicia-se com a criação do serviço, atividade na qual o *Provedor de Serviços* é encarregado. Logo após, ocorre a criação de um documento que descreve o novo serviço no formato XML, chamado WSDL (*Web Service Description Language*), que iremos falar mais adiante. Após a criação deste documento, o serviço é então publicado em um *Registro de Serviços*, que por sua vez é a entidade responsável pela manutenção e armazenamento de registros de serviços disponíveis e de suas respectivas localizações. Surge então uma terceira entidade de nome *Consumidor de Serviços*, que utilizará os serviços, mas antes disso solicitará a localização de determinado serviço ao *Registro de Serviços*. O Registro de Serviços será responsável por efetuar um concreto entre o serviço web requerido e *Consumidor de Serviços*, conseqüentemente a interação entre estes.

O documento WSDL é um elemento fundamental para que o processo de interação entre os serviços web se inicie. É neste documento onde são descritas informações como a localização, o protocolo de rede utilizado para a comunicação, a definição dos métodos e tipos de dados e estruturas utilizadas pelo serviço. Este documento pode ser publicado de diversas formas, seja ele publicado em um UDDI (*Universal Description, Discovery, and Integration*), que mantém um registro central de serviços, ou submetido ao e-mail do consumidor do serviço, por exemplo. O objetivo de todo esse processo, é conseguir fazer com que o *Consumidor de Serviços*, tenha acesso ao WSDL dos serviços ofertados pelo *Provedor de Serviços*, para que o vínculo seja efetivado. O próximo passo deste processo

é iniciar a invocação remota de métodos através de um protocolo de troca de mensagens em XML, naturalmente feita através do SOAP. Nas próximas seções haverá uma breve descrição sobre estes componentes da arquitetura de Serviços Web.

Na estrutura de um sistema de serviços web, é possível identificar as camadas básicas para seu funcionamento. A Figura 2 mostra essas camadas na forma de pilha, a chamada pilha básica de serviços web, onde pode-se identificar as tecnologias que estão sendo usadas em cada camada da pilha. As camadas são as seguintes:

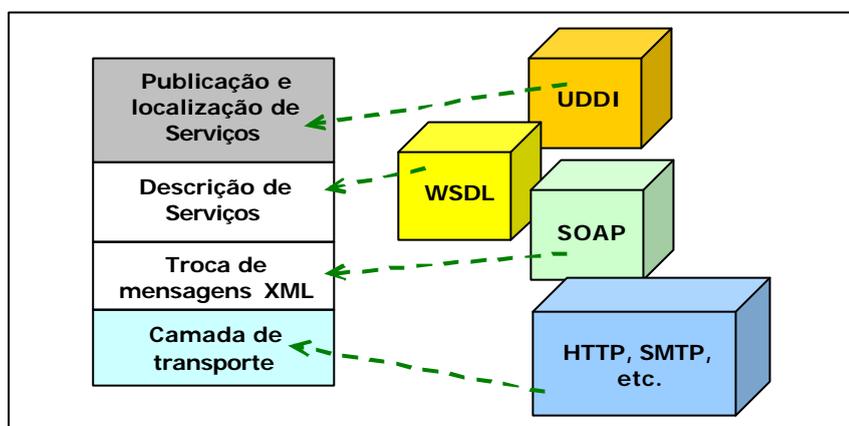


Figura 2 - Pilha básica de funcionamento de um serviço web.

- **Camada de transporte:** é a camada que disponibiliza o serviço ao meio externo de comunicação. Podendo utilizar qualquer um dos protocolos de comunicação, sendo o mais comumente utilizado o protocolo HTTP.
- **Troca de mensagens XML:** é a camada responsável pela definição do formato da mensagem que será utilizado na interação entre as aplicações. O padrão que normalmente se utiliza nesta camada é o protocolo SOAP, que será mais discutido na próxima seção.
- **Descrição do serviço:** é a camada responsável pelo fornecimento de um mecanismo de descrição da funcionalidade a qual o serviço web proporciona. Nesta camada, pode-se encontrar o padrão de descrição de serviços web, o WSDL, ao qual segue a especificação XML para descrever

o serviço. Na próxima seção, este documento será descrito com mais detalhes.

- **Publicação e localização de serviços**: nesta camada reside um mecanismo que armazena de forma organizada os registros de localização dos serviços web. É aqui onde os serviços são publicados, e onde os clientes irão encontrar os serviços que procuram antes de utilizá-los. Uma tecnologia padrão para esta camada é o UDDI, que será abordado na próxima seção.

2.3. Conceitos

Nesta seção serão apresentados os principais conceitos que fazem parte da especificação da tecnologia de Serviços Web.

2.3.1. SOAP

SOAP (*Simple Object Access Protocol*), como o próprio nome já diz, é um protocolo simples que fornece uma maneira padrão para a troca de mensagens XML. Este protocolo é escrito em XML, o qual o faz independente de plataforma de hardware, sistema operacional ou linguagem de programação. Por ser baseado em XML, que é descrito em formato textual, é um grande diferencial que o SOAP tem diante de outras tecnologias como DCOM e CORBA, que utilizam formato binário para representar as operações de comunicação. A sua transmissão pode ser efetuada sobre protocolos como o HTTP, SMTP, etc. Comparado a outros protocolos de computação distribuída, ele possui menos recursos, o que o torna menos complexo.

Por apresentar características tão superficiais, uma mensagem SOAP pode facilmente atravessar *firewalls* comuns em *intranets* organizacionais, por não barrarem o fluxo de informações em formato texto. Isso de uma certa forma pode ser encarado como uma vantagem, mas também como uma desvantagem, pois há possibilidade de burlar o

esquema de segurança de uma rede corporativa. Devendo assim haver uma boa política de segurança que considere este problema para evitar maiores danos.

O SOAP possui uma convenção para representar invocação remota de procedimentos, o chamado RPC (*Remote Procedure Call*) em uma mensagem SOAP, fornecendo suporte à troca de mensagens XML.

Uma mensagem SOAP basicamente contém 3 (três) partes principais:

- **Envelope SOAP**

Define uma maneira para descrever a composição da mensagem e como ela deve ser processada e interpretada. Um envelope SOAP contém um cabeçalho opcional e um corpo SOAP obrigatório.

- **Cabeçalho SOAP**

É um elemento opcional e pode ser efetivamente utilizado pelos usuários para fornecer informações adicionais. Este elemento pode ser utilizado para passar informações para autenticação, passando o nome de usuário e uma senha, por exemplo.

- **Corpo SOAP**

É a parte onde está toda a carga útil da mensagem, onde as chamadas e as respostas de procedimento são codificadas, e onde são representadas as falhas ocorridas na invocação.

Cada invocação remota é representada em mensagens SOAP, como também o resultado da invocação. A Figura 3 mostra como funciona o processo de troca de mensagens SOAP. Inicialmente a *Aplicação Consumidora* deseja enviar uma mensagem SOAP para a *Aplicação Fornecedora*, e esta mensagem pode simbolizar uma invocação remota de procedimento. Uma vez criada a mensagem SOAP de requisição, esta é inserida a um protocolo de rede, que então a submete para o destinatário. No outro ponto de comunicação, o processo inverso é efetuado, a mensagem SOAP, que é recebida no protocolo de rede, é encaminhada para a *Aplicação Fornecedora* que a processa e gera uma mensagem SOAP de resposta para a solicitação.

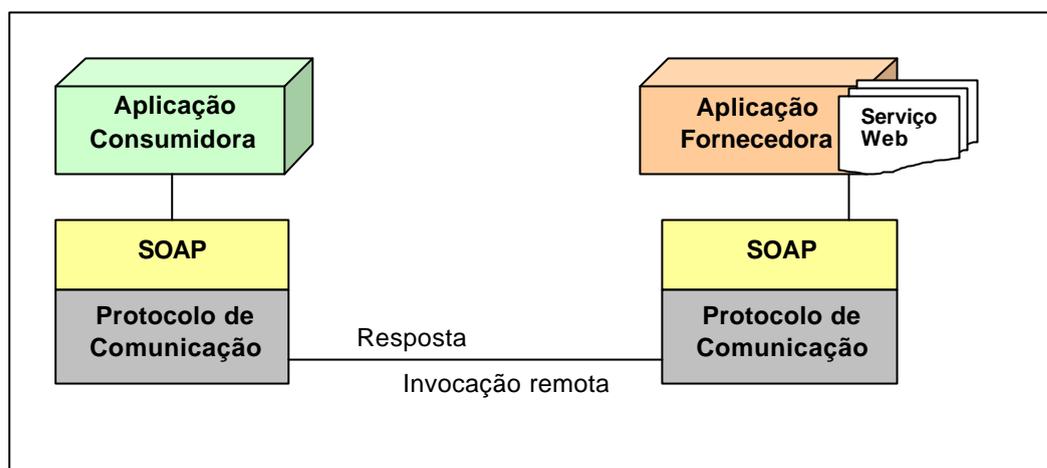


Figura 3 - Como se dá a troca de mensagens utilizando o protocolo SOAP.

2.3.2. WSDL

Elaborado através de um esforço conjunto da Microsoft, IBM e Ariba, e logo após submetida a W3C para padronização, o documento WSDL é um padrão para descrição de um serviço web baseado em XML. O documento WSDL apresenta uma descrição geral da interface para comunicação com serviço web, de maneira a descrever suas operações, parâmetros de entrada e tipos de retorno, e uma forma de como interagir com o serviço, fornecendo a localização do serviço, os tipos de dados e o protocolo de Internet utilizado. Na verdade, um documento WSDL é para um serviço web, o que IDL¹ é para o CORBA. Este documento será útil no momento de reconhecimento do serviço por parte de quem utilizará os serviços. Não se pode esquecer que este documento é escrito em XML, sendo assim independente de qualquer arquitetura ou plataforma de software.

Atualmente existem ferramentas capazes de gerar automaticamente o documento WSDL. O toolkit de ferramentas fornecido pelo Apache Axis tanto automatiza essa tarefa como também faz o processo inverso.

¹ IDL (*Interface Definition Language*) – Linguagem de definição da interface de implementação de objetos CORBA.

Os principais elementos utilizados em um documento WSDL para a descrição do serviço, são os seguintes:

- Elemento *<definitions>* → elemento raiz do documento ao qual contém todos os outros elementos. Nele são definidos atributos que serão visíveis no resto do documento, como declarações de XML *namespaceing*.
- Elemento *<import>* → permite que as declarações do WSDL não estejam contidas em um só arquivo, podendo haver uma reutilização de definições já feitas.
- Elemento *<types>* → elemento que contém a definição dos tipos de dados aos quais o serviço utiliza.
- Elemento *<message>* → define o tipo de dados utilizado nas trocas de mensagens, tanto as de requisição (parâmetros) quanto a resposta (tipo de retorno).
- Elemento *<portType>* → elemento que contém declarações de operações, funciona como a definição de qual tipo de classe responderá as requisições.
- Elemento *<operation>* → elemento declarado dentro de um elemento *<portType>*, descreve a operação como os parâmetros e tipos de retorno (já definidos no elemento *<message>*). Este elemento equivale-se a definição de qual método daquela classe (*<portType>*) irá responder a requisição.
- Elemento *<binding>* → define o mecanismo usado pelo serviço para se comunicar, como a descrição de qual protocolo de comunicação o serviço utiliza.
- Elemento *<service>* → descreve como localizar o serviço, especificando o seu endereço.

2.3.3. UDDI

UDDI é a principal tecnologia utilizada como padrão para registro, descrição e localização de Serviços Web a partir de uma base de registro compartilhada entre diferentes clientes. O UDDI dispõe de duas funções principais: de publicação e de

pesquisa. A de publicação é usada pelos provedores de serviço para publicarem seus serviços, enquanto as de pesquisa são utilizadas por clientes (consumidores) dos serviços que necessitam saber onde localizá-los.

O conjunto de benefícios fornecidos pelo UDDI o torna muito útil para grandes corporações que desejam fazer de seus serviços web um negócio lucrativo na Internet, e acabam por disponibilizarem seus serviços em servidores de registros de serviços UDDI. Por motivos de escopo desta monografia, não serão descritos detalhes sobre esta tecnologia.

2.4. Java Web Services

A linguagem Java tem se fortalecido bastante nos últimos anos por oferecer portabilidade, flexibilidade, por trabalhar com o paradigma de orientação a objeto e por oferecer suporte uniforme para diversos tipos de aplicações (desktop, web, *handhelds* e celulares). Além disso, a plataforma Java trabalha bem com XML, fornecendo suporte a APIs para sua manipulação como o DOM (*Document Object Model*) e o SAX. O DOM processa um documento XML colocando todo este documento em memória e depois o manipula, isto pode tornar este processo de manipulação do XML mais simplificado e fácil por parte do desenvolvedor. Já o SAX (*Simple API for XML*), processa documentos XML com leitura sob demanda, não necessitando então inserir todo o documento em memória.

Java Web Services trabalha basicamente com as APIs JAXM (*Java API for XML Messaging*) e JAX-RPC (*Java API for XML-based RPC*). O JAXM é a API relacionada com o mecanismo de troca de mensagens SOAP através da rede. Já a API JAX-RPC está relacionada ao mecanismo que manipula mensagens SOAP como chamadas RPC (chamada remota de procedimento). No entanto, o mecanismo utilizado pelo JAX-RPC oferece maiores vantagens quanto à facilidade de uso.

Além dessas facilidades, ferramentas cada vez mais práticas e acessíveis vêm sendo desenvolvidas sobre a plataforma Java, proporcionando um ambiente ideal para o desenvolvimento de aplicações com Serviços Web.

Atualmente existem *toolkits* de ferramentas e APIs para facilitar e agilizar o desenvolvimento de serviços web em Java, um deles é o Java WSDP (*Java Web Services Development Package*), desenvolvido pela SUN e por membros da *Java Community*.

Existe também uma ferramenta disponibilizada pela IBM chamada ETTK (*Emerging Technologies ToolKit*), que oferece suporte ao desenvolvimento de serviços web, com ferramentas completas para facilitar o trabalho do desenvolvedor.

Outra ferramenta é o Apache Axis (*Apache eXtensible Interaction System*), fornecido pela Apache. Esta ferramenta foi escolhida para o desenvolvimento do estudo de caso desta monografia. Na próxima seção, o Axis será descrito em mais detalhes.

2.4.1. Ferramenta: Apache Axis

Apache Axis (*Apache eXtensible Interaction System*) é uma ferramenta da Apache o qual facilita o processo de implementação e a disponibilização de serviços web. Essencialmente o Apache Axis é uma implementação para manipular o SOAP, sendo a continuação de um projeto anterior de nome Apache SOAP, o qual sofreu alterações profundas em sua arquitetura e implementação para ser o que é hoje. O Apache Axis é uma ferramenta prática para se trabalhar com serviços web.

Algumas das principais modificações efetuadas no antigo Apache SOAP foi a troca do mecanismo de manipulação de documentos XML, o qual utilizava-se o DOM. Como melhoria, seu substituto, Apache Axis, trocou este mecanismo para utilizar o SAX, o que fez o Axis ganhar em performance.

Dentre os mecanismos em que Java trabalha com serviços web, o Apache Axis utiliza o modelo da JAX-RPC API, visto anteriormente. O Axis encapsula o modelo de

chamada remota baseada na utilização de *Stubs* e *Skeletons*, vastamente utilizados em sistemas distribuídos. Os *Stubs* e *Skeletons* são conhecidos como *client-side proxy object* e *server-side proxy object* respectivamente. O *stub* está localizado no cliente e encapsula para o cliente o mecanismo de chamada remota, fazendo-a parecer local, já o *skeleton* está localizado no servidor, e este faz o processo inverso, onde mascara para o servidor as chamadas que são oriundas da rede, fazendo-as parecerem chamadas locais. A Figura 4 ilustra o funcionamento dos *stubs* e *skeletons*, onde existe um objeto remoto no qual o cliente deseja acessar. Este objeto, por sua vez, baseia-se em uma interface remota, a qual define as operações que o objeto remoto oferecerá. Diante disso, o *skeleton* e o *stub* para este objeto serão também baseados na interface remota, o que leva o cliente pensar que *stub* seja o próprio objeto remoto. Dessa forma, o mecanismo de comunicação na rede é transparente para o cliente.

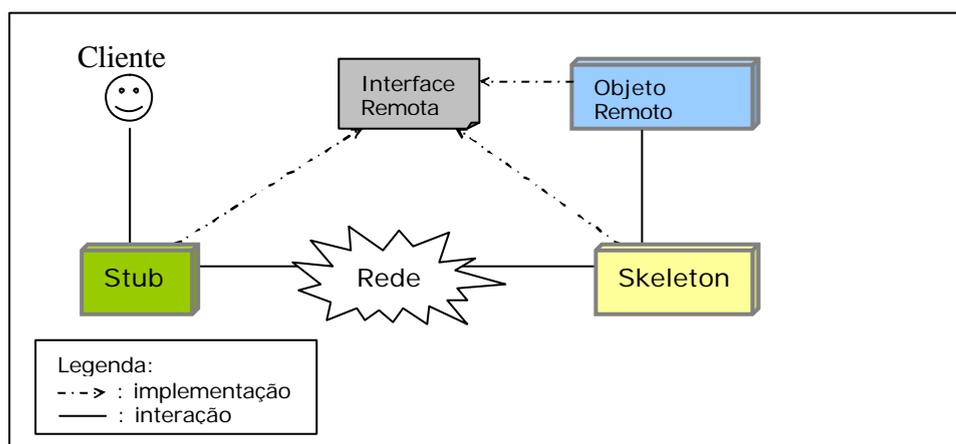


Figura 4 – Funcionamento dos *Stubs* e *Skeletons*

O Apache Axis possui um conjunto de ferramentas completas para se trabalhar com Serviços Web. Segue, logo abaixo, a lista de algumas tarefas e facilidades que o Axis oferece:

- Disponibilização “*on-the-fly*”, que fornece um meio de se publicar serviços de uma maneira mais fácil, bastando apenas que a classe que ofereça o serviço tenha a extensão de seu arquivo trocada de .java para .jws, com isso o serviço web já estará rodando tendo acesso ao WSDL e etc.

- Disponibilização de serviços de maneira formal, mecanismo fornecido pelo Axis para facilitar o *deploy* e o *undeploy* do serviço, descrevendo o mecanismo em um arquivo de extensão *.WSDD* (arquivos *Web Service Deployment Descriptor*).
- Utilitário para geração automática de documentos WSDL a partir de uma interface ou classe (o Axis criará uma interface) Java, o que facilita no desenvolvimento.
- Utilitário para geração automática de classes de controle de interação entre os pontos como classes *Stubs* e *Skeletons* baseados no WSDL já definido.
- Encapsula todo o mecanismo de interação entre a aplicação e o protocolo SOAP, fornecendo classes representativas que facilitam essa interação na geração de mensagens e na leitura de mensagens SOAP.
- Uma ferramenta para monitorar o fluxo de mensagens TCP/IP, possibilitando acompanhar a interação das mensagens SOAP.

2.5. Conclusão

Neste capítulo foram explorados os diversos conceitos que abrangem a tecnologia de Serviços Web, demonstrando o seu funcionamento e arquitetura. Os Serviços Web, apesar de fáceis de se utilizar e de se desenvolver, ainda é uma tecnologia considerada nova, muitos conceitos que envolvem segurança, transações, ainda estão surgindo. Mesmo assim, muitos apostam na consolidação desta tecnologia.

Capítulo 3

3. Padrões de Projeto

3.1. Introdução

Padrões de projeto têm sido vastamente aplicados e utilizados ultimamente. Eles descrevem um problema e uma solução, ou seja, especificam soluções para um problema específico. Padrões de projeto podem ser aplicados nos mais diversos níveis de projeto, com objetivos diversos como diminuir o acoplamento entre camadas, facilitar a migração de tecnologias, facilitar futuras modificações na aplicação, etc.

Em seguida, será fornecida uma definição mais ampla sobre o que vem a ser padrões de projeto e serão apresentados alguns destes padrões, focando sua aplicabilidade no desenvolvimento orientado a objetos e serviços web.

3.2. O que são Padrões de Projeto?

Projetistas descobriram que para problemas comuns no dia-dia em seus projetos, eles aplicam soluções parecidas. Quando estes projetistas identificam estas semelhanças e se abstraem, eles tentam formalizar de forma genericamente aplicável a sua solução para aquele problema identificado. Padrões de projeto é um conjunto de soluções para problemas comuns enfrentados em projetos de software, que se adapta a cada necessidade específica.

Tendo clara uma definição conceitual do que vem a ser padrões de projeto, esta monografia limitar-se-á a apresentar os padrões que serão utilizados no estudo de caso desta monografia. São eles: *DAO (Data Access Object)*, *Façade*, *Service Locator* e *MVC (Model View Controller)*. Maiores detalhes sobre este assunto podem ser encontrados em [J2EE,2003].

3.3. Padrões utilizados no estudo de caso

3.3.1. Data Access Object (DAO)

Este padrão encapsula a base de dados, provendo mecanismos para manutenção e pesquisa dos dados armazenados. Um dos principais aspectos deste padrão é sua maneira de abstrair e encapsular o acesso aos dados. Isto diminui o acoplamento entre a aplicação e o banco de dados, deixando o acesso transparente e não sendo específico a nenhum tipo de sistema de banco de dados.

Sem a utilização deste padrão, existe uma forte amarração entre o código da aplicação com o código para consultar e manter o banco de dados. Assim, a necessidade de modificação ou troca do sistema de armazenamento de dados pode afetar toda a aplicação. Portanto a utilização deste padrão traz alguns benefícios:

- Transparência quanto à fonte de dados, pois todo o mecanismo de localização e acesso da base de dados está encapsulado na camada DAO.
- O acesso aos dados é centralizado, todo o mecanismo de acesso à fonte de dados está descrito e encapsulado em uma só camada da aplicação.
- Facilidade de migração da base de dados, pois é uma consequência da separação e centralização da camada de acesso a dados.
- Redução da complexidade da implementação, proporcionando reuso do código, pois a definição de acesso às informações é definida uma vez, de forma genérica possibilitando a reutilização do que foi definido em diversas partes da aplicação.

Veja na Figura 5 o funcionamento deste padrão.

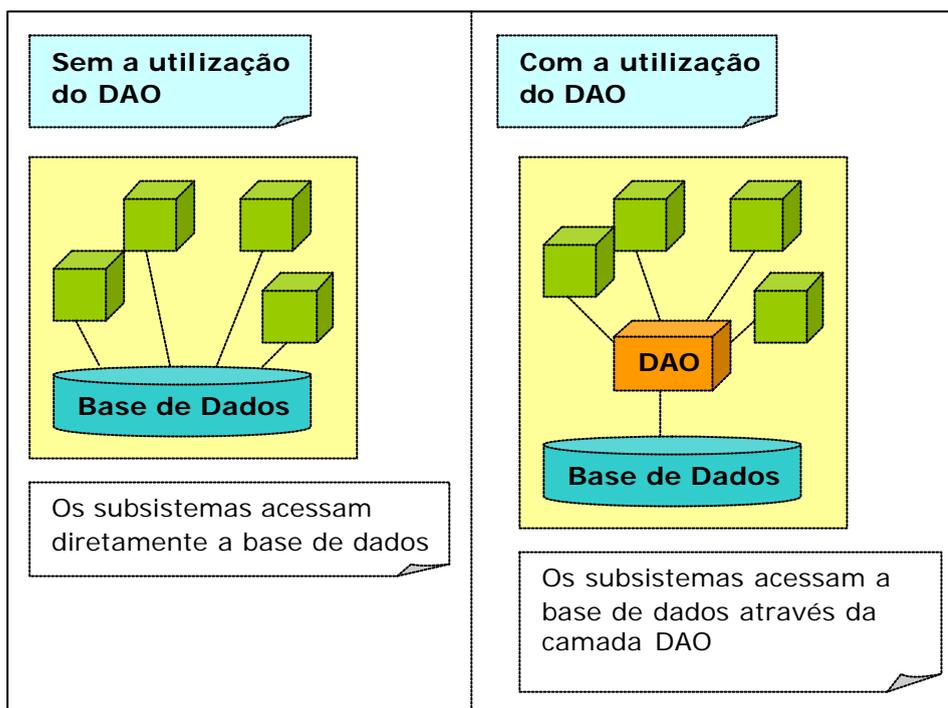


Figura 5 - Funcionamento do DAO

3.3.2. Façade

Este padrão encapsula a complexidade dos objetos de negócio da aplicação, fornecendo um mecanismo de acesso mais fácil e controlado às informações úteis. Em outras palavras, o padrão *façade* fornece uma interface única de acesso ao sistema, minimizando a complexidade para o cliente (utilizador).

Outro aspecto interessante sobre este padrão é que ele centraliza o fluxo de requisições efetuadas pelos clientes (utilizadores) no sistema, delegando as requisições aos subsistemas responsáveis pela execução da tarefa, servindo como uma interface de entrada para o sistema. (Ver Figura 6).

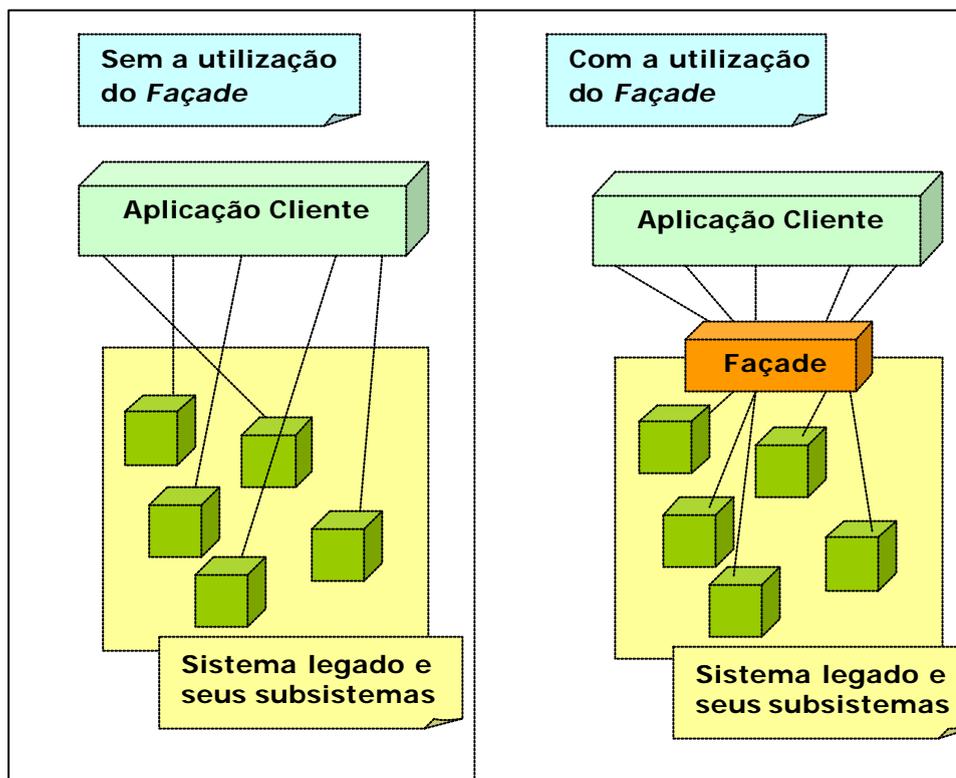


Figura 6 - Funcionamento do Façade

A utilização deste padrão traz benefícios, tanto em praticidade de acesso por parte dos clientes, como maior controle de acesso feito por estes. Isto se encaixa perfeitamente no modelo de solicitação de acesso aos serviços web, em que o fornecedor de serviços oferece uma interface de serviços disponíveis, aos quais o cliente poderá obter acesso.

3.3.3. Service Locator

O padrão *Service Locator* isola os componentes do cliente da complexidade de localização e criação de serviços. Geralmente para localizar e criar serviços é necessária a execução de uma série de passos seguindo uma lógica a qual torna-se redundante caso o

cliente possua vários componentes que acessam recursos (serviços) externos. Normalmente os serviços são disponibilizados em um registro de nomes, com seu nome ou chave de identificação. A Figura 7 mostra o funcionamento do padrão Service Locator. Pode-se perceber que sem a utilização deste padrão, cada componente do cliente que deseja interagir com um serviço ou recurso da aplicação servidora, necessita codificar a lógica para localizar o serviço desejado. Já com a utilização do padrão Service Locator, toda a lógica de localização de serviços está encapsulada em um só local, não sendo necessário que os vários componentes do cliente repliquem o código de acesso, bastando apenas solicitar o recurso desejado ao Service Locator e este se encarregará de encontrá-lo.

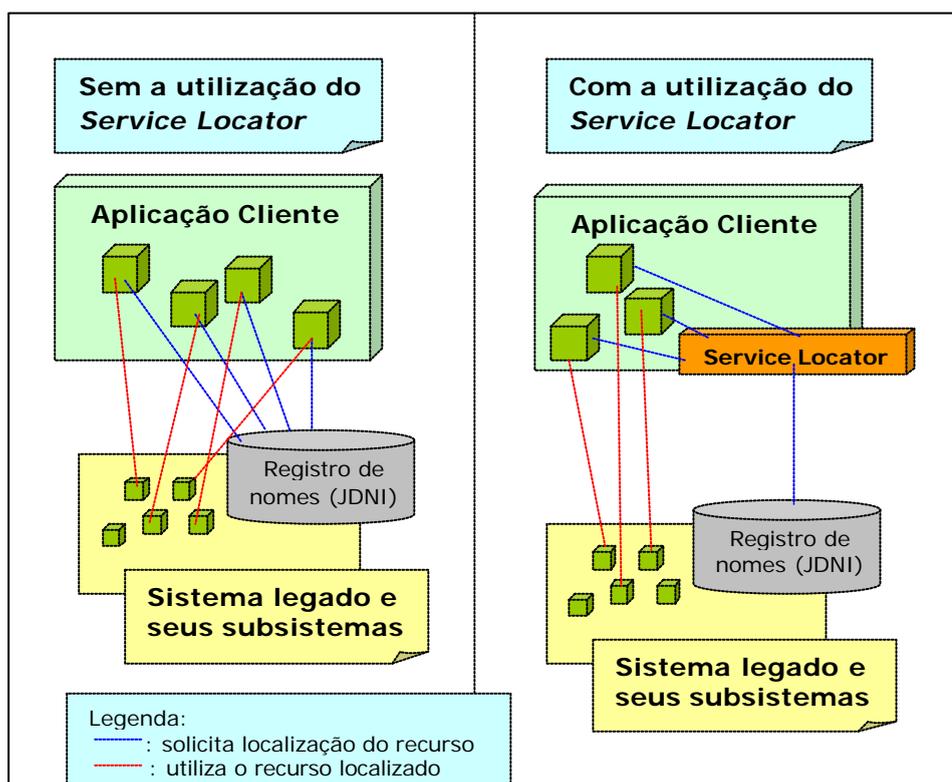


Figura 7 - Funcionamento do Service Locator

3.3.4. Model View Controller (MVC)

O padrão de arquitetura MVC encoraja a separação entre as camadas de apresentação e da lógica de negócio, a fim de evitar o acoplamento entre a aplicação e o dispositivo de visualização. MVC significa:

Model (Modelo) → representa a camada de negócio, que gerencia e armazena os dados.

View (Visão) → representa a interface da aplicação com o usuário.

Controller (Controlador) → responsável por coordenar as interações efetuadas entre a visão e o modelo.

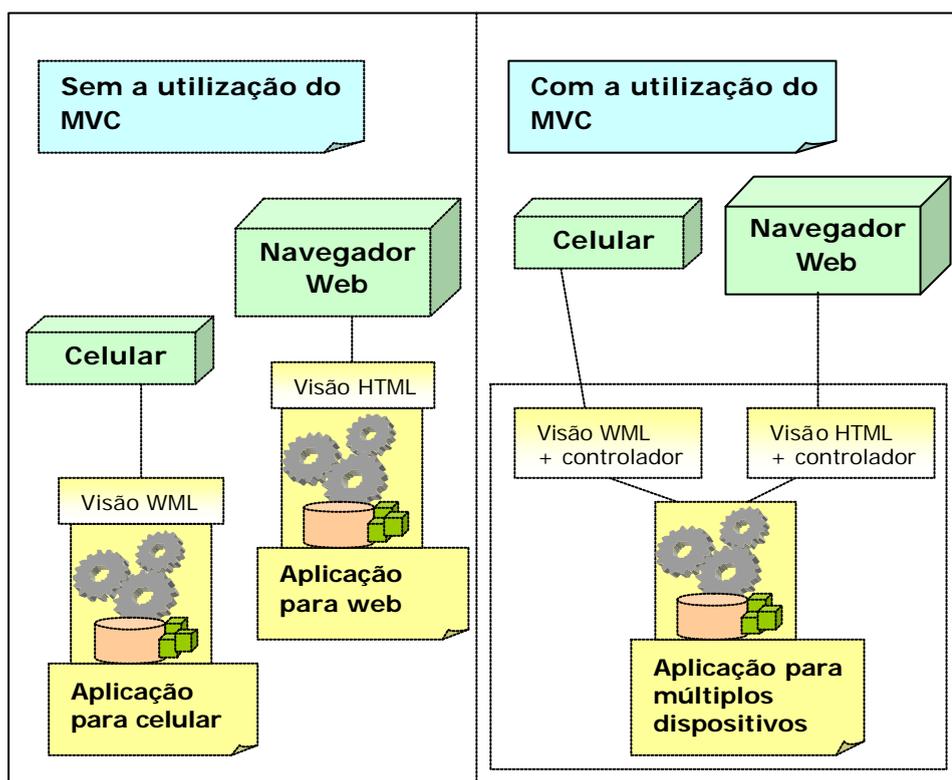


Figura 8 - Funcionamento do MVC

Com a utilização deste padrão, por exemplo, uma visão para dispositivos móveis (celulares, *Handhelds*) poderia ser facilmente adicionada a uma aplicação feita para web, sem precisar repetir todas as informações relativas à lógica da aplicação. (Ver Figura 8).

3.4. Conclusão

Neste capítulo foi apresentada uma visão geral sobre o que vem a ser padrões de projeto e sua importância na definição de uma arquitetura que atenda a todas as necessidades do sistema. A utilização de padrões de projeto é um fator fundamental para se obter escalabilidade, flexibilidade e robustez de um software.

Capítulo 4

4. Uma Metodologia de Integração com Serviços Web

4.1. Introdução

Desenvolver sistemas de software seguindo metodologias consiste em uma boa prática dos desenvolvedores de software.

No processo de desenvolvimento de sistemas distribuídos, a complexidade é inevitavelmente multiplicada. Atualmente existem diversos métodos para o desenvolvimento de sistemas, e sempre se encontra uma forma de encaixá-los em projetos de software que se desenvolve.

Quando se integra sistemas utilizando serviços web, dificuldades e peculiaridades são percebidas durante o processo, assim como em qualquer arquitetura distribuída. Mesmo com a agilidade e facilidade de uso que algumas ferramentas para desenvolvimento de serviços web oferecem, necessita-se que o processo de desenvolvimento de sistemas desta categoria seja melhor acompanhado. Com o objetivo de minimizar os riscos e auxiliar nesse processo de desenvolvimento, é preciso seguir uma metodologia específica para esse tipo de sistema, uma metodologia que contemple todos os aspectos requeridos, que auxilie nos dois lados da integração, ou seja, tanto do lado que fornece o serviço, quanto do lado que o utiliza. Para sistemas que utilizam serviços web como tecnologia de integração, se propõe uma metodologia que guiará seus desenvolvedores, definindo a melhor forma de se construir uma integração de sistemas com serviços web.

A metodologia MISW (Metodologia de Integração com Serviços Web), visa contemplar os diversos aspectos a serem destrinchados em um processo de integração com serviços web, agrupando as tarefas a serem executadas em etapas. O objetivo é a qualidade do processo, através de sua estrutura organizada em etapas que possibilitem a

qualquer período do processo, identificar a fase atual do sistema. Isto faz com que todos os participantes tenham uma visão uniforme do processo de desenvolvimento. Na próxima seção, serão definidas em mais detalhes esta metodologia e a forma como se propõe auxiliar no processo de integração de sistemas distribuídos.

4.2. MISW – Metodologia de Integração com Serviços Web

4.2.1. Definição

A MISW é uma metodologia de integração de sistemas que utilizam serviços web como tecnologia. Esta metodologia descreve uma forma de melhorar o processo e a qualidade do desenvolvimento em projetos de integração de sistemas distintos.

Esta metodologia se propõe em guiar os desenvolvedores durante todo o processo de desenvolvimento da integração de sistemas. Ela define alguns conceitos, tais como papéis, artefatos, atividades e fases, objetivando fornecer uma visão geral das tarefas dos profissionais envolvidos, o cumprimento de prazos e a redução de custos estabelecidos.

Uma característica original da MISW, é que esta define não somente a visão de desenvolvimento dos serviços (lado Fornecedor), mas também descreve e elabora as tarefas a serem executadas na visão de quem os utilizará (lado Utilizador de serviços). Isso faz com que esta metodologia seja de certa forma completa, pois abrange as duas visões de desenvolvimento.

Na próxima seção, serão apresentados os principais conceitos dessa metodologia e os itens que a compõem.

4.2.2. Principais conceitos

A MISW define em sua essência conceitos que serão úteis na aplicação de seu método. Alguns destes conceitos foram baseados no modelo RUP (Rational Unified Process) definido pela Rational, os quais foram inseridos na MISW em um contexto específico. O RUP define um processo de engenharia de software na construção de sistemas, organizando um conjunto de atividades bem elaboradas, definindo os responsáveis por cada atividade com artefatos de entrada e saída, ele segue uma seqüência lógica iterativa e incremental guiada a casos de uso, sendo fortemente ligado a UML. [KRUCHTEN, 2000].

A MISW envolve 4 (quatro) conceitos principais:

- Atividades (COMO)
- Artefatos (O QUÊ)
- Papéis (QUEM)
- Fases (QUANDO)

Em seguida, descreve-se com mais detalhes cada um destes conceitos e serão fornecidos exemplos práticos envolvendo cada um.

4.2.2.1. Atividades

Entende-se por atividade um conjunto de uma ou mais tarefas a serem executadas por um papel, podendo obter como resultado um ou mais artefatos produzidos.

São exemplos de atividades: definir serviços, implementar serviços, gerar interface de comunicação, publicar serviço web, documentar, projetar arquitetura, etc.

4.2.2.2. Artefatos

Artefato é o produto, o resultado da execução de uma determinada atividade. Como exemplo, pode-se citar, a atividade “gerar interface de comunicação”, que o documento WSDL é o produto.

4.2.2.3. Papéis

Entende-se por papel, a responsabilidade pela execução de uma determinada atividade, conseqüentemente sendo o construtor de um ou mais artefatos.

A MISW define alguns papéis padrão, podendo haver adição de novos papéis, caso haja necessidade durante o processo. Ao todo são 5 (cinco) papéis que se dividem na execução das atividades durante o processo de integração dos sistemas. Os papéis são:

- **Projetista de Integração:** é responsável pelo projeto de toda a arquitetura do sistema, fazendo a definição dos serviços e da interface de comunicação, do protocolo de comunicação e o detalhamento do funcionamento arquitetural da integração. O perfil recomendado para este papel é que tenha um bom conhecimento da arquitetura do sistema, com boa experiência em padrões de projeto.
- **Programador de Serviços:** é responsável pela concretização dos serviços, fazendo a implementação dos serviços e de todas as classes de apoio à interação entre os sistemas. O perfil recomendado para este papel é que tenha um bom conhecimento sobre o funcionamento dos serviços web e de computação distribuída.
- **Programador de Integração:** é responsável pela concretização da integração dos serviços, fazendo a implementação das classes que utilizarão os serviços web disponibilizados. O perfil recomendado para este papel é que tenha um bom conhecimento sobre o funcionamento dos serviços web e de computação distribuída.

- **Documentador:** é responsável pela especificação de uma documentação descritiva de cada serviço oferecido. O perfil recomendado para este papel é que tenha um bom conhecimento do negócio da aplicação.
- **Testador:** é responsável pela validação do sistema, apontando possíveis problemas, tanto de negócio, quanto funcional, e sugerindo melhorias na usabilidade. Geralmente os próprios programadores desenvolvem este papel, mas o ideal é que tenha disponível uma equipe específica para esta função, que siga de forma metódica e neutra a especificação de testes.

4.2.2.4. Fases

A MISW define suas fases levando em consideração os pontos de vista de desenvolvimento de uma integração, o lado fornecedor de serviços e o lado utilizador. Suas fases tentam de maneira macro categorizar e dividir as atividades da melhor forma durante o processo de integração.

A metodologia é dividida em 6 (seis) fases, sendo metade relativa à visão de desenvolvimento do fornecedor de serviços e a outra metade dedicada à visão do sistema utilizador. Em seguida, estas fases serão apresentadas e descritas.

1) Fases do ponto de vista: Fornecedor dos serviços

As fases a serem executadas pelo fornecedor de serviços concretizam o serviço em si. Elas descrevem e categorizam todo o processo de desenvolvimento dos serviços, os quais serão disponibilizados para utilização. Portanto, é de fundamental importância que todo o processo deste ponto de vista seja bem elaborado e arquitetado, pois o produto final deste processo fornecerá acesso aos serviços a partir de diversos outros clientes. As fases que compõem esta visão de desenvolvimento estão ilustradas na Figura 9.

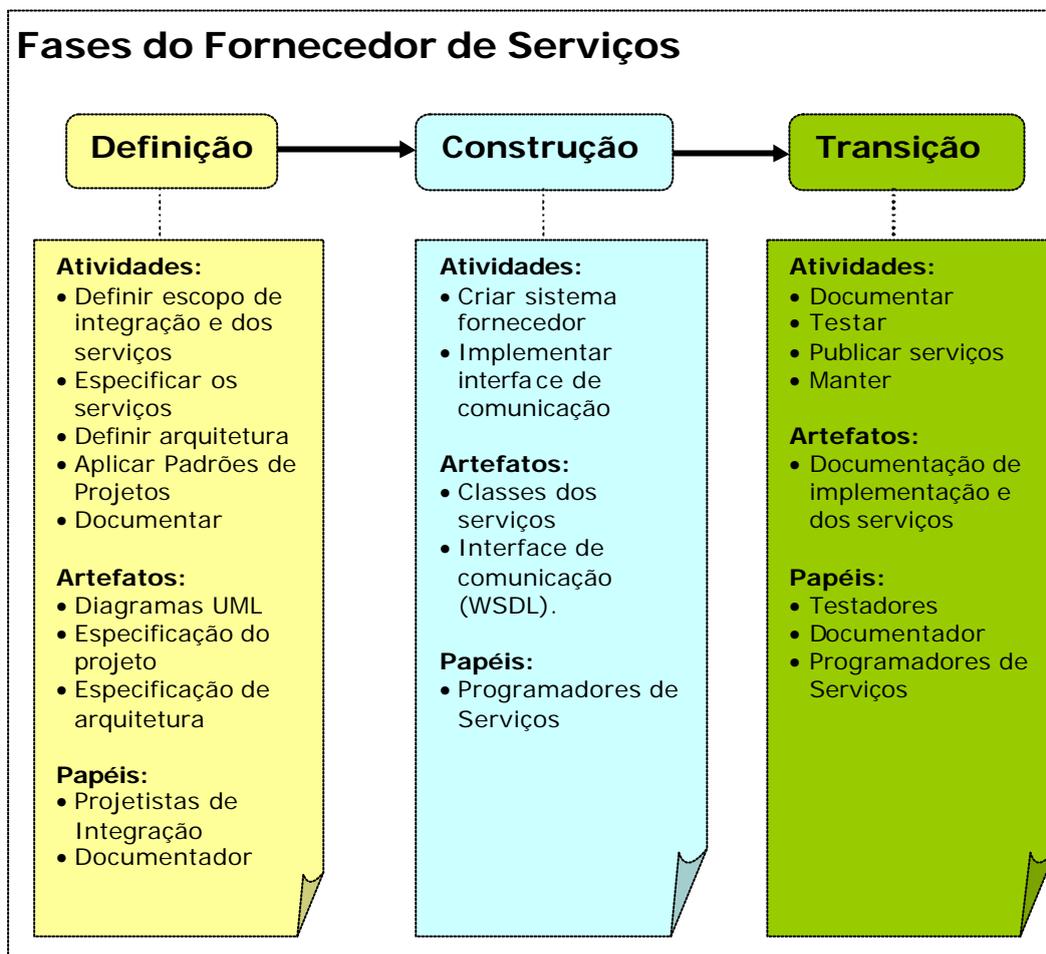


Figura 9 - Fases do Fornecedor de Serviços

I. Definição

A fase de definição é a fase inicial destinada à definição do escopo da integração, e dos serviços que se deseja disponibilizar. Nesta fase são identificados como se dará o acesso aos serviços, bem como o protocolo de comunicação e o mecanismo de integração dos serviços. Nesta fase, os Projetistas de Integração irão estabelecer uma arquitetura para a integração, definindo a forma na qual os serviços estarão disponíveis, como assinatura de métodos, parâmetros e tipos de retorno. Nesta arquitetura é fortemente recomendada a utilização padrões de projeto, de acordo com as necessidades identificadas.

Nesta fase, obtêm-se artefatos que descreverão todo o resultado da análise sobre a integração e a estratégia escolhida. Estes artefatos são, por exemplo o projeto da arquitetura proposta, como diagramas de casos de uso, de classe, de seqüência ou colaboração, etc.

A escolha de ferramentas de desenvolvimento, e softwares necessários também acontece nesta fase, bem como uma definição abstrata da interface de comunicação para os serviços ofertados.

II. Construção

Nesta fase os Programadores de Serviços trabalharão de forma efetiva na construção dos serviços de acordo com o especificado na fase de definição pelos projetistas. Será criada a parte que servirá de apoio aos serviços (base de dados, subsistemas, etc), e realizada a implementação concreta para a interface de comunicação. O objetivo é possibilitar a criação do sistema fornecedor de serviços e do documento WSDL, que descreverá a interface de comunicação de serviços.

III. Transição

Após a elaboração e implementação dos serviços, inicia-se a fase de transição, na qual se dará a disponibilização dos serviços. A disponibilização é efetuada publicando o documento WSDL produzido na fase anterior em um Registro de Serviços de forma que possibilite aos interessados, acesso aos serviços definidos. Nesta fase, realiza-se também uma documentação dos serviços, especificando suas funcionalidades, objetivos, etc. A documentação, possibilitará uma melhor utilização dos mesmos, que é de fundamental importância para o entendimento pelos utilizadores dos serviços.

2) Fases do ponto de vista: Utilizador dos serviços

As fases a serem executadas pelo utilizador de serviços são fundamentais para a efetivação da integração dos serviços. Elas descrevem e categorizam todo o processo de desenvolvimento da integração com os serviços de um determinado fornecedor, os quais foram previamente disponibilizados para utilização. As fases que compõem esta visão estão ilustradas na Figura 10 e são as seguintes:

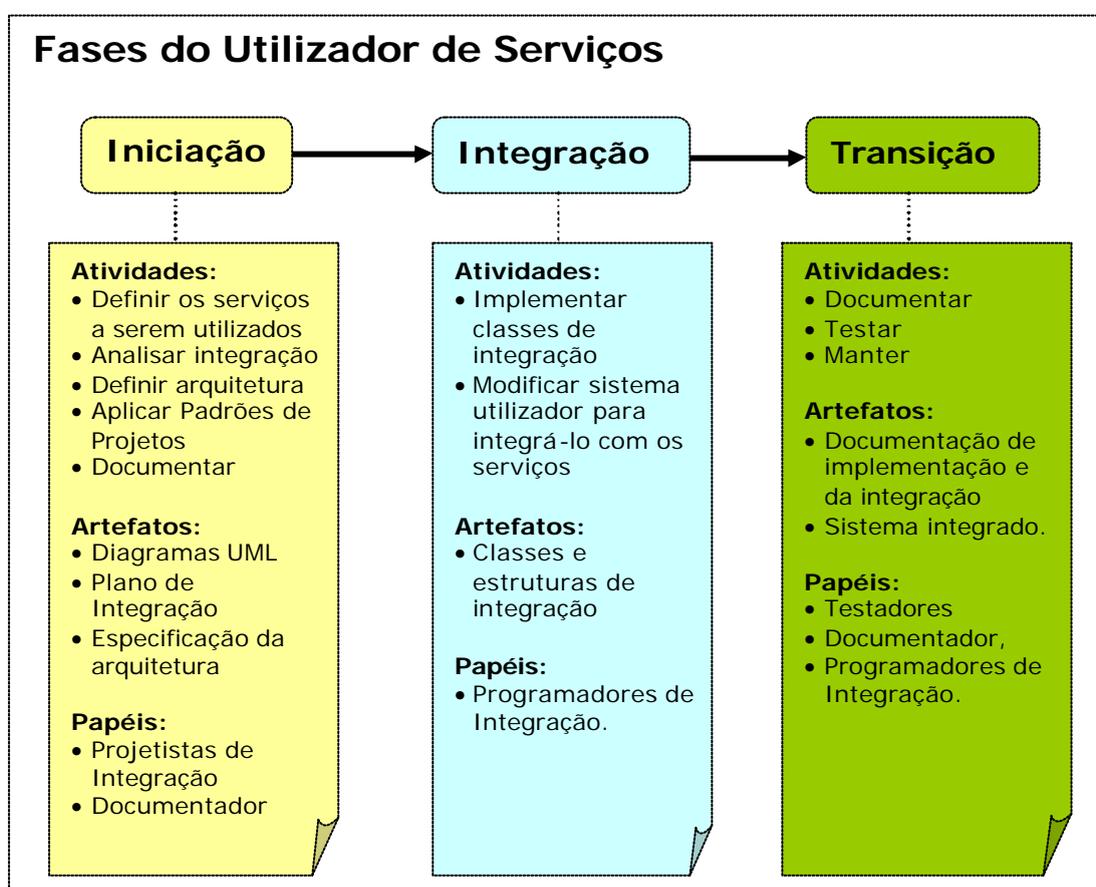


Figura 10 - Fases do Utilizador de Serviços

I. Iniciação

Esta fase inicia-se com a obtenção do documento WSDL através de um Registro de Serviços. A partir daí, escolhe-se os serviços a serem integrados e

então se elabora um plano para a integração. Este plano de integração é feito a partir da adequação dos serviços com o sistema que os utilizará.

Nesta fase os Projetistas de Integração definirão este plano de integração, projetando uma arquitetura que integrará o sistema atual com os serviços disponibilizados pelo fornecedor, definindo assim como o sistema irá utilizá-los. É fundamental que nesta arquitetura, sejam aplicados os padrões de projetos necessários para tornar o sistema mais flexível a mudanças.

II. Integração

Nesta fase se dá a concretização do que havia sido definido na fase anterior, isto é, a integração é implementada. Nela se implementa a arquitetura definida para se utilizar os serviços disponibilizados, implementando as classes necessárias para funcionamento da arquitetura proposta e gerando mecanismo de acesso aos serviços.

Nesta fase participam de forma efetiva os Programadores de Integração, que irão implementar as classes na arquitetura proposta para se utilizar os serviços.

III. Transição

Com a integração dos serviços concluída, esta fase enfoca em utilizá-los e em sua manutenção, ou adaptação. Nesta fase pode haver uma documentação da arquitetura definida para a integração, sendo útil em futuras modificações. Aplica-se também a implementação de testes para verificar se a integração foi bem sucedida, podendo verificar também a qualidade da comunicação e a velocidade de interação.

4.2.3. Estudo de Caso: Serviços Web de um sistema de EAD

4.2.3.1. Apresentação do Sistema de EAD: Fornecedor de serviços

O sistema fornecedor de serviços escolhido para se aplicar o estudo de caso, foi o sistema de EAD desenvolvido na UNIFOR, chamado Cadinet.

O Cadinet é o ambiente de educação a distância completo que tem como objetivo permitir a criação dos cursos pelo professor e acompanhamento de cursos pelos alunos. Estes cursos estão disponíveis através de uma estrutura em hipertexto, onde o professor poderá obter, construir e compartilhar uma série de documentações (artigos, textos, teses, trabalhos). A elaboração e compartilhamento dos cursos ocorrem através de interações assíncronas (fóruns, e-mails, grupos de estudo virtuais).

4.2.3.2. Apresentação do Sistema: Utilizador de Serviços

O sistema que utilizará os serviços é um sistema simples. Para demonstração, serão feitas para um sistema utilizador duas visões: uma visão web e outra em desktop padrão *Windows*.

4.2.3.3. Aplicando a MISW no estudo de caso

Nesta seção, a metodologia MISW proposta neste trabalho será seguida passo a passo objetivando mostrar a construção, disponibilização e utilização dos serviços web de EAD. Em seguida, a seqüência da elaboração do estudo de caso será exibida.

1) Elaboração dos serviços: Fornecedor de Serviços

I. Definição

Inicialmente foram definidos os serviços que seriam disponibilizados. Os serviços definidos foram os seguintes:

- Informações sobre os cursos ofertados;
- Informações sobre as Turmas do curso;
- Informações sobre as Disciplinas;
- Informação sobre as Unidades de cada Disciplina;
- Informações sobre os Temas de cada Unidade;
- Informação sobre Materiais Didáticos de cada Tema;
- Informação sobre Trabalhos de cada Tema;

Em seguida, os diagramas de classes dos objetos de negócio foram especificados, objetivando esboçar as entidades envolvidas. (Ver Figura 11).

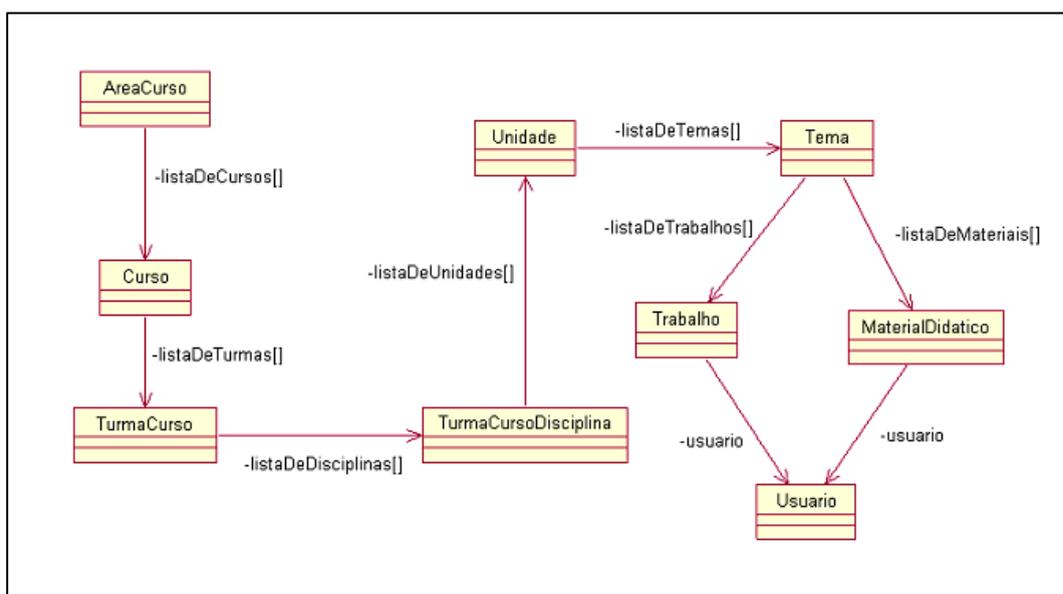


Figura 11 - Diagrama de Classes dos objetos de negócio.

Após a definição dos serviços e dos objetos de negócio envolvidos, serão definidas as classes da camada DAO. Como visto no capítulo anterior, o padrão de projeto DAO diminui o acoplamento entre a aplicação e o sistema de armazenamento de dados. Portanto ele será aplicado neste estudo de caso, devido a seus benefícios.

A idéia para se aplicar o padrão de projeto DAO, é que cada objeto de negócio deve ter um objeto DAO associado que será responsável pela manipulação de seus dados no sistema de banco de dados. A Figura 12 mostra o diagrama de classes para as classes DAO deste estudo de caso.

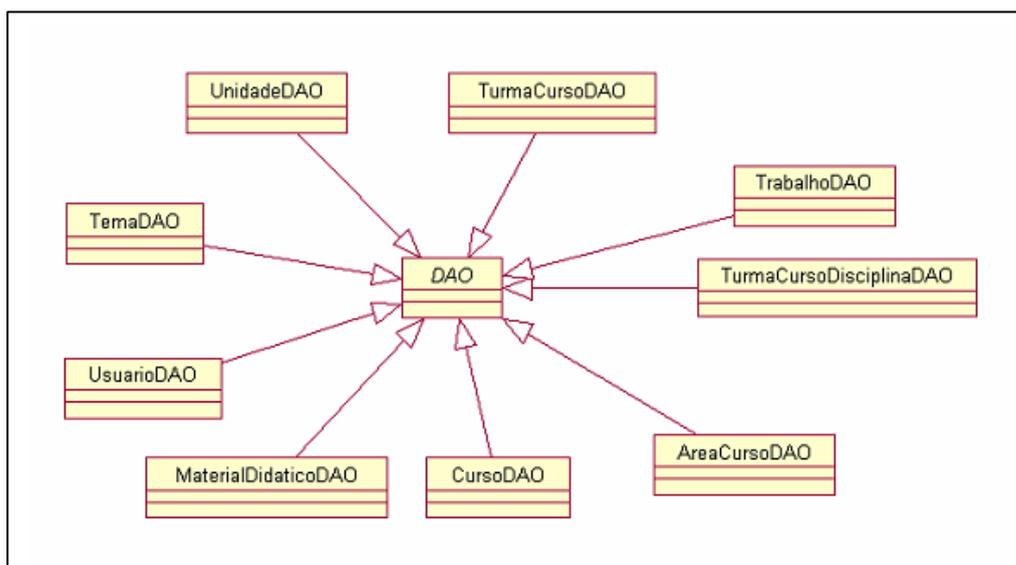


Figura 12 - Diagrama de Classes das classes DAO.

Após a definição das classes de apoio (objetos de negócio, objetos DAO) da aplicação, é hora de definir como será a arquitetura que proporcionará a disponibilização dos serviços requeridos.

A arquitetura do sistema para possibilitar a integração será baseada no modelo de padrão de projeto *Façade*, pois seu funcionamento se adapta perfeitamente com serviços web por expor todos os serviços da aplicação em forma de operações. Na arquitetura proposta para o sistema em estudo existirá uma classe com a funcionalidade *Façade*, a qual conterà todos os serviços ofertados via serviço web. Em outras palavras seria a interface de comunicação, ou de acesso aos serviços. No sistema a classe SistemaFacade tem a funcionalidade *Façade*, e os métodos contidos nesta classe estão

definidos na InterfaceSistema, que na verdade é uma *interface* (classe abstrata). Através dos métodos implementados, a classe SistemaFacade irá proporcionar acesso às funcionalidades do sistema, será a “porta” de acesso ao sistema.

O relacionamento entre “InterfaceSistema” e “SistemaFacade” está ilustrado na Figura 13. O “SistemaFacade” implementa todos os métodos definidos na “InterfaceSistema”. Tomando como exemplo o método getListasCursos(), na implementação deste método, haverá um acesso à classe DAO chamada CursoDAO, responsável pela manutenção dos cursos, para que esta classe informe a lista de cursos ofertados.

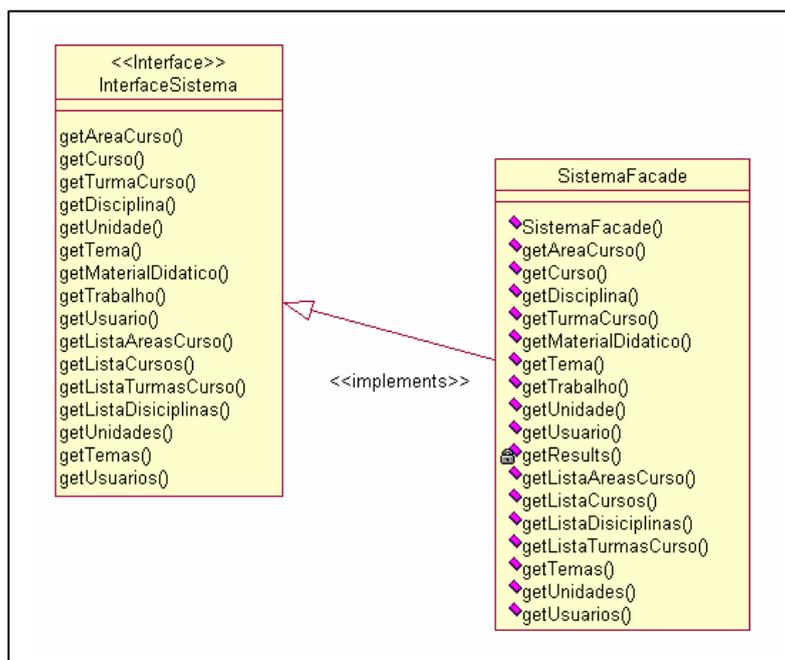


Figura 13 - Utilização do *Facade*: SistemaFacade implementa Interface Sistema.

O diagrama de seqüência associado à execução do método `getListasCursos` é mostrado na Figura 14. A aplicação cliente solicita, via serviços web, a lista de cursos. Esta solicitação é repassada para a classe `SistemaFacade` que solicitará à classe `CursoDAO` a listagem de cursos, os quais serão obtidos do banco de dados. O resultado então é encaminhado à aplicação cliente.

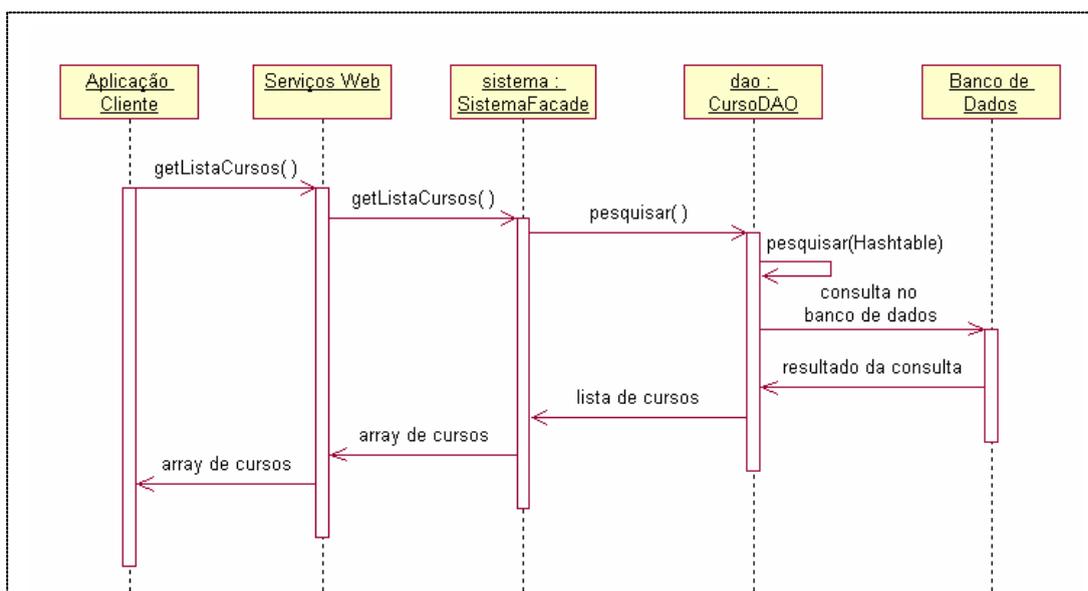


Figura 14 - Diagrama de seqüência para o método getListaCursos.

A etapa de Definição é finalizada com elaboração esquemática da arquitetura do sistema. A Figura 15 mostra o modelo de arquitetura para este estudo de caso:

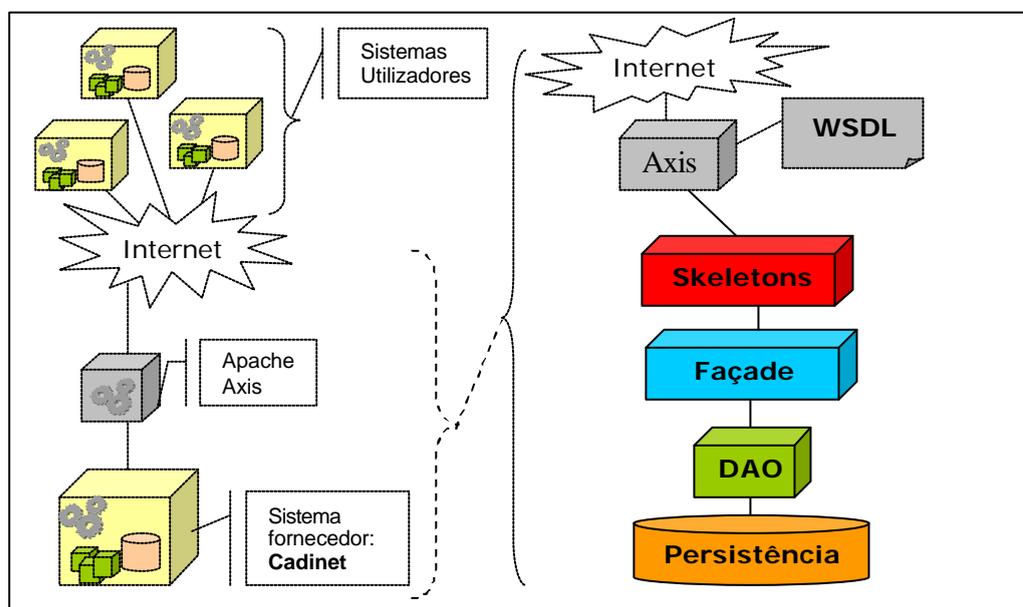


Figura 15 - Esboço da arquitetura de funcionamento dos elementos e padrões envolvidos.

II. Construção

Nesta fase, os programadores de serviços implementam a arquitetura definida na fase anterior, possibilitando a geração automática de um documento WSDL, com o auxílio de uma ferramenta de desenvolvimento.

Primeiramente, serão implementadas as classes de entidade, e as classes DAO para interação com a persistência do Cadinet. Por exemplo, para se obter informações sobre um curso, uma classe DAO para curso (CursoDAO) irá fazer uma consulta à base de dados do Cadinet, e irá retornar como resultados da consulta uma coleção de objetos do tipo Curso (classe de entidade). A Figura 16 ilustra como ficaria a implementação desta operação.

```
public Curso[] getListaCursos() {
    ArrayList lista = new CursoDAO().pesquisar();

    Curso[] cursos = new Curso[lista.size()];
    Iterator it = lista.iterator();
    int i=0;
    while(it.hasNext()){
        cursos[i++]=(Curso)it.next();
    }
    return cursos;
}
```

Figura 16 - Implementação do método getListaCursos no SistemaFacade.

Após a construção dessas classes, os programadores de serviços, implementam a interface abstrata de comunicação (InterfaceSistema) e a classe SistemaFacade, como definida na fase anterior. A Figura 17 ilustra a implementação da interface InterfaceSistema, onde podemos observar que somente são definidos os métodos (sem conteúdo).

```

package sistemacurso;

import sistemacurso.entidades.*;

public interface InterfaceSistema {
    public AreaCurso[] getListaAreasCurso();
    public AreaCurso getAreaCurso(int cd);
    public Curso[] getListaCursos();
    public Curso getCurso(int cd);
    public TurmaCurso getTurmaCurso(int cd);
    public TurmaCurso[] getListaTurmasCurso(int cdCurso);
    public TurmaCursoDisciplina getDisciplina(int cd);
    public TurmaCursoDisciplina[] getListaDisiciplinas();

    public Unidade getUnidade(int cd);
    public Unidade[] getUnidades(int cdTurmaCurso,int cdDisciplina);
    public Tema getTema(int cd);
    public Tema[] getTemas(int cdUnidade);
    public MaterialDidatico getMaterialDidatico(int cd);
    public Trabalho getTrabalho(int cd);
    public Usuario getUsuario(int cd);
    public Usuario[] getUsuarios();
}

```

Figura 17 - Implementação da interface InterfaceSistema.java

```

package sistemacurso.entidades;

public class Curso implements java.io.Serializable {
    private int cdCurso;
    private java.lang.String descricao;
    private java.util.Calendar dtCriacao;
    private java.lang.String nmCurso;

    public int getCdCurso() {
        return cdCurso;
    }
    public void setCdCurso(int cdCurso) {
        this.cdCurso = cdCurso;
    }
    public java.lang.String getDescricao() {
        return descricao;
    }
    public void setDescricao(java.lang.String descricao) {
        this.descricao = descricao;
    }
    public java.util.Calendar getDtCriacao() {
        return dtCriacao;
    }
    public void setDtCriacao(java.util.Calendar dtCriacao) {
        this.dtCriacao = dtCriacao;
    }
    public java.lang.String getNmCurso() {
        return nmCurso;
    }
    public void setNmCurso(java.lang.String nmCurso) {
        this.nmCurso = nmCurso;
    }
}

```

Figura 18 - Implementação da classe Curso.java

Na Figura 18 é ilustrada a implementação da classe Curso (classe de entidade), que é apenas uma classe que define seus atributos com métodos para acessá-los e modificá-los.

Na Figura 19 é ilustrada a implementação da classe CursoDAO (classe da camada DAO), que é a classe que manipula as informações de cursos, acessando sua respectiva tabela no banco.

```

public class CursoDAO extends sistemacurso.dao.DAO {

    public java.util.ArrayList pesquisar(java.util.Hashtable filtros) {
        String sql = "SELECT C.*,A.NmAreaCurso as NmAreaCurso FROM T_Curso as C, T_Area_Curso as A";
        sql+=" WHERE C.cdAreaCurso *= A.CdAreaCurso AND ";
        if(filtros!=null){
            java.util.Enumeration enum = filtros.keys();
            while(enum.hasMoreElements()){
                String campo = (String)enum.nextElement();
                if(campo.equalsIgnoreCase("cdcurso") ||
                    campo.equalsIgnoreCase("cdAreaCurso") ){
                    sql+= "C." + campo + "=" + filtros.get(campo);
                }else{
                    sql+= "C." + campo + " like '%" + filtros.get(campo) + "%"; |
                }
                sql+=" AND ";
            }
        }
        sql=sql.endsWith("AND ")?sql.substring(0,sql.length()-4):sql;

        sql+= " ORDER BY NmCurso";
        ArrayList resultado = new ArrayList();
        Curso curso = null;
        try{
            java.sql.Connection con = abrirConexao();
            ResultSet rs = createStatement(con).executeQuery(sql);

            while(rs.next()){
                curso = new Curso();
                curso.setCdCurso(rs.getInt("CdCurso"));
                curso.setNmCurso(rs.getString("NmCurso"));
                curso.setDescricao(rs.getString("DeCurso"));
                curso.setDtCriacao(getCalendar(rs.getTimestamp("DtCriacao")));

                String tipoCurso = rs.getString("TpCurso");
                if(tipoCurso!=null)
                    tipoCurso = tipoCurso.equals("E")?"Extensão":"Graduação";
                curso.setTipoCurso(tipoCurso);

                curso.setNomeArea(rs.getString("NmAreaCurso"));
                resultado.add(curso);
            }
        } catch(Exception e){
            e.printStackTrace();
        }
        return resultado;
    }
}

```

Figura 19 - Implementação da classe CursoDAO.java

Na Figura 20 é ilustrada a implementação da classe SistemaFacade, onde podemos observar que ela implementa a InterfaceSistema, seus métodos acessam o negócio da aplicação.

```

public class SistemaFacade implements sistemacurso.InterfaceSistema{

    public AreaCurso getAreaCurso(int cd) {
        Hashtable filtros = new Hashtable();
        filtros.put("CdAreaCurso", String.valueOf(cd));
        ArrayList lista = new AreaCursoDAO().pesquisar(filtros);
        AreaCurso areaCurso = lista.size()==0?null:(AreaCurso)lista.get(0);
        return areaCurso;
    }

    public AreaCurso[] getListaAreasCurso() {
        ArrayList lista = new AreaCursoDAO().pesquisar();

        AreaCurso[] areaCursos = new AreaCurso[lista.size()];
        Iterator it = lista.iterator();
        int i=0;
        while(it.hasNext()){
            areaCursos[i++]=(AreaCurso)it.next();
        }
        return areaCursos;
    }

    public Curso getCurso(int cd) {
        Hashtable filtros = new Hashtable();
        filtros.put("CdCurso", String.valueOf(cd));
        ArrayList lista = new CursoDAO().pesquisar(filtros);
        Curso curso = lista.size()==0?null:(Curso)lista.get(0);
        return curso;
    }

    public Curso[] getListaCursos() {
        ArrayList lista = new CursoDAO().pesquisar();

        Curso[] cursos = new Curso[lista.size()];
        Iterator it = lista.iterator();
        int i=0;
        while(it.hasNext()){
            cursos[i++]=(Curso)it.next();
        }
        return cursos;
    }

    .
    .
    .

}

```

Figura 20 – Trecho de código da classe SistemaFacade.java

Após a implementação destas estruturas de apoio, pode-se utilizar as ferramentas oferecidas pelo Apache Axis, para se gerar o documento WSDL. O Axis irá gerar este artefato baseado no que foi definido no arquivo InterfaceSistema.java. Observe na Figura 21 o documento WSDL gerado pelo Axis:

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="urn:EadUNIFOR"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="urn:EadUNIFOR" xmlns:intf="urn:EadUNIFOR"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:tns2="http://entidades.sistemacurso"
  xmlns:wSDL="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wSDLsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
  <schema targetNamespace="http://entidades.sistemacurso" xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />

    <complexType name="Curso">
      <sequence>
        <element name="cdCurso" type="xsd:int"/>
        <element name="descricao" nillable="true" type="xsd:string"/>
        <element name="dtCriacao" nillable="true" type="xsd:dateTime"/>
        <element name="nmCurso" nillable="true" type="xsd:string"/>
        <element name="tipoCurso" nillable="true" type="xsd:string"/>
        <element name="nomeArea" nillable="true" type="xsd:string"/>
      </sequence>
    </complexType>
    .
    .
  </schema>
  <schema targetNamespace="urn:EadUNIFOR" xmlns="http://www.w3.org/2001/XMLSchema">
    <import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
    <complexType name="ArrayOf_tns2_Curso">
      <complexContent>
        <restriction base="soapenc:Array">
          <attribute ref="soapenc:arrayType" wsdl:arrayType="tns2:Curso[]"/>
        </restriction>
      </complexContent>
    </complexType>
    <complexType name="ArrayOf_tns2_TurmaCursoDisciplina">
      .
      .
    </complexType>
  </schema>
  <wsdl:message name="getListaCursosResponse">
    <wsdl:part name="getListaCursosReturn" type="impl:ArrayOf_tns2_Curso"/>
  </wsdl:message>

  <wsdl:message name="getCursoRequest">
    <wsdl:part name="in0" type="xsd:int"/>
  </wsdl:message>
</wsdl:definitions>
```

Figura 21 – Trecho do WSDL gerado pelo Axis para os serviços

Após a criação destes artefatos, pode-se utilizar o Axis novamente para se gerar a partir deste documento WSDL mostrado na Figura 21, as classes que este irá utilizar para efetivar a interação do SOAP com as classes da aplicação.

Dentre estas classes estão os *Skeletons* que como já foram vistos no capítulo 2, irão responder às solicitações recebidas via SOAP.

III. Transição

Nesta fase de transição, é feita a documentação dos artefatos produzidos nas etapas anteriores, bem como o manual de utilização dos serviços. A documentação sobre objetos de negócio envolvidos faz-se útil para os desenvolvedores por parte da integração dos serviços, para que melhor possam utilizar-se dos serviços ofertados.

Logo após, se dá a finalização da implementação com empacotamento das classes, e publicação dos serviços no Axis, que possui um registro dos serviços publicados. Veja, na Figura 22, a lista dos serviços registrados:

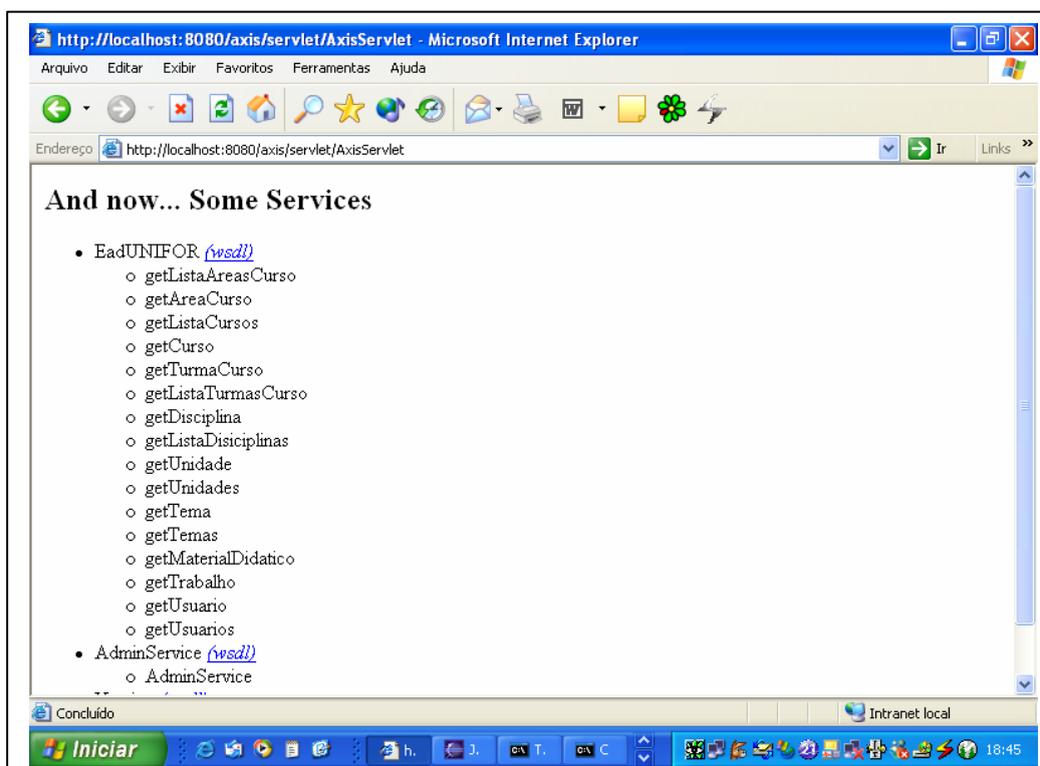


Figura 22 - Serviços publicados no Registro de Serviços do Axis

2) Integração dos serviços: Utilizador de Serviços

I. Iniciação

Nesta fase, os projetistas de integração irão projetar a arquitetura para a integração e definir como será feita a integração do sistema utilizador com os serviços ofertados pelo Cadinet.

Como dito anteriormente, o sistema utilizador terá duas visões: uma visão da aplicação disponibilizada em web e a outra aplicação em desktop (janelas windows).

Considere que os requisitos funcionais (serviços web) identificados para a aplicação são os seguintes:

- Mostrar a lista de cursos do Cadinet.
- Fornecer informações sobre cada curso ofertado.

Um requisito não funcional identificado é o seguinte:

- Os serviços web deverão estar acessíveis na aplicação web, como também na aplicação desktop do sistema utilizador.

A partir destes requisitos, os projetistas de integração projetam então a arquitetura do sistema integrado. (Ver Figura 23).

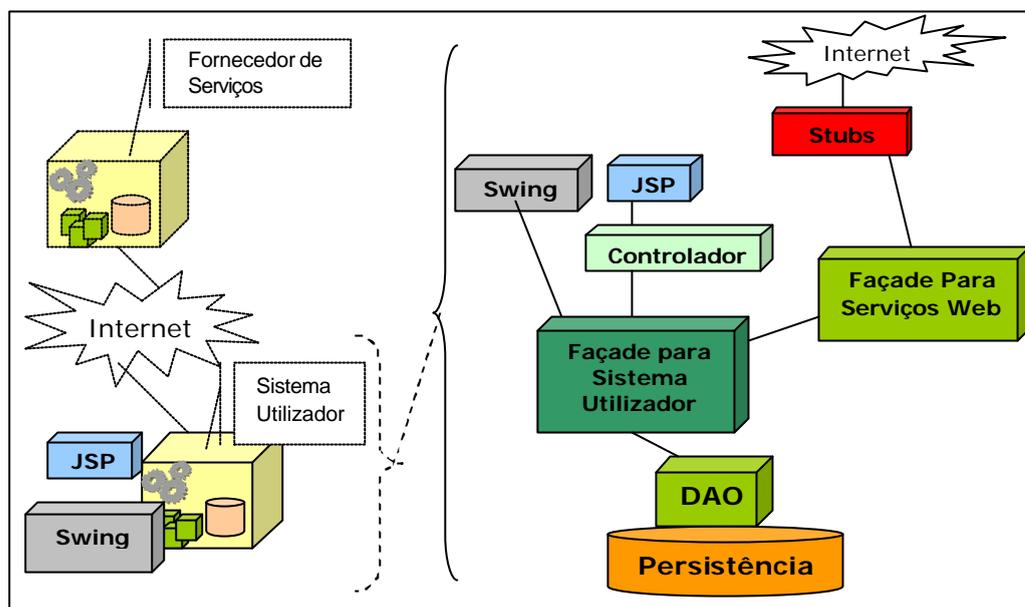


Figura 23 - Esboço da arquitetura de integração

Na arquitetura de integração foi criada uma classe *Façade* identificada como “Façade Para Serviços Web”, a qual se comunica diretamente com as classes *Stubs* para o serviço web. Isso proporciona uma diminuição do acoplamento entre a aplicação utilizadora e os serviços web. Desta forma, o “Façade Para o Sistema Utilizador”, considerando que o mesmo já existia antes da integração, é modificado para interagir com o “Façade Para Serviços Web”, proporcionando assim uma comunicação de alto nível entre a aplicação e os serviços. Diante deste cenário, considerando que os demais elementos da arquitetura do sistema utilizador já existiam anteriormente, é notável a flexibilidade da arquitetura, o que facilita na modificação.

Para definir como a integração do Sistema Utilizador e os serviços web requisitados será feita, é importante fazer uma modelagem das estruturas das classes envolvidas. Em seguida, a Figura 24 mostra o relacionamento estrutural entre estas classes. Percebe-se que o acesso ao negócio da aplicação é todo mediado pela classe “FacadeUtilizador”, a qual fornece uma visão uniforme para o acesso via JSP/Controladores e o acesso via aplicação swing

(desktop). Isso facilita, pois faz com que as visões não necessitem saber se os dados de cursos, que estão sendo apresentados por elas, são de proveniência de uma base de dados local, ou de serviços web remotos. Outro fato interessante, é que a classe “FacadeUtilizador” irá interagir com a classe “ServicosCadinetFacade”, a qual detém o conhecimento de quais *Stubs* interligam-na com os serviços web do Cadinet. Deixar o sistema o mais independente o possível foi o motivo da criação de um *Façade* para os serviços web (*Stubs*).

Desta forma, se futuramente a aplicação utilizadora deixar de utilizar os serviços do Cadinet, basta apenas que esta efetue o desligamento entre as duas classes *Façade*.

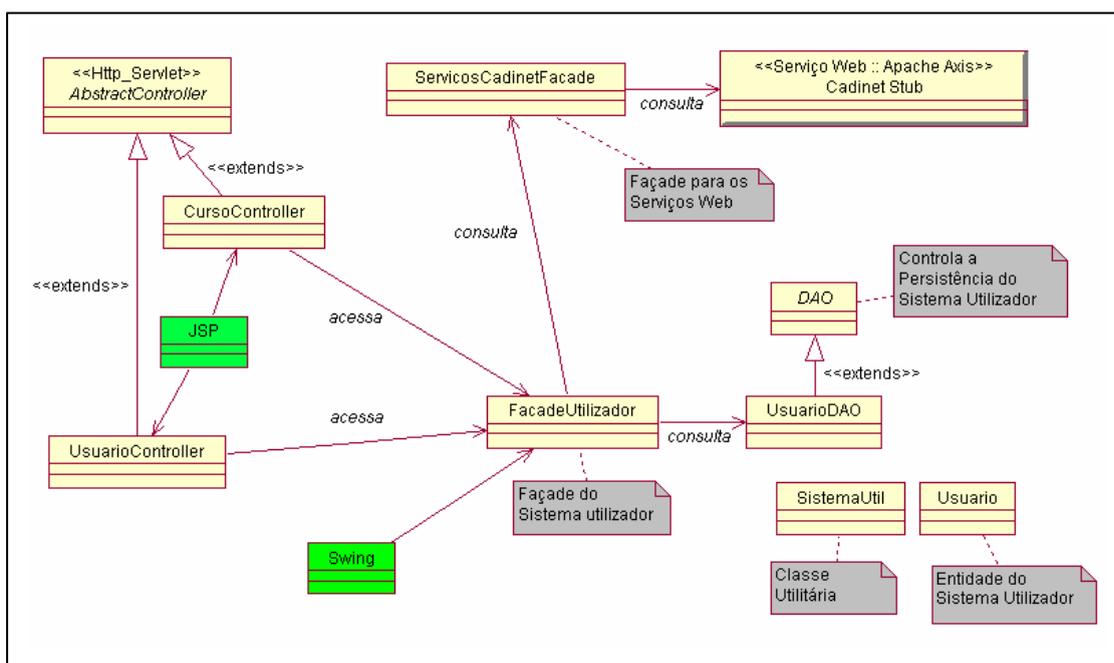


Figura 24 - Diagrama de classes do sistema utilizador

A Figura 25 ilustra a troca de mensagens entre as classes no diagrama de seqüência, representando o funcionamento do acesso ao método `getListaCurso()` por parte das visões do sistema utilizador. Pode-se perceber que as visões JSP ou Swing irão efetuar a solicitação da lista de cursos do

Cadinet ao *FacadeUtilizador*, que repassará a solicitação para a classe *ServicosCadinetFacade*, na qual obterá o *stub* para interagir com o serviço web de uma classe *ServiceLocator*. Com o *stub*, a solicitação é enviada para o Serviço Web do Cadinet que responderá com a lista de cursos, podendo então ser exibido o resultado na visão.

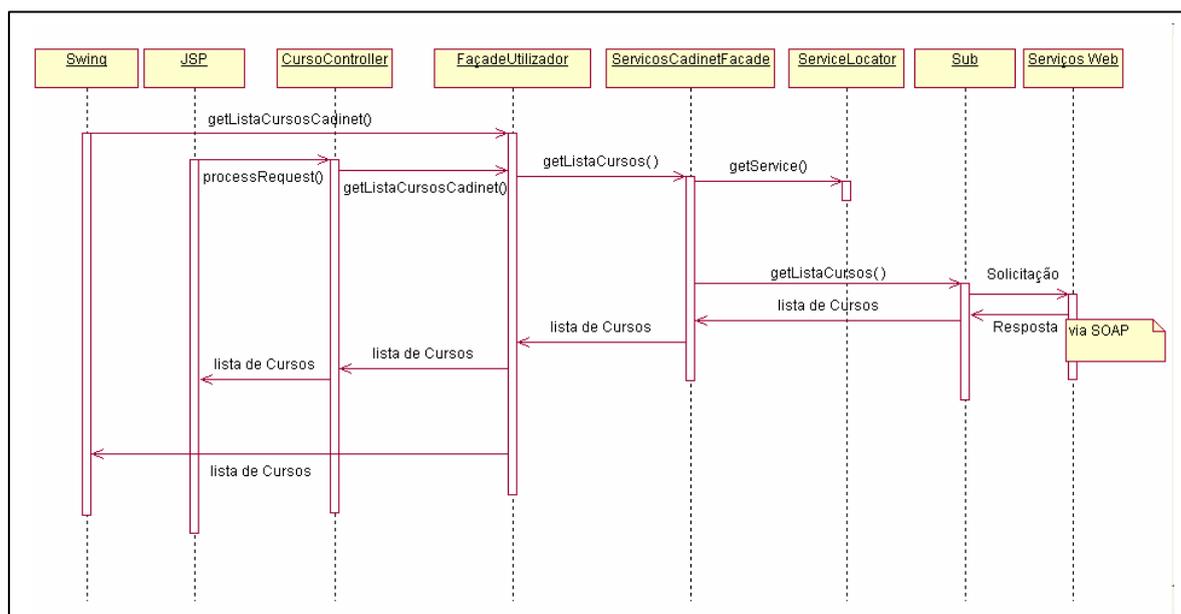


Figura 25 - Diagrama de seqüência para recuperar lista de cursos

II. Integração

Nesta etapa, os programadores de integração se envolvem e implementam o que foi definido na fase anterior. É utilizada a ferramenta do Axis novamente para se obter os *stubs* a partir do WSDL anteriormente publicado. Esta ferramenta portanto, irá gerar todas as classes necessárias para que o sistema utilizador possa efetuar a interação com os serviços web, como é o caso dos *stubs*, *ServiceLocators*, e classes de entidade do Cadinet (Curso, etc.).

A partir daí, a estratégia é criar um novo *Facade* só para controlar estes serviços, ou seja, ter acesso direto com os *stubs*. O objetivo é diminuir o

acoplamento, não vinculando diretamente o *Façade* principal da aplicação utilizadora com os *stubs* que são frequentemente modificados. Após a criação desta classe *Façade*, faz-se uma ligação entre a classe *FaçadeUtilizador* (já pertinente ao sistema utilizador) com a classe *ServicosCadinetFacade* (classe criada para interagir com os *Stubs*). Logo em seguida são modificados as visões e os controladores para que possibilitem a visualização e utilização dos novos serviços. Na Figura 26 é ilustrada a implementação da classe *ServicosCadinetFacade*, onde podemos observar a utilização de um *ServiceLocator*, que localiza e retorna um objeto *stub* para que possam ser efetivadas as chamadas remotas.

```
public class ServicosCadinetFacade implements InterfaceSistema{
    private InterfaceSistema eadSistema;
    private InterfaceSistemaService service;
    private SistemaUtil sisUtil;

    public ServicosCadinetFacade() {
        try {
            service = new InterfaceSistemaServiceLocator();
            eadSistema = service.getEadUNIFOR();
            sisUtil = new SistemaUtil();
        } catch(Exception e) {
            e.printStackTrace();
        }
    }

    public Curso[] getListaCursos() {
        Curso[] cursos = null;
        try {
            cursos = eadSistema.getListaCursos();
        } catch(Exception e) {
            e.printStackTrace();
        }
        return cursos;
    }

    public AreaCurso getAreaCurso(int cd) {
        ▪
        ▪
        ▪
    }
}
```

Figura 26 – Trecho de código da classe *ServicosCadinetFacade*

Na Figura 27, é ilustrada a implementação da classe FacadeUtilizador onde pode-se observar métodos de negócio referentes a aplicação utilizadora, bem como a interação com a classe ServicosCadinetFacade da qual se obtém acesso aos serviços web do Cadinet, como visto anteriormente.

```
public class FacadeUtilizador{

    private ArrayList getResults(sistemautilizador.dao.DAO dao,String chave,String valor){
        Hashtable filtros = new Hashtable();
        filtros.put(chave, valor);
        return dao.pesquisar(filtros);
    }

    public Usuario validaUsuario(String login,String senha){
        Hashtable filtros = new Hashtable();
        filtros.put("login", ""+login);
        filtros.put("senha", ""+senha);
        ArrayList lista = new UsuarioDAO().pesquisar(filtros);
        Usuario user = lista.size()==0?null:(Usuario)lista.get(0);
        return user;
    }

    public Curso[] getListaCursoCadinet(){
        return new ServicosCadinetFacade().getListaCursos();
    }

    public Curso getCursoCadinet(int cd){
        return new ServicosCadinetFacade().getCurso(cd);
    }

    •
    •
    •
}
```

Figura 27 – Trecho de código da classe FacadeUtilizador

III. Transição

Nesta fase, os documentadores fazem a documentação do que foi implementado para a efetivação da integração, bem como sua documentação da arquitetura e proposta.

O próximo passo, portanto, é a utilização do que foi construído, isto envolve a necessidade dos programadores de integração em manter e melhorar o que foi construído em um ciclo contínuo durante a vida do software. Os testadores irão ficar responsáveis pela efetivação dos testes, e da verificação da performance de acesso aos serviços. A Figura 28 ilustra a visão web da aplicação utilizadora, enquanto a Figura 29 ilustra a visão em desktop (swing), ou aplicativo *windows*.

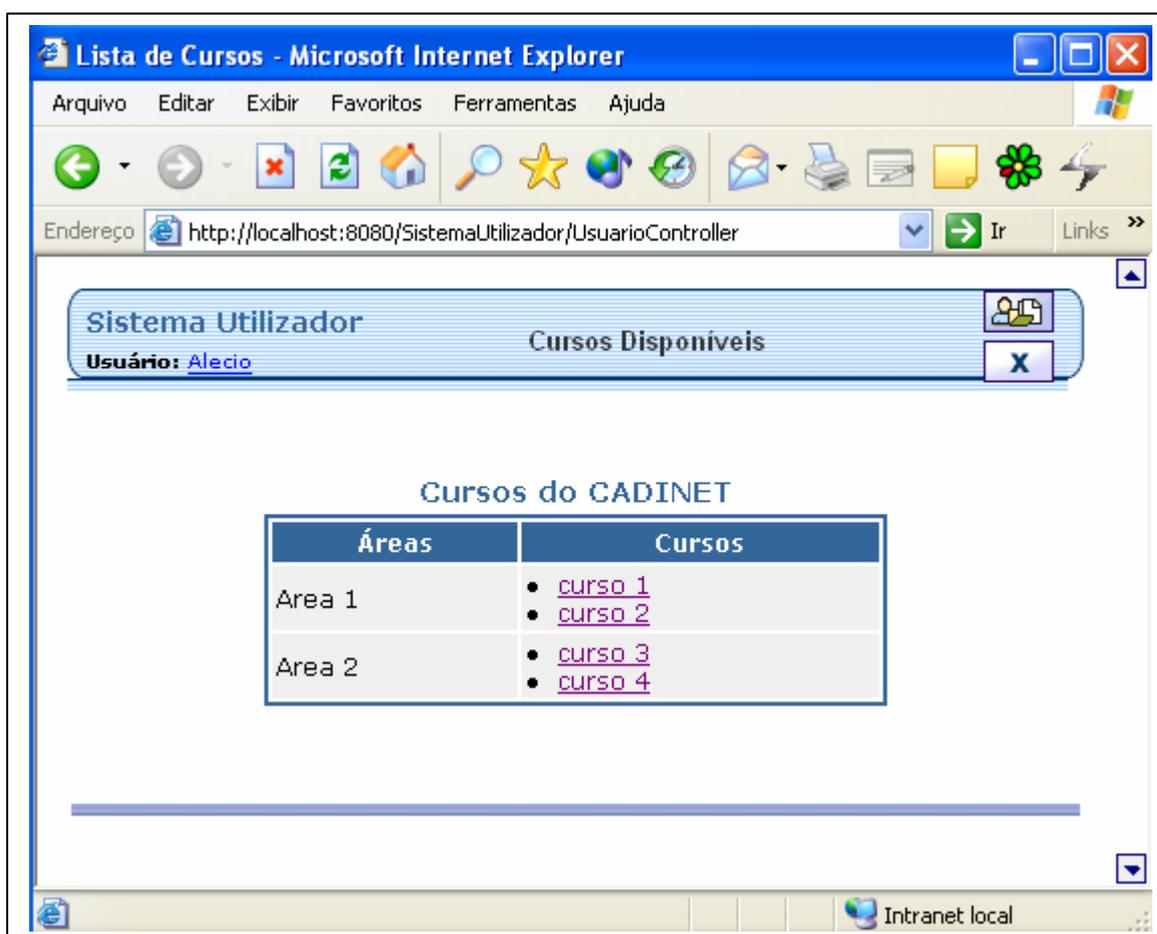


Figura 28 – Visão Web da aplicação utilizadora

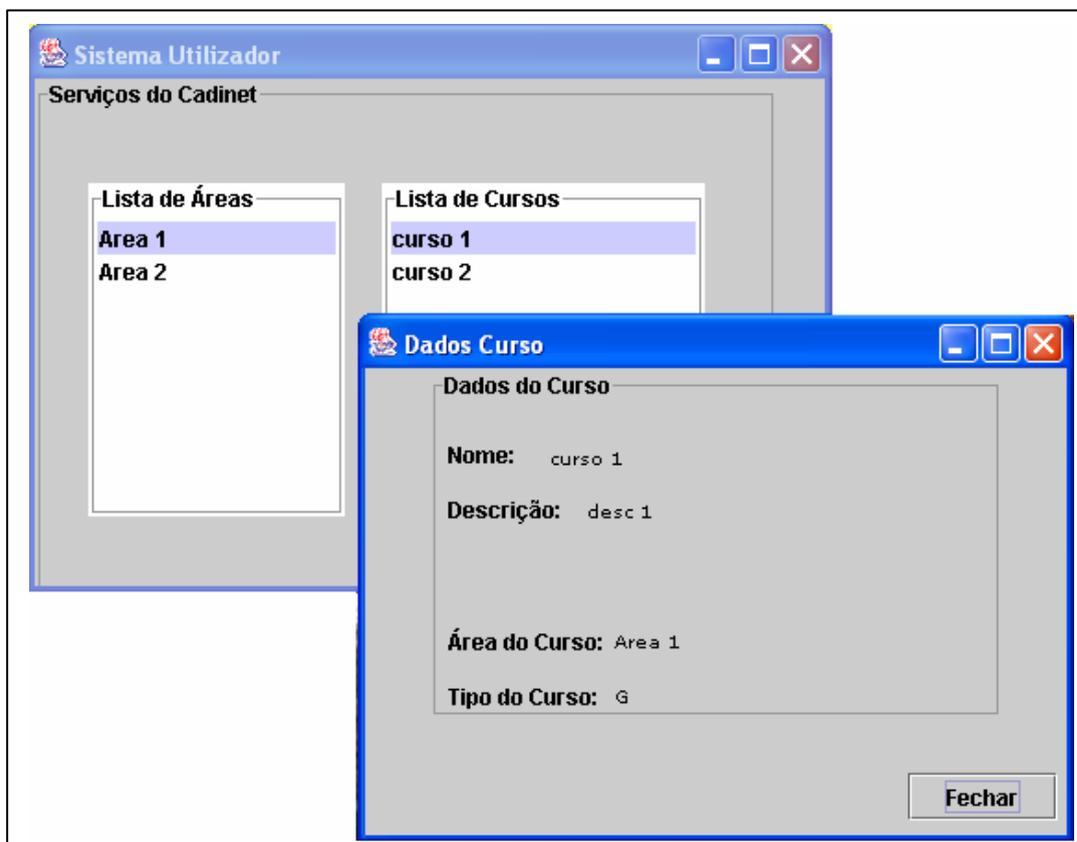


Figura 29 – Visão Swing (desktop) da aplicação utilizadora.

4.3. Conclusão

Neste capítulo foi definida uma metodologia para guiar o processo de desenvolvimento de softwares que se integram utilizando serviços web como tecnologia. Foram explorados as etapas, e os conceitos que envolvem esta metodologia, bem como sua aplicação em um estudo de caso, demonstrando sua aplicabilidade. Foram também mostrados a utilização de padrões de projeto na especificação da arquitetura dos sistemas envolvidos no estudo de caso, e o quão flexíveis os tornaram.

5. Conclusão

Neste trabalho foi apresentada uma nova metodologia de desenvolvimento para projetos de integração de software com a utilização de serviços web. Esta auxilia os desenvolvedores na definição dos serviços e na utilização dos mesmos.

Conceitos sobre serviços web, padrões de projeto, os quais foram vistos neste trabalho, foram aplicados em um estudo de caso, demonstrando a aplicabilidade da metodologia. Pudemos discutir a importância de se aplicar padrões de projeto em arquiteturas de softwares para integração com serviços web, objetivando obter um sistema mais independente e flexível possível.

Pode-se concluir que a aplicação desta metodologia proporcionará a organização do processo de desenvolvimento de software para sistemas de integração com serviços web. Isso porque a organização não irá se preocupar somente com o lado Fornecedor, mas também com o lado Utilizador dos serviços. Também pode-se concluir que com a utilização de padrões de projetos, pode-se arquitetar uma estrutura de software capaz de responder às frequentes modificações de um sistema.

Como possibilidade de trabalhos futuros podem ser citados os seguintes:

- Para verificar a abrangência da MISW, poderia ser efetuado um estudo de sua aplicabilidade em serviços web construídos na plataforma .NET, por exemplo.
- Criar uma ferramenta que auxilie no processo de desenvolvimento proposto pela MISW.

6. Bibliografia

- [TJWST,2003] Eric Armstrong, Stephanie Bodoff, Debbie Carson, Maydene Fisher, Scott Fordin, Dale Green, Kim Haase, Eric Jendrock, **The Java Web Services Tutorial**, SUN Microsystems, 2003.
- [MEJB,2001] Ed Roman, Scott Ambler, Tyler Jewell, **Mastering Enterprise JavaBeans (2nd edition)**, John Wiley & Sons, 2001.
- [SCWCDSK,2002] Hanumant Deshmukh, Jignesh Malavia, Jacquelyn Carter, **SCWCD Exam Study Kit: Java Web Component Developer Certification**, Manning, 2002.
- [PJWS,2002] Scott Cable, Ben Galbraith, Romin Irani, Mack Hendricks, James Milbury, Tarak Modi, Andre Tost, Alex Toussaint, Jeelani Basha, **Professional Java Web Services**, Wrox, 2002.
- [JWSA,2003] James McGovern, Sameer Tyagi, Michael Stevens, Sunil Mathew, **Java Web Services Architecture**, Morgan Kaufmann, 2003.
- [JWSOR,2002] David A. Chappell, Tyler Jewell, **Java Web Services**, O'Reilly, 2002.
- [PPROJ,2000] Richard Helm, Ralph Johnson, John Vlissides, **Padrões de Projeto**, Bookman, 2000.
- [KRUCHTEN,2000] KRUCHTEN, Philippe. **The Rational Unified Process - An Introduction**. 2 ed. New Jersey: Addison -Wesley, 2000

6.1. Referências On-line

- [J2EE,2003] SUN Microsystems, **J2EE Patterns**
Disponível em: <<http://java.sun.com/blueprints/patterns>>
Acessado em: 14 de Outubro, 2003
- [AXIS,2003] Apache, Apache Axis
Disponível em: <<http://ws.apache.org/axis>>
Acessado em: 18 de Agosto, 2003
- [BPFWS1,2002] IBM, Best practices for Web services, **Back to the basics Part 1**
Disponível em: <<http://www-106.ibm.com/developerworks/webservices/library/ws-best1>>
Acessado em: 21 de Agosto, 2003
- [BPFWS2,2002] IBM, Best practices for Web services, **Back to the basics Part 2**
Disponível em: <<http://java.sun.com/blueprints/patterns>>
Acessado em: 21 de Agosto, 2003
- [RUP,2003] Mauro Vianna, **Conheça o Rational Unified Process (RUP)**
Disponível em:
<http://www.linhadecodigo.com.br/artigos.asp?id_ac=79&page=1>
Acessado em: 18 de Setembro, 2003