DIÓGENES CORRÊA CANOZA

PADRÕES DE PROJETO: UM ESTUDO DE CASO UTILIZANDO UM FRAMEWORK DE DESENVOLVIMENTO ORIENTADO A SERVIÇOS

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Orientador
Prof. André Vital Saúde

Lavras
Minas Gerais - Brasil
2009

DIÓGENES CORRÊA CANOZA

PADRÕES DE PROJETO: UM ESTUDO DE CASO UTILIZANDO UM FRAMEWORK DE DESENVOLVIMENTO ORIENTADO A SERVIÇOS

Monografia de Graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Ciência da Computação para obtenção do título de Bacharel em Ciência da Computação.

Aprovada em *16 de junho de 2009*

Prof. Antônio Maria Pereira de Resende

Prof. Paulo Henrique de Souza Bermejo

Prof. André Vital Saúde
(Orientador)

Lavras
Minas Gerais - Brasil
2009

Dedico essa monografia à Simone.

Agradecimentos

Agradeço aos meus familiares e amigos que muito me ajudaram em todos esses anos. E a meu orientador, que tornou possível tudo isso.

Sumário

Lista de Figuras

Lista de Tabelas

PADRÕES DE PROJETO: UM ESTUDO DE CASO UTILIZANDO UM FRAMEWORK DE DESENVOLVIMENTO ORIENTADO A SERVIÇOS

Resumo

No Departamento de Ciência da Computação da Universidade Federal de Lavras está sendo desenvolvido um *framework* para desenvolvimento de sistemas baseado em *Service-Oriented Architecture* (SOA), chamado Iguassu. Ele traz vantagens em agilizar o tempo de contrução de uma aplicação,mas ainda não está concluído. Tendo como motivação melhorar o Iguassu, este trabalho realizou um estudo sobre padrões de projeto e verificou-se como poderiam ser integrados ao Iguassu. Foram encontrados alguns casos de uso de chamada a serviços, os quais, com o auxílio dos padrões de projeto, foi possível modelar e implementar soluções para eles. Para testar esta implementação foi proposto um modelo de testes, também aplicado neste trabalho de conclusão de curso.

Palavras-Chave: Arquitetura Orientada a Serviço; Padrões de projeto; Iguassu.

DESIGN PATTERNS: A STUDY CASE USING A SERVICE ORIENTED DEVELOPMENT FRAMEWORK

Abstract

At the Department of Computer Science of thr Federal University of Lavras is being developed a framework for developing systems based on Service-Oriented Architecture (SOA), called Iguassu. It brings advantages in speeding up the development time of an application, but is not yet complete. Having as a motivation to improve the Iguassu, this work conducted a study on design patterns and was observed how them could be integrated to Iguassu. We found some use cases of call services, which, with the help of design patterns, it was possible to model and implement solutions for them. To test this implementation has been proposed a model for testing, also applied to this curse conclution work.

Keywords: Service-Oriented Architecture; Design Patterns; Iguassu.

Capítulo 1

Introdução

A construção de sistemas não é uma tarefa simples de ser executada sem o auxílio de um *framework* de desenvolvimento.

No Departamento de Ciência da Computação da Universidade Federal de Lavras está sendo desenvolvido um *framework* para desenvolvimento de sistemas baseado em *Service-Oriented Architecture* (SOA). Este é chamado Iguassu.

O Iguassu vem sendo implementado em tecnologia Java, e faz uso de *frameworks* livres existentes, como *frameworks* de persistência e de comunicação *web*. Ele ainda proporciona geração automática de código a partir de diagramas representativos, com o intuito de diminuir o tempo de desenvolvimento das aplicações que o utilizam. Além disso, é fundado em vários padrões de projeto, onde alguns serão discutidos no decorrer deste trabalho.

O *framework* Iguassu ainda está em fase de desenvolvimento. Mesmo que ainda não esteja pronto, ele já está sendo utilizado para o desenvolvimento de dois sistemas, um de rastreabilidade e um de monitoramento de produção industrial.

Durante o desenvolvimento destes sistemas foram identificadas algumas deficiências no Iguassu. Uma delas está no modo como são acessados os serviços. A maneira como eles estão organizados atualmente gera dúvidas quanto a sua localização. Isto é algo que deve ser melhorado.

O fato do Iguassu ser uma ferramenta que utiliza de *frameworks* e tecnologias que estão atualmente em evidência e ainda ter a proposta de diminuir o tempo de desenvolvimento de sistemas, já justificaria o interesse em melhorá-lo. Esta justificativa ainda é reforçada pelo fato de existir sistemas que o utilizam e estes necessitarem que o Iguassu atinja um grau maior de maturidade, para que possam ser concluídos.

A melhoria proposta por este trabalho é a de buscar por padrões de projeto que tragam benefícios ao Iguassu. Além disso devem ser encontrados pontos onde esses padrões podem ser integrados, para assim poderem ser implementados. Por final essa implementação deve ser testada.

1.1 Objetivo geral

Este trabalho tem por objetivo geral propor melhorias ao framework Iguassu, a partir do estudo e integração de padrões de projeto ao mesmo.

1.2 Objetivos específicos

A fim de alcançar o objetivo geral deste trabalho, serão considerados os seguintes objetivos específicos:

- Estudar padrões de projeto que possam trazer melhorias ao Iguassu;
- Integrar os padrões estudados ao Iguassu;
- Testar a integração dos padrões de projeto;

1.3 Estrutura do trabalho

Este trabalho está divido em cinco capítulos, esses capítulos estão estruturados como descrito abaixo:

- *Capítulo 1: Introdução* - Apresenta a introdução do trabalho, incluindo a apresentação, definição do problemas, os objetivos geral e específicos, justificativa e estrutura do trabalho.
- *Capítulo 2: Referencial teórico* - Apresenta os conceitos que foram utilizados por este trabalho.
- *Capítulo 3: Metodologia* - Demonstra os métodos que foram utilizados para se alcançar os objetivos do trabalho.
- *Capítulo 4: Resultados* - Apresentada os resultados conseguidos pela execução do trabalho.
- *Capítulo 5: Discussão* - Discute sobre todo o trabalho que foi realizado.

Capítulo 2

Revisão Bibliográfica

Neste capítulo, serão levantados os conceitos necessários para o entendimento do projeto. O capítulo está dividido nas seguintes seções: arquitetura orientada a serviço, *Web Services*, padrões de projeto e Iguassu.

Na Seção 2.1 será apresentado o conceito de arquitetura orientada a serviços, descrevendo quais suas características e vantagens. Também será apresentada uma arquitetura de referência.

Existem muitas maneiras de se implementar SOA, utilizando-se de diferentes protocolos e estratégias para a distribuição dos serviços. Mas MACKENZIE *et al.* (2006) diz que SOA geralmente é implementada com a utilização de *Web Services*. Então a Seção 2.2 fará referência a esse conceito.

A Seção 2.3 listará os padrões de projeto que foram utilizados no decorrer desse trabalho. Os padrões foram escolhidos por mostrarem-se vantajosos ao Iguassu. Esse por sua vez será abordado na Seção 2.4.

2.1 Arquitetura Orientada a Serviço (SOA)

Arquitetura Orientada a Serviço (*Service-oriented Architecture*, SOA) é uma arquitetura de desenvolvimento cujo objetivo é criar módulos funcionais chamados de serviços, com baixo acoplamento e permitindo reutilização de código (SAMPAIO, 2006). Com isto, o uso de SOA permite aumentar a capacidade de uma empresa, em atender a novas necessidades de negócios, em um curto espaço de tempo (KRAFZIG; BANKE; SLAMA, 2004).

Para MACKENZIE *et al.* (2006), quando uma entidade (pessoa ou organização) possui uma necessidade ela acaba desenvolvendo uma habilidade para resolvê-la. A necessidade dessa entidade pode corresponder a uma habilidade de outra (MACKENZIE *et al.*, 2006). Essa correspondência não necessariamente precisa aparecer na razão um pra um. Assim cada necessidade pode precisar de mais de uma habilidade para ser satisfeita, já cada habilidade pode satisfazer mais de uma necessidade (MACKENZIE *et al.*, 2006). SOA é uma poderosa arquitetura que ajuda a corresponder as habilidades úteis para resolver uma necessidade, e dizer quais necessidades uma habilidade pode satisfazer (MACKENZIE *et al.*, 2006).

MACKENZIE *et al.* (2006) diz que, visibilidade, interação e efeito são as chaves dos conceitos para a descrição do paradigma SOA. Esses conceitos como descritos por MACKENZIE *et al.* (2006) são dados a seguir:

- Visibilidade refere-se à capacidade de cada necessidade e cada habilidade de poder enxergar uma à outra. Isto é tipicamente feito através do fornecimento de descrições tais como funções e aspectos técnicos, exigências, limitações e políticas afins, e mecanismos de acesso ou de resposta. As des-

crições têm de estar em um formulário (ou podendo ser transformadas para um formulário) no qual a sua sintaxe e semântica são amplamente acessíveis e compreensíveis.

- Interação é a atividade que consiste de se utilizar as habilidades. Normalmente medida pela troca de mensagens, produto de interações através de uma série de informações trocadas e ações invocadas. Existem muitos modos de interação, mas todos são fundamentados em um contexto especial de execução, um conjunto de técnicas e elementos de negócio que formam um caminho entre as necessidades e as habilidades. Isso permite que provedores de serviços e consumidores interajam e fornece um ponto de partida para quaisquer políticas e contratos que possam estar em vigor.

- A razão para se utilizar as habilidades é a de produzir um ou mais efeitos no mundo real. Uma interação é uma ação e o resultado dessa ação é um efeito ou um conjunto de efeitos. Esse efeito pode retornar uma informação ou mudar o estado de entidades (conhecidas ou não) que estão envolvidas na interação.

A descrição de SOA que está sendo apresentada, ainda não considerou seu principal conceito: o serviço. Serviço é um componente de *software* com um significado funcional distinto (KRAFZIG; BANKE; SLAMA, 2004). Ele tipicamente encapsula uma lógica de negócio (KRAFZIG; BANKE; SLAMA, 2004; BARAI; CASELLI; CHRISTUDAS, 2008). Um serviço pode ser uma unidade lógica individual ou ainda ser composto de outros serviços (BARAI; CASELLI; CHRISTUDAS, 2008).

2.1.1 Arquitetura de referência SOA

Para ajudar a desenvolver uma linguagem comum e um corpo de conhecimento coletivo sobre SOA, um grupo de praticantes de SOA formado por profissionais de diversas empresas com o patrocínio da *BEA Systems*, decidiu por criar um guia, esse guia é o *SOA Practitioner's Guide* (DURVASULA *et al.*, 2006). Nesse guia está descrita uma arquitetura de referência SOA. A Figura 2.1 ilustra essa arquitetura. Ela é formada por várias camadas e em diferentes níveis. As camadas que compõe a arquitetura são: a camada de aplicações *web, (Web Application Tier)*, a camada de serviços *(Service Tier)*, camada de aplicações *(Application Tier)* e a camada de infra-estrutura (*Infrastructure*). Essas camadas serão brevemente abordadas a seguir, com base no que descreve DURVASULA *et al.* (2006):

- **Camada de Aplicações Web**. Essa camada é a camada de interface ou apresentação. A principal exigência dessa camada é que todos os sistemas e soluções, possam ser acessados a partir de qualquer navegador *web* compatível. Ela ainda possui subdivisões que não serão abordadas por não serem pertinentes a este trabalho.

- **Camada de Serviço**. A camada de serviço é a principal camada para SOA. Essa camada é responsável por manter e manipular os serviços. Algumas de suas subdivisões, que serão utilizadas por este trabalho, são abordadas a seguir:

  - ***Service Registry* e *SOA Repository***. *Service Registry*, registrador de serviços, é um catálogo de serviços disponíveis. Já *SOA Repository*, repositório de SOA, é um componente que mantém informações, a

cerca dos serviços. Como por exemplo uma lista de todos os serviços que foram definidos pela empresa.

  - ***Service Manager***. O *Service Manager*, gerenciador de serviços, possui a função de manter, monitorar e reportar sobre todos os serviços definidos.
  - ***Shared Data Services***. *Shared Data Services*, serviços de dados compartilhados, são os serviços de acesso a dados. Eles fazem a manutenção dos dados no banco de dados.

- **Camada de Aplicação**. Os maiores investimentos das organizações de TI são em aplicações. Essa camada representa todas as aplicativos de uma empresa. Mesmo que os investimentos das empresas em SOA avancem, a camada de aplicação não deixará de existir tão cedo.

- **Camada de infra-estrutura**. Nessa camada estão todos os componentes necessários para realizar a computação, a distribuição e o armazenamento dos dados. Como por exemplo: redes, roteadores, servidores, *data centers* entre outros. Todas as outras camadas são construídas sobre a camada de infra-estrutura.

Nem todas as organizações de TI necessitam implementar tudo o que está nesta arquitetura de referência SOA, (Figura 2.1). Uma das melhores práticas SOA está em investir na infra-estrutura somente quando for obrigado a fornecer soluções de negócios (DURVASULA *et al.*, 2006).

O entendimento da arquitetura apresentada é importante para o entendimento do *framework* estudado por este trabalho. Isso porque este *framework* é baseado na arquitetura de referência apresentada. O *framework* será melhor abordado na Seção 2.4.

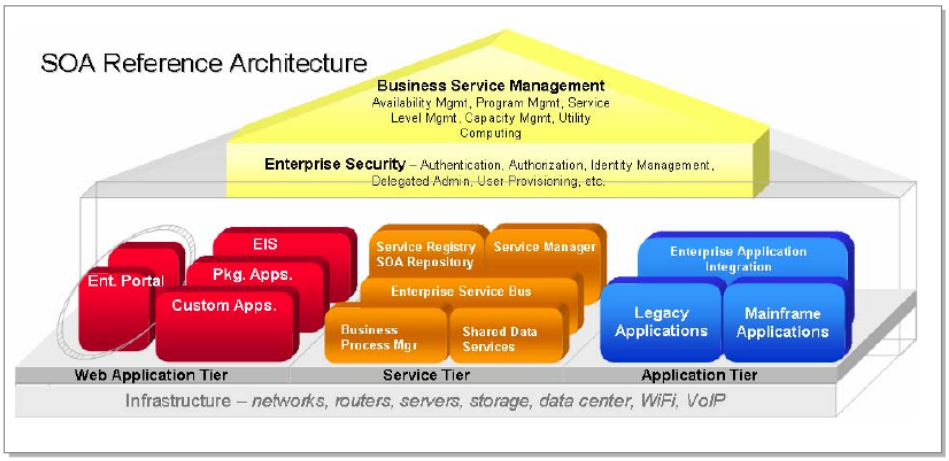

Figura 2.1: Arquitetura de referência SOA, retirada de DURVASULA *et al.*(2006)

2.2 Web Services

Web Service (BOOTH *et al.*, 2004) é uma forma que foi criada para auxiliar a comunicação entre diferentes aplicações. Os *Web Services* fazem sua comunicação através de mensagens independentes de plataforma, utilizando-se de *Extensible Markup Language (XML)*, sobre um *Simple Object Access Protocol (SOAP)*, (GUDGIN *et al.*, 2007).

GUDGIN *et al.* (2007) descreve *SOAP* como um protocolo para realizar a troca de informações estruturadas em um ambiente decentralizado e distribuído. Ele define uma estrutura para a construção da mensagem *XML* capaz de trafegar por diversos protocolos de comunicação. *SOAP* possui uma mensagem estruturada, mas quem define a gramática para a construção dessa mensagem é a *Web Services Description Language (WSDL)*, (RYMAN *et al.*, 2007).

Para RYMAN *et al.* (2007), *WSDL* é um modelo em *XML* que descreve o formato da mensagem que um *Web Service* reconhece e também um conjunto de

endereços onde são providos os serviços. Utilizando-se desse modelo de comunicação as aplicações podem estar em diferentes plataformas e implementadas em diferentes linguagens. Assim consegue-se um maior reuso de código e ainda pode-se dividir a carga de processamento para mais de um local.

2.3 Padrões de Projeto

Padrões de projeto são soluções recorrentes para problemas de projeto que são encontrados constantemente (ALPERT; BROWN; WOOLF, 1998). Eles nos permitem documentar esse problema e sua solução em um contexto particular, e divulgar esse conhecimento para outras pessoas (ALUR; MALKS; CRUPI, 2003). GAMMA *et al.* (1994) diz que um padrão de projeto geralmente possui quatro elementos: nome, descrição do problema, solução e conseqüências. Esses elementos segundo GAMMA *et al.* (1994), são descritos a seguir:

- O nome do padrão de projeto é um algo que define, em uma ou duas palavras, o problema, a solução e as conseqüências.

- A descrição do problema diz quando o padrão deve ser utilizado, descrevendo todo o contexto de sua aplicação.

- A solução descreve como o problema pode ser resolvido, ela é descrita de forma genérica não fazendo uso de implementações, pois pode ser implantada em diversas situações.

- As conseqüências são os resultados da aplicação do padrão, assim ela descreve o custo/benefício em se utilizar o padrão.

Um livro que ficou muito conhecido como precursor desse conceito de padrões de projeto é o *"Design Patterns - Elements Of Reusable Object-Oriented Software"* (GAMMA *et al.*, 1994), que é chamado pelo apelido de Gang of Four, GoF, por ser escrito por quatro autores. Existem outros conjuntos de padrões de projeto, como o conjunto dos padrões J2EE (ALUR; MALKS; CRUPI, 2003).

A maioria dos padrões de projeto são independentes de paradigmas de programação. Geralmente eles podem ser implementados facilmente com orientação a objetos (BUSCHMANN *et al.*, 1996). Levando em conta essa facilidade e também o fato de que a aplicação dos padrões de projeto neste trabalho, é em cima desse paradigma, toda a discussão sobre os padrões será com a utilização dele. Os padrões de projetos que foram utilizados no projeto são o *Observer*, o *Singleton*, o *Service Locator*, o *Business Delegate* e o *Front Controller*. A Tabela 2.1 mostra uma relação deles com seus objetivos e justificativas para sua utilização. Um melhor detalhamento sobre os padrões de projeto que foram estudados será dada a seguir:

2.3.1 Observer

O padrão *Observer* faz parte dos padrões integrantes do GoF. De acordo com GAMMA *et al.* (1994), o objetivo do *Observer* é definir uma dependência um-para-muitos entre objetos para que quando um objeto mude seu estado, todas suas dependências sejam notificadas e realizem uma atualização automaticamente.

A Figura 2.2, demonstra o diagrama de classes do padrão *Observer*. Nela podem ser observadas duas classes, *Observado* e a *Observer* (observador). A classe *Observado* possui o método *notifyObservers()* e a classe *Observer* possui o método *update()*. O método *notifyObservers()* é responsável por enviar um aviso a todos

Tabela 2.1: Padrões de projeto estudados relacionados com seus objetivos e justificativas

Padrão	Objetivo	Justificativa
Service Locator	Encontrar um serviço ou componente	Principal padrão utilizado por este trabalho, ele será utilizado para melhorar o acesso aos serviços de aplicações que utilizam o *framework* estudado
Observer	Monitorar um objeto para saber se este possui alterações	Utilizado para que o *Service Locator* monitore os serviços
Singleton	Evitar múltiplas instâncias de um objeto	Utilizado para que exista uma única instância do *Service Locator*
Business Delegate	Diminui acoplamento entre o cliente da aplicação e as regras de negócios	Utilizado para restringir o acesso ao *Service Locator*
Front Controller	Manipular requisições ao sistema e realizar controle de acesso	Utilizado para interpretar as mensagens recebidas pelo servidor e manter o controle de acesso

os observadores caso ocorra alguma alteração na classe observada. Quando esse método é chamado, automaticamente o método *update()* de todos os *Observers*, que observam essa classe, é executado. Durante a execução do método *update()* é realizada uma atualização na classe que observa, mas essa atualização também pode refletir na classe observada.

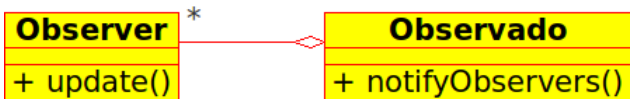

Figura 2.2: Diagrama de classes representativo do padrão Observer.

2.3.2 Singleton

O padrão *Singleton* também faz parte dos padrões integrantes do GoF. Segundo GAMMA *et al.* (1994), o objetivo do padrão *Singleton* é assegurar que uma classe possua somente uma instância e forneça um ponto de acesso global a essa instância.

A Figura 2.3 representa o diagrama de classes do padrão *Singleton*. Pode ser observado que na classe Singleton do diagrama, existe um atributo estático *instancia*. Esse atributo é responsável por manter a única instância dessa classe. E o método *getInstancia()*, também estático, é responsável por criar a instância da classe caso ela não exista, relacionando essa instância com o atributo *instancia*, e também por prover acesso ao atributo *instancia*.

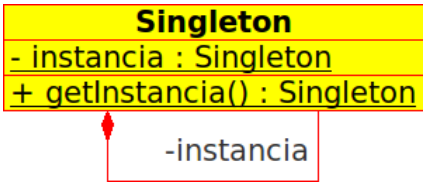

Figura 2.3: Diagrama de classes representativo do padrão Singleton.

2.3.3 Service Locator

O *Service Locator* é um padrão J2EE. O *Service Locator* implementa e encapsula a busca por serviços e componentes (ALUR; MALKS; CRUPI, 2003). Ele diminui a complexidade resultante da necessidade do cliente em procurar por um serviço distribuído e criá-lo (MURALI; PAWLAK; YOUNESSI, 2004). Então o cliente da aplicação não precisa se preocupar com a localização física do serviço, o serviço pode estar alocado localmente, remotamente na internet ou em outro lugar. O *Service Locator* também é responsável por manter um repositório de serviços chamados recentemente, chamado de *cache* (ALUR; MALKS; CRUPI, 2003). O *cache* evita buscas exaustivas por um mesmo serviço. Assim, consegue-se um aumento significativo no desempenho da execução.

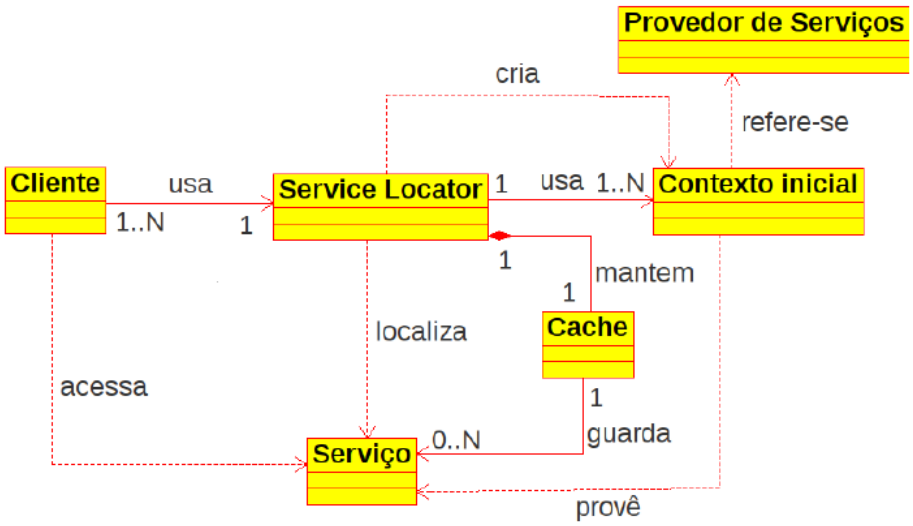

Figura 2.4: Diagrama de classes representativo do padrão Service Locator.

Na Figura 2.4 está um diagrama que demonstra como está estruturado o *Service Locator*, e na Figura 2.5, está descrita a seqüência da sua execução. Pode-se obser-

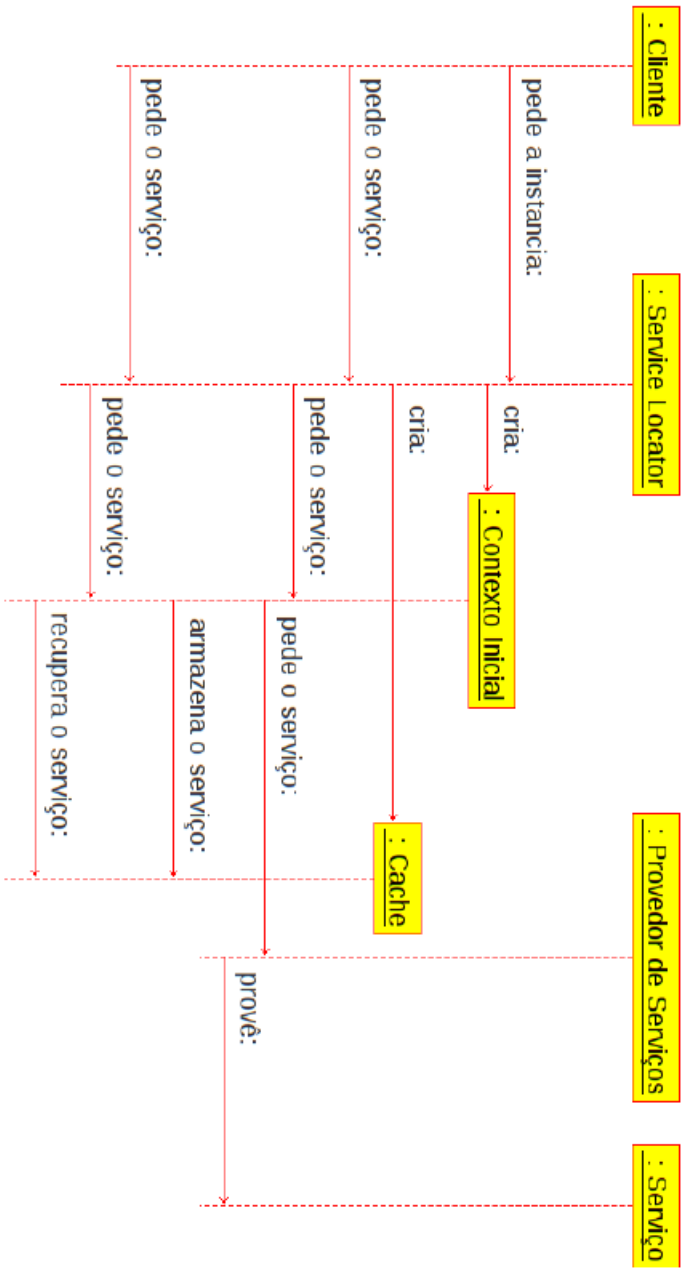

Figura 2.5: Diagrama de seqüência representativo do padrão Service Locator.

var que o cliente da aplicação interage somente com *Service Locator*. Esse por sua vez cria um contexto inicial e um *cache*. O contexto inicial conhece todos os provedores de serviços disponíveis e se comunica diretamente com eles. Provedores de serviços são responsáveis por manter a referência para os serviços ou componentes (ALUR; MALKS; CRUPI, 2003). Uma aplicação que utiliza serviços de uma outra, deve possuir dois contextos, um responsável por seus próprios serviços e outro por conhecer os provedores de serviços da outra aplicação. O *cache*, como já foi dito, é uma maneira de se ganhar *performance* na execução, não sendo necessário procurar novamente no provedor de serviços, por um serviço que já foi referenciado.

Então, quando o cliente pede por um serviço, ele o recebe a partir do *Service Locator*, sem tomar conhecimento de toda a execução. O cliente representado nos diagramas pode ser o próprio cliente da aplicação ou ainda um *Business Delegate*, padrão que será tratado a seguir.

2.3.4 Business Delegate

O *Business Delegate* é outro padrão de projeto integrante dos padrões J2EE. O papel do *Business Delegate* é esconder as regras de negócios e acesso a dados(ALUR; MALKS; CRUPI, 2003; MURALI; PAWLAK; YOUNESSI, 2004). Ao ser utilizado, o acoplamento entre o cliente da aplicação e o restante do sistema é diminuído (ALUR; MALKS; CRUPI, 2003).

Deixar transparente para o cliente operações de busca e execução do serviço é o principal benefício em se utilizar o *Business Delegate*. Assim tudo que o cliente precisa saber é como chamar o *Business Delegate* e requisitar o serviço desejado a ele. Com essa fronteira lógica criada, outros benefícios aparecem, mesmo que

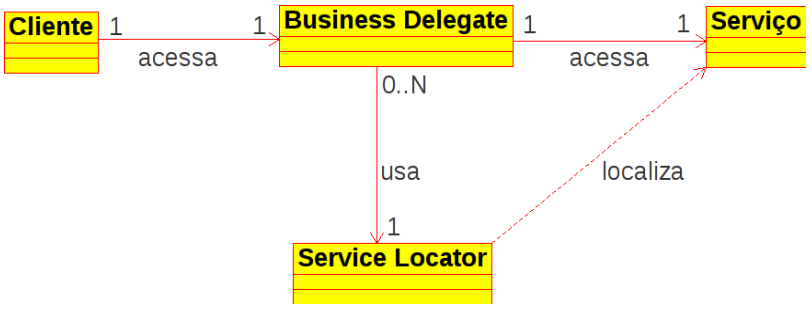

Figura 2.6: Diagrama de classes representando o Business Delegate.

estes não precisem ser necessariamente implementados com o *Business Delegate*. É possível interceptar as exceções lançadas pelos serviços e repassá-las ao cliente de forma mais amigável. Em caso de falha de um serviço é possível realizar operações de recuperação e re-tentativa sem que o cliente tome ciência disso. Além de ser possível guardar *cache* de resultados de requisições a serviços remotos, aumentando significativamente a *performance* (ALUR; MALKS; CRUPI, 2003).

A Figura 2.6 mostra como o *Business Delegate* atua. O cliente pede um serviço ao *Business Delegate* que por sua vez usa o *Service Locator* para encontrá-lo.

Na Figura 2.7 é demonstrado como funciona a requisição e execução de um serviço utilizando o *Business Delegate*. Pode ser observado que o cliente da aplicação cria uma instância do *Business Delegate* já realizando o pedido de um serviço, para posteriormente pedir sua execução.

2.3.5 Front Controller

O *Front Controller* proporciona um ponto central de entrada para manipulação de requisições (ALUR; MALKS; CRUPI, 2003; WOJCIECHOWSKI *et al.*, 2004). Com o controle lógico de acesso o *Front Controller* também diminui a quantidade

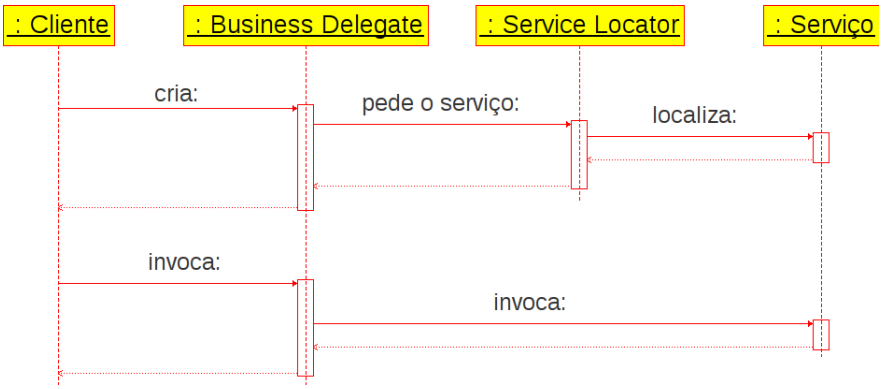

Figura 2.7: Diagrama de seqüência representando o Business Delegate.

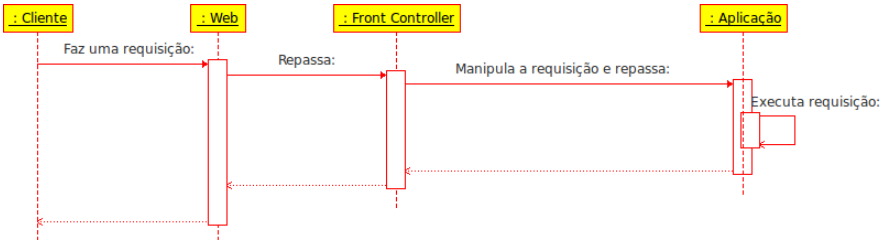

Figura 2.8: Diagrama de seqüência representando o Front Controller.

de programação embutida nas interfaces. O mais comum é se ter apenas um *Front Controller* que controla o acesso de toda a aplicação, mas isso não é uma regra. Nele ainda podem ser feitas a autenticação do usuário no sistema e controle dos privilégios de acesso a serviços do usuário (ALUR; MALKS; CRUPI, 2003). O padrão *Front Controller* está descrito nos padrões J2EE. A Figura 2.8 demonstra seu diagrama de seqüência.

2.4 Framework de referência - Iguassu

O Iguassu é um *framework* de desenvolvimento SOA que está sendo desenvolvido pelo Departamento de Ciência da Computação da Universidade Federal de Lavras. Tendo o início do seu desenvolvimento no ano de 2007, o Iguassu já está sendo utilizado no desenvolvimento de dois sistemas, um de rastreabilidade e um de monitoramento de produção industrial. Ele é fundado na arquitetura de referência descrita na Seção 2.1, principalmente na *service tier*, trazendo a essa arquitetura uma implementação em um nível mais baixo e também integrando a ela vários padrões de projeto.

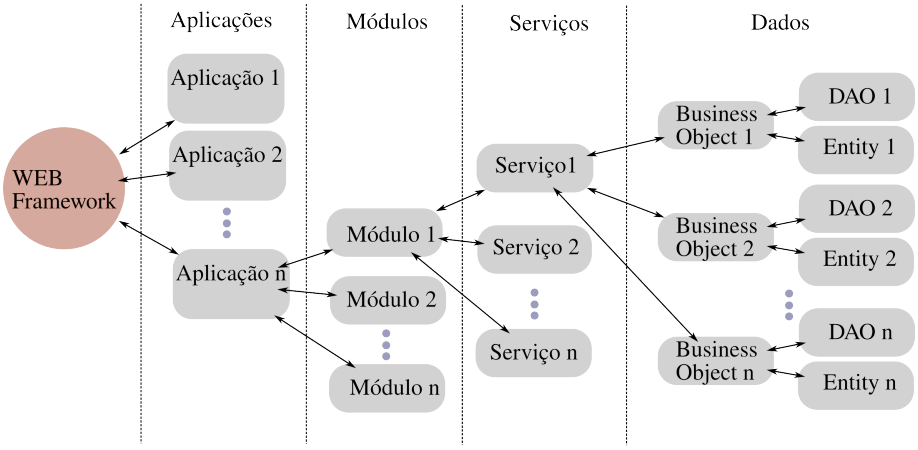

Figura 2.9: Representação gráfica da organização do framework Iguassu

O Iguassu é uma ferramenta de integração, que agrega funcionalidades de diversas ferramentas livres. Ele também se baseia, em alguns pontos, na especificação do *Enterprise Java Beans* (BURKE; MONSON, 2007). Existem outras ferramentas de integração disponíveis no mercado, tal como o *Maker* da empresa *Softwell Solutions* (Softwell Solutions, 2009), o *jCompany* da empresa *Powerlogic* (POWERLOGIC, 2009) e o *Spring Framework* que é um projeto de *software* livre

mantido pela *Spring Source Community* (JOHNSON, 2009), mas nenhuma delas é fundada em SOA.

A representação gráfica da organização do Iguassu é demonstrada na Figura 2.9, na qual pode ser observadas as camadas que formam o Iguassu. Essas camadas são: camada de aplicação, camada de módulos, camada de serviços e camada de dados. Devido ao fato de não existirem publicações que descrevam as camadas do *framework*, a explicação sobre elas será dada a partir do entendimento do autor deste trabalho.

A camada de aplicação é a camada onde ficam todas as aplicações que utilizam o Iguassu. As aplicações podem ser locais e serem acessadas a partir de uma interface local ou podem ser acessadas por uma camada *web*, utilizando-se de frameworks de desenvolvimento *web*. Cada aplicação se divide em módulos e cada módulo possui um conjunto de serviços. Já a camada de dados é responsável pela manutenção dos dados, interagindo diretamente com o banco de dados. Neste trabalho a camada de dados não é explorada. Algo importante sobre a camada de dados é que ela somente pode ser acessada pela camada de serviços.

Pode ser observada uma deficiência no projeto do Iguassu. Da forma como está na Figura 2.9, todos os serviços devem estar relacionados obrigatoriamente a um módulo da aplicação. Isso traz uma limitação, quando existem serviços que utilizem dados referentes a mais de um módulo. Esses serviços ficariam deslocados e talvez fosse necessário a criação de um módulo adicional para eles.

Atualmente, a comunicação entre o cliente e o servidor ocorre através de mensagens independentes de plataforma. Esse formato de requisição utilizando de mensagens foi baseado no modelo dos *Web Services*. Então essa comunicação é

feita utilizando-se de mensagens em *XML*, seguindo um padrão específico utilizado pelo Iguassu, semelhante às normas do *WSDL*.

O Iguassu como um *framework* está sendo implementado na tecnologia Java e faz uso de algumas ferramentas livres na sua composição. Para se criar uma aplicação utilizando o Iguassu é preciso seguir uma série de padrões e respeitar várias regras de implementação. Isto limita um pouco a liberdade do programador na criação do código, mas por outro lado proporciona que uma grande parte da aplicação seja gerada automaticamente, além de proporcionar um bom reuso de código. Assim o desenvolvimento da aplicação acelerado.

A idéia é que o Iguassu sempre se mantenha uma ferramenta simples de ser utilizada por desenvolvedores, que diminua o tempo de desenvolvimento e gastos com manutenção do sistema, por ele já agregar várias funcionalidades. E ainda se mantenha uma ferramenta que necessita baixo poder de processamento, não prejudicando o processamento das aplicações que o utilizam.

Capítulo 3

Metodologia

Neste capítulo, serão abordados os procedimentos que foram adotados para atingir os objetivos propostos.

3.1 Métodos de pesquisa

Para JUNG (2004) a pesquisa científica pode ser dividida quanto à sua natureza em duas partes: pesquisa básica ou fundamental e pesquisa aplicada ou tecnológica. Ele ainda destaca que para ser caracterizada como pesquisa aplicada ou tecnológica, a pesquisa deve ter como objetivo um novo produto ou processo. Como o principal objetivo deste trabalho é melhorar um *framework* que está sendo desenvolvido, ele se enquadra nessa classificação.

Este trabalho se classifica quanto aos seus objetivos como exploratório (JUNG, 2004). JUNG (2004) diz que a pesquisa exploratória visa procurar por teorias ou

práticas que modificarão algo existente. Por este trabalho realizar uma busca por melhorias ao Iguassu, ele pode ser classificado dessa maneira.

Quanto aos seus procedimentos este trabalho possui as características de um estudo de caso (JUNG, 2004). Isso se justifica pelo fato deste realizar um estudo sobre padrões de projetos aplicados a um framework de desenvolvimento específico.

3.2 Desenho da pesquisa

Desenho da pesquisa é uma representação gráfica dos componentes da pesquisa em seqüência lógica (YIN, 2005). Essa representação pode ser observada na Figura 3.1.

Figura 3.1: Desenho da pesquisa

3.3 Técnicas da pesquisa

As atividades que foram realizadas durante a execução deste trabalho são: estudo da arquitetura orientada a serviço, estudo dos padrões de projeto, avaliação e estudo de casos de uso, implementação e testes.

3.3.1 Estudo da Arquitetura Orientada a Serviço

Entender como SOA funciona é fundamental para entender o próprio Iguassu, pois o mesmo é fundado em cima dessa arquitetura. Mas, como o trabalho não é estritamente voltado para a arquitetura SOA, não se faz necessário um estudo aprofundado das práticas SOA e sim um conceito global e suas aplicações. É necessário entender o conceito de "serviço" e qual é o comportamento de uma aplicação baseada em SOA. Além de se estudar alguns pontos da arquitetura de referência mostrada na Figura 2.1.

3.3.2 Estudo dos padrões de projeto

O estudo dos padrões de projeto utilizados no Iguassu é essencial. Sem esse estudo não é possível implementar funcionalidades para o mesmo, de modo a evitar que sejam feitas alterações a sua organização. Além da importância para o entendimento de sua estrutura.

Tal estudo foi feito a partir dos padrões do GoF e J2EE. A partir do estudo foi possível propor maneiras de poder implementá-los. Os padrões estudados foram: *Singleton* e *Observer* do GoF e os padrões *Business Delegate*, *Service Locator* e *Front Controller* dos padrões J2EE.

3.3.3 Avaliação e estudo de casos de uso

Para poder propor implementações para os padrões de projeto estudados, antes foi necessário verificar em quais situações poderiam ser utilizados. Em aplicações feitas utilizando o Iguassu, um serviço pode ser invocado de lugares diferentes e

também pode estar fisicamente posicionado em locais diferentes. A Figura 3.2 mostra os casos de uso para chamadas de serviços, esses foram listados a seguir:

- **Chamada remota a execução de um serviço**. Isso ocorre principalmente quanto existe uma interface *web* e todas as chamadas são remotas.
- **Chamada local a execução de um serviço**. Ocorre quando existe uma aplicação *desktop* que necessita acessar um serviço na mesma máquina.
- **Chamada a execução de um serviço a partir de um processo interno ou outro serviço**. Um serviço pode precisar da execução de outros serviços para poder atingir seu objetivo. Então esse faz chamada a outros serviços.
- **Chamada a execução de um serviço não alocado fisicamente no mesmo local do Iguassu**. Quando existe um serviço externo que pode ser acessado por algum protocolo de comunicação. Ele pode ser um *Web Service*, por exemplo.

Tendo isto precisou-se modelar esses casos de uso para poder propor uma implementação.

3.3.4 Implementação

Após ter sido proposta uma modelagem para os casos de uso, pode ser realizada sua implementação. A implementação foi realizada em linguagem Java e integrada ao Iguassu.

Tendo feito isto foi necessário realizar testes para verificar se as modificações atenderam ao que foi proposto.

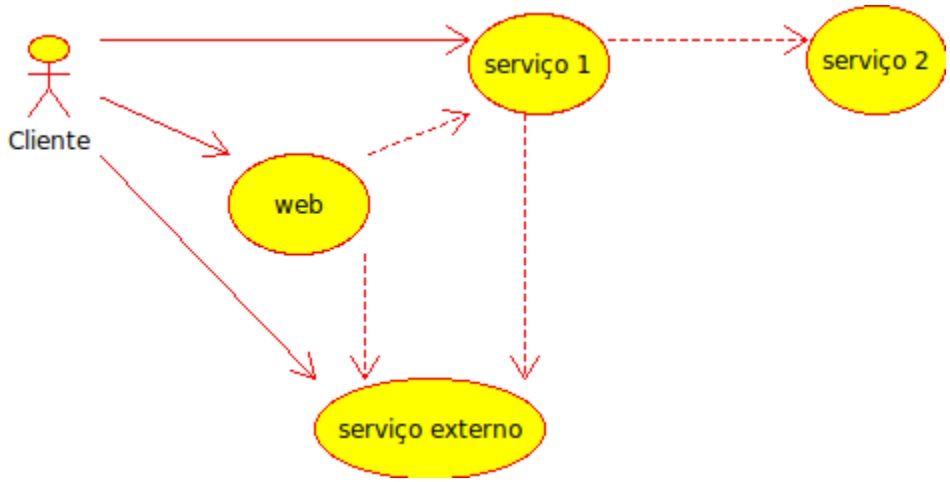

Figura 3.2: Casos de uso de chamada de serviço

3.3.5 Testes

Para poder verificar a corretude da implementação dos padrões de projeto e se estes conseguiram abranger todos os casos de uso, foram necessários realizar testes. Como os casos de uso abrangem somente chamadas a serviços, realizar um teste que execute as chamadas possíveis de serviços atenderia ao problema. O teste deve abranger dois pontos, testar a implementação do Iguassu, incluindo a implementação dos padrões de projeto, e testar a execução do serviço.

Como dito anteriormente o Iguassu realiza suas requisições através de mensagens *XML* e elas obedecem um padrão próprio. Então é interessante que o teste realize sua execução desde o envio da mensagem. Pois desta forma ele consegue interagir com todas as camadas do Iguassu.

Por possuir essa estrutura toda específica e pelos testes necessários serem somente executar as chamadas a serviços, este trabalho não se preocupou em in-

vestigar sobre técnicas e ferramentas de testes existentes, mesmo porque avaliar métodos e ferramentas de testes não é o enfoque deste trabalho.

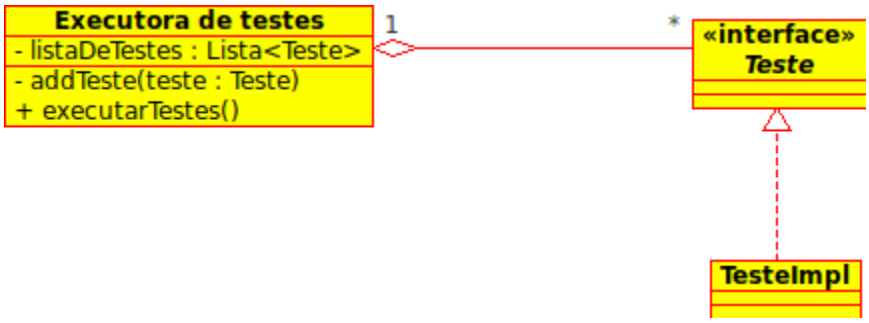

Figura 3.3: Diagrama de classes representando o modelo de testes do Iguassu

No Iguassu existia um modelo de testes, representado na Figura 3.3, este tinha por objetivo testar a manipulação dos dados de cada elemento do banco de dados. Esse modelo baseava-se em uma classe que mantinha uma lista de testes. Todos os testes que fossem adicionados nessa lista eram executados seqüencialmente. Para que possa fazer parte dessa lista o teste deveria implementar uma interface de testes. Para implementar essa classe o programador tinha que implementar todo o teste arbitrariamente, desde a montagem da mensagem de requisição. Durante a execução o programador também tinha que fazer a verificação da corretude do retorno das requisições. Nesse ponto poderiam ocorrer falhas que fossem imperceptíveis a um olhar menos detalhado. Havia uma necessidade de melhorar esse modelo, e principalmente diminuir a responsabilidade do programador, evitando falhas nos testes. Então uma extensão a esse modelo foi proposta por esse trabalho.

3.4 Abrangência da pesquisa

A abrangência deste trabalho é limitada a aplicação dos padrões de projeto que foram escolhidos ao *framework* de referência. Como o estudo foi direcionado, não existem parâmetros para se dizer que este se aplique, ou não, em outros casos.

Capítulo 4

Resultados

Seguindo os métodos propostos conseguiu-se como resultado, modelar e implementar os casos de uso e também o modelo de testes. Essas implementações serão abordadas neste capítulo .

4.1 Acesso a Serviços

Para melhorar o acesso aos serviços foram implementados os padrões *Service Locator* e *Business Delegate*.

O *Service Locator* foi implementado como sendo um *Singleton*. Existindo assim somente uma instância sua, essa instância é responsável pela localização dos serviços para a aplicação. Quando criada a instância carrega todos os contextos que por sua vez têm mapeados todos os serviços através dos provedores de serviços. Assim o *Service Locator* utiliza-se desses contextos para encontrar os serviços necessários. Ele também foi implementado segundo o padrão *Observer*. Essa

implementação é interessante para o monitoramento do *cache* de serviços, pois um serviço que está em uso não poderia ser acessado novamente. Para garantir isso no *cache* de serviços existe uma forma de se bloquear o acesso à serviços em utilização. O *Observer* entra para se saber quando o serviço termina sua execução, nesse momento ele pode ser liberado do bloqueio e novamente ser acessado.

A implementação do *Business Delegate* foi feita pensando no encapsulamento de acesso aos serviços. Onde ele é responsável por acessar o *Service Locator* para encontrar o serviço e posteriormente executá-lo. Caso a execução do serviço resulte em uma exceção, ele não conseguirá notificar o *Service Locator* para ser liberado do bloqueio. Então o *Business Delegate* captura a exceção resultante, repassa a mensagem de erro ao *Front Controller* e ainda avisa ao *Service Locator* que o serviço terminou sua execução.

O modo como foram implementadas as utilizações do *Business Delegate* e do *Service Locator* e como o Iguassu ficou estruturado, estão a seguir.

4.1.1 Iguassu

A implementação dos padrões citados resultou em uma mudança na organização do Iguassu. Sua nova representação está descrita na Figura 4.1.

Antes da implementação dos padrões de projeto, para se acessar um serviço era necessário saber em qual módulo ele estava. A busca pelo serviço então começava buscando pelo módulo deste, isso pode ser observado na Figura 2.9. Com as alterações realizadas por este trabalho, essa organização mudou. Agora para se acessar um serviço basta solicitar ao *Business Delegate*, Figura 4.1. A aplicação não deixou de ser dividida em módulos, mas não é mais necessário passar pelos módulos antes de encontrar os serviços.

Nesta nova organização do Iguassu, os serviços foram agrupados em provedores de serviços. Uma boa prática é a de criar um provedor de serviços para cada módulo. Esta organização é a que foi adotada por este trabalho.

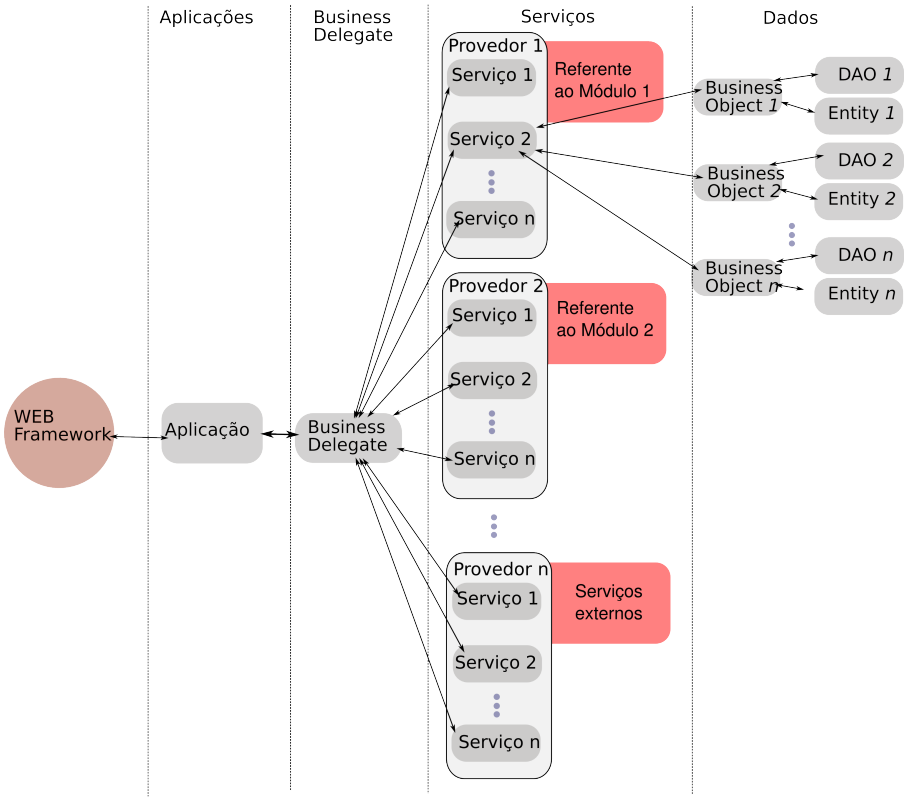

Figura 4.1: Nova representação do Iguassu

4.1.2 Chamada remota a execução de um serviço

Para se poder realizar chamadas a serviços a partir de uma aplicação remota ou uma interface *web*, é necessário comunicar-se com o Iguassu através de mensagens *XML*, como explicado anteriormente. Essa mensagem ao chegar no Iguassu passa pela camada *web* e ao chegar no *Front Controller* é traduzida, uma autenticação

do usuário no sistema deve ser feita neste momento. Após ter acesso ao sistema, a requisição ao serviço que vem dentro da mensagem pode ser executada. Para isso ela deve passar pelo *Business Delegate* que verifica se o usuário autenticado no sistema tem autorização para acessar o serviço que está tentando executar. Caso seja garantido o acesso o *Business Delegate* pede pela instância do *Service Locator*. Se já estiver instanciado ele retorna sua instância, senão o *Service Locator* cria uma instância carregando os contextos. Após instanciado o *Service Locator* busca no seu contexto o serviço. Caso encontre ele o retorna ao *Business Delegate*, que o retorna ao *Front Controller*, caso contrário ele lança uma exceção dizendo que não foi encontrado o serviço. Ao receber o serviço o *Front Controller* pede sua execução. O *Business Delegate* então executa o serviço e ao fim da execução retorna o resultado ao *Front Controller*. No *Front Controller* esse resultado é encapsulado em uma mensagem *XML* e retornado à aplicação que realizou a requisição. Toda essa execução pode ser visualizada na Figura 4.2.

4.1.3 Chamada local a execução de um serviço

Uma aplicação que está rodando no mesmo local que o Iguassu não necessita de realizar uma requisição via *web*. Essa pode chamar diretamente o *Front Controller* e pedir a execução do serviço desejado. Como não há uma requisição *web*, também não se faz necessária a utilização da mensagem *XML*, podendo ser transitados os dados diretamente. Este é o único ponto que diverge sua utilização em relação ao item anterior. Sua implementação está demonstrada na Figura 4.3.

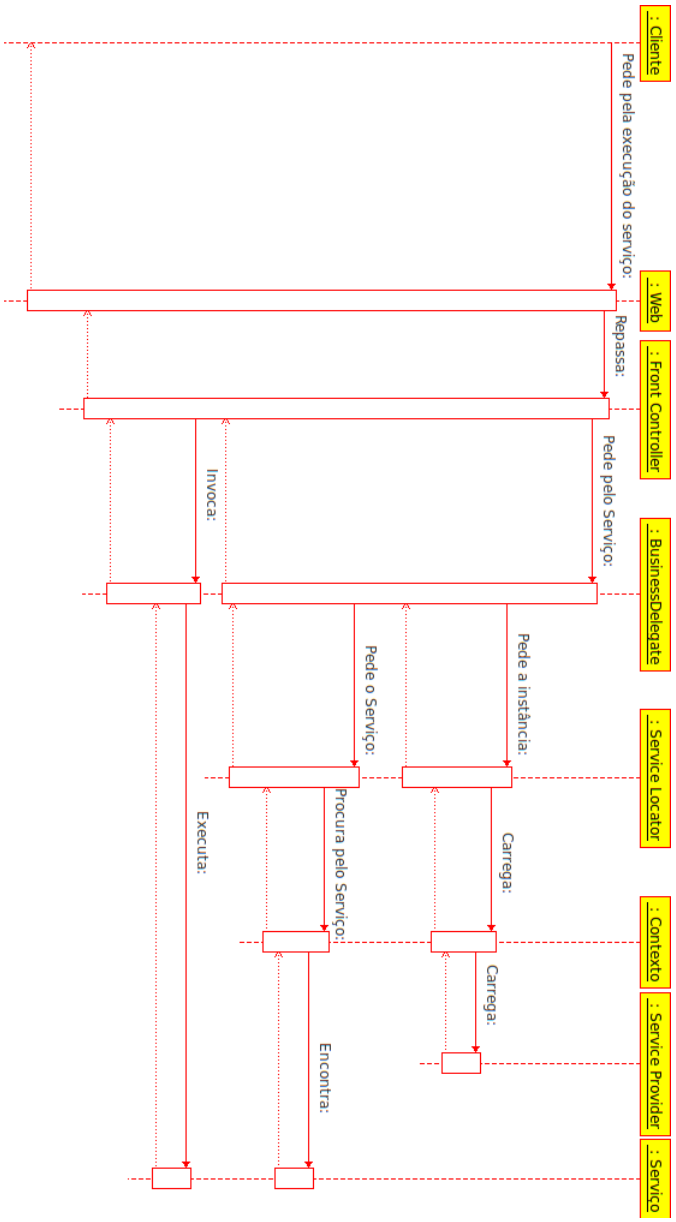

Figura 4.2: Diagrama de classes representando a chamada remota à execução de um serviço.

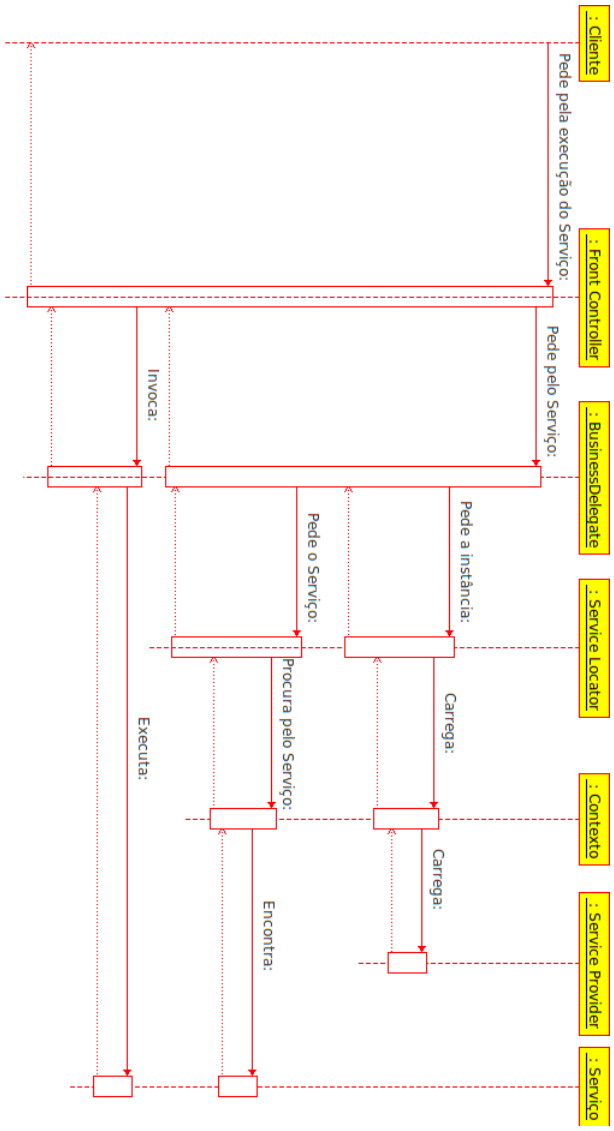

Figura 4.3: Diagrama de seqüência representando a chamada local à execução de um serviço.

4.1.4 Chamada a execução de um serviço a partir de um processo interno ou outro serviço

Um serviço que está em execução já passou pelas validações de acesso então as chamadas de serviços que esse pode realizar não precisam ser novamente valida-

das. O mesmo ocorre quando é um mecanismo interno que está tentando executar um serviço, ele já é algo integrante do sistema, não precisa de autorização. Então, nesses casos, para encontrar o serviço basta pegar a instância do *Service Locator* e pedir que ela encontre o serviço desejado. Quando o serviço é retornado pode-se então executá-lo diretamente. Essa chamada está descrita na Figura 4.4.

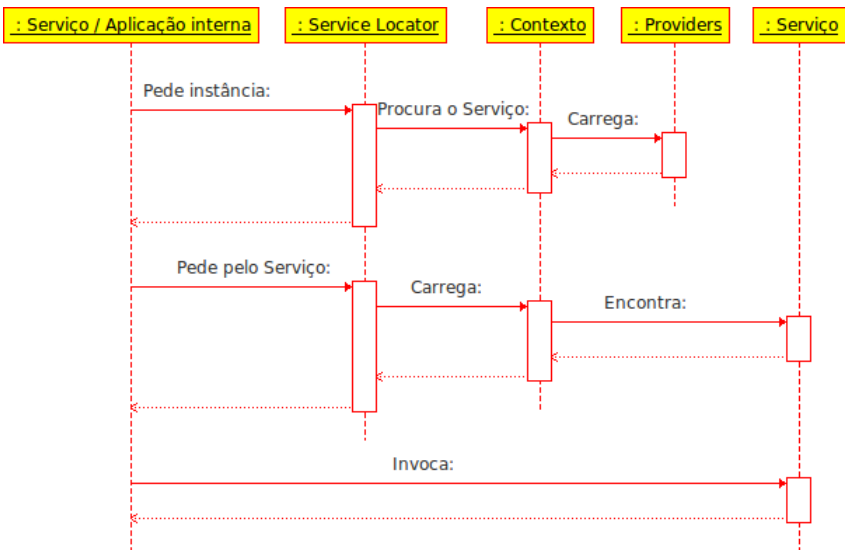

Figura 4.4: Diagrama de seqüência representando a chamada à execução de um serviço a partir de um processo interno ou outro serviço.

4.1.5 Chamada a execução de um serviço não alocado fisicamente no mesmo local do Iguassu

No Iguassu para ser reconhecido com serviço, uma classe deve implementar uma interface de serviço própria e ser provido por um provedor dentro de um contexto. Existe uma interface estendida que define métodos para acesso remoto do serviço. Então para executar um serviço alocado em outro local é necessário que exista um outro serviço que vai estar alocado localmente que implemente essa interface estendida e seja provido em um contexto. Esse serviço fica responsável por realizar a chamada para o serviço externo. Funcionando como intermediador pega a resposta do serviço externo e a converte em um formato compreendido pelo Iguassu. Essa resposta é devolvida a quem fez a requisição sem que esse e nem o *Service Locator*, tomem conhecimento de sua real localização. Esta implementação está demonstrada na Figura 4.5. Um serviço seguindo esse modelo foi implementado para executar uma chamada a um *Web Service* público. Esse *Web Service* é um conversor de unidades de computador. Ele recebe três parâmetros dentro da mensagem *SOAP*, a unidade atual, a unidade desejada e o valor numérico. Então depois da execução ele retorna uma mensagem contendo o valor convertido. Para poder acessar esse *Web Service* foi implementado um serviço no modelo Iguassu responsável por montar, fazer a requisição e interpretar a resposta. O cliente da aplicação para executar esse serviço somente precisa passar os valores da operação para a chamada desse serviço Iguassu. Outro exemplo possível de ser implementado nesse formato, é o de se criar um serviço, que faça uma chamada a um serviço de outra aplicação Iguassu, que esteja rodando em outro lugar. Para isso, esse serviço deve montar uma mensagem *XML* realizar uma requisição tomando o papel de cliente do diagrama na Figura 4.2. Ele também deve interpretar a mensagem de

volta e repassar o resultado internamente como se o serviço estivesse executando localmente.

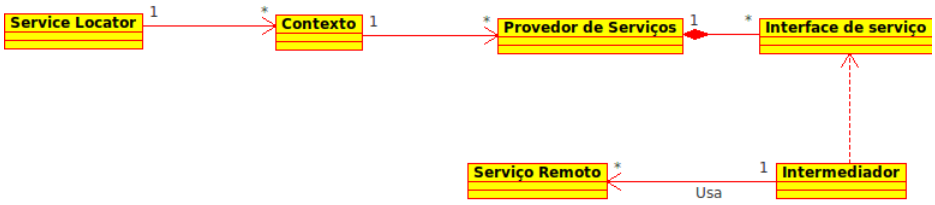

Figura 4.5: Diagrama de classes representando um serviço alocado remotamente.

4.2 Modelo de Testes

Na tentativa de melhorar o modelo de testes utilizado pelo Iguassu foi proposta uma extensão ao modelo de testes, representada na Figura 4.6. A extensão busca comprovar a corretude da implementação dos padrões de projeto e ainda diminuir a responsabilidade do programador na criação das classes de teste.

O novo modelo de testes se baseia na criação de uma classe que centraliza todos os métodos que realizam as operações de manutenção do banco de dados. Essa classe se responsabiliza por: criar as mensagens *XML* das requisições, executar as requisições e verificar se as requisições foram bem sucedidas.

Para criar um teste o programador precisa extender essa classe e implementar três métodos que dirão quais dados serão manipulados. Um método que preenche uma lista de elementos, outro que preenche um elemento isolado com características diferenciadas e um terceiro que altere o elemento isolado.

Após a criação da classe de testes ela será colocada na lista de execução e aguardará sua vez de ser executada. No momento de sua execução esta classe exe-

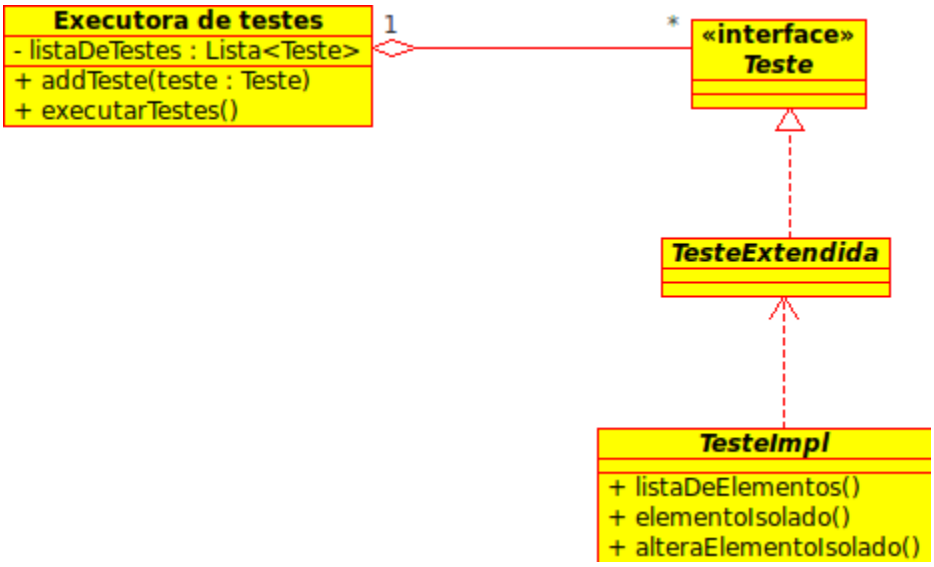

Figura 4.6: Diagrama de classes representando o novo modelo de testes do Iguassu

cutará a sequência de execuções representada na Figura 4.7, essa pilha é descrita a seguir:

- Os elementos definidos pelo método que preenche a lista de elementos são inseridos no banco;
- É realizada uma busca pela chave primária, valor único identificador de um elemento no banco de dados, dos elementos inseridos;
- Uma verificação é realizada pra se saber se todos os elementos foram corretamente inseridos;
- Todos os elementos inseridos são removidos da base de dados;
- Para se verificar se todos foram de fato removidos, é realizada uma busca por todos os elementos do mesmo tipo dos inseridos;

- Após a busca é comparado o seu resultado com os elementos que foram inseridos, verificando se ainda se encontram no banco de dados;
- Nesse ponto é inserido o elemento isolado;
- Uma busca parametrizada com as características especiais do elemento isolado é realizada;
- É verificado se o retorno da busca parametrizada é idêntico ao elemento que foi inserido;
- Então é realizado uma operação para modificar esse elemento;
- E por fim, é realizada uma busca pela chave primária desse elemento e em seguida verificado se o mesmo foi alterado com sucesso.

Após toda essa execução as informações geradas no teste são armazenadas em um relatório, onde quem realiza os testes pode se orientar para verificar as falhas.

Esta nova maneira de se realizar os testes realiza operações exaustivas, e cada uma dessas operações passa por todas as camadas do Iguassu e interagem com praticamente todo o sistema. Como a chamada dos testes passa por todas as camadas do Iguassu pode-se dizer que se o teste for bem sucedido, a implementação dos padrões de projeto foi feita de forma correta. Desta forma comprovamos a corretude das implementações do trabalho.

Mesmo assim os testes ainda não englobam tudo que é preciso testar numa aplicação, como por exemplo a execução de regras de negócios que não são somente manutenção de dados. Mas já servem para o propósito deste trabalho e ainda previnem uma grande quantidade de erros, além de terem se tornado simples e rápidos de serem implementados e executados.

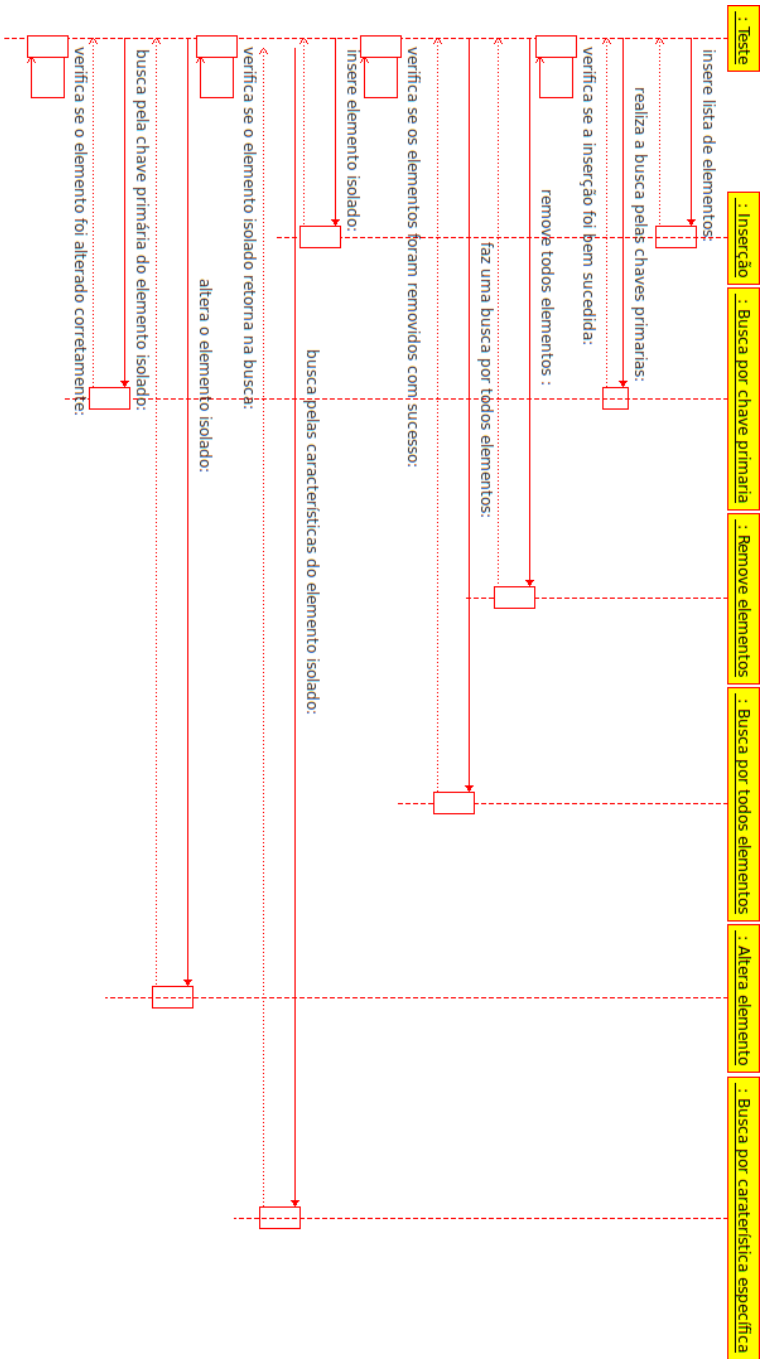

Figura 4.7: Diagrama de sequência representando a execução de uma classe de testes. Nele podem ser observados a classe de teste e os serviços que ela acessa.

Capítulo 5

Discussão

Neste trabalho, a partir do estudo de padrões de projeto, foram propostas melhorias na arquitetura do *framework* Iguassu, simplificando o acesso a serviços. Ainda para poder testar a implementação dos padrões de projeto, foi proposto um novo modelo de testes. Este auxiliará também nos testes das aplicações que utilizam o Iguassu senão o próprio *framework*.

Após a implementação dos modelos propostos, foi possível verificar melhorias na utilização do Iguassu. Isso porque existe um projeto em andamento que o utiliza como ferramenta de desenvolvimento. Uma melhoria observada, foi uma melhor organização da chamada dos serviços e com isto melhorando seu entendimento por parte dos desenvolvedores. Além de ter sido inserido o modelo para a execução de um serviço remoto a partir de um serviço Iguassu. Isso possibilitou a integração com *Web Services* ou qualquer outra aplicação, abrindo assim os horizontes de comunicação de aplicações que utilizam o Iguassu. Outra melhoria foi a diminuição significativa do tempo de implementação das classes de testes e também sua execução.

Apesar deste trabalho ter ajudado a melhorar a organização do Iguassu, muito ainda pode ser feito. Um exemplo é a implementação do padrão *Business Delegate*, que atualmente funciona somente para abstrair o acesso ao *Service Locator* pelo *Front Controller*, mas existem várias outras funcionalidades que podem ser incorporadas a ele. Uma delas é a criação de um *cache* para armazenar resultados de execução de serviços para melhorar performance na sua execução. Outra funcionalidade que pode ser implementada é a de o *Business Delegate* poder realizar re-tentativas caso um serviço não seja executado corretamente. Estas duas funcionalidades se tornam ainda mais interessante quando se trata da chamada de um serviço que realiza a execução de um serviço externo ou *Web Service*, onde podem existir erros de comunicação com o servidor que os provê. Tanto estas implementações como também continuar a busca por outros padrões que tragam melhorias ao Iguassu, são temas para trabalhos futuros.

Referências Bibliográficas

ALPERT, S.; BROWN, K.; WOOLF, B. *The Design Patterns Smalltalk Companion*. [S.l.: s.n.], 1998.

ALUR, D.; MALKS, D.; CRUPI, J. *Core J2EE Patterns: Best Practices and Design Strategies (2nd Edition)*. [S.l.]: Prentice Hall PTR, 2003.

BARAI, M.; CASELLI, V.; CHRISTUDAS, B. A. *Service Oriented Architecture with Java*. [S.l.]: Packt Publishing, 2008.

BOOTH, D.; HAAS, H.; MCCABE, F.; NEWCOMER, E.; CHAMPION, M.; FERRIS, C.; ORCHARD, D. Web Services Architecture,W3C Working Group Note. 2004.

BURKE, B.; MONSON, R. *Enterprise Javabeans 3.0*. [S.l.: s.n.], 2007.

BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H.; SOMMERLAD, P.; STAL, M. *Pattern-Oriented Software Architecture Volume 1: A System of Patterns*. [S.l.: s.n.], 1996.

DURVASULA, S.; GUTTMANN, M.; KUMAR, A.; LAMB, J.; MITCHELL, T.; ORAL, B.; PAI, Y.; SEDLACK, T.; SHARMA, H.; SUNDARESAN., S. R. *Soa practitioner's guide.* [S.l.]: BEA Systems Inc. white paper, 2006.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. M. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison-Wesley Professional., 1994.

GUDGIN, M.; HADLEY, M.; MENDELSOHN, N.; MOREAU, J.-J.; NIELSEN, H. F.; KARMARKAR, A.; LAFON., Y. SOAP Version 1.2.W3C Recommendation. 2007.

JOHNSON, R. *Introduction to the Spring Framework*. 2009. Acessado em 25 de junho de 2009. Disponível em: <http://www.theserverside.com/tt/articles/article-.tss?l=IntrotoSpring25>.

JUNG, C. F. *Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos.* [S.l.]: Rio de Janeiro: Axcel Books do Brasil, 2004.

KRAFZIG, D.; BANKE, K.; SLAMA, D. *Enterprise SOA: Service-Oriented Architecture Best Practices.* [S.l.]: Prentice Hall PTR, 2004.

MACKENZIE, C. M.; LASKEY, K.; MCCABE, F.; BROWN, P. F.; METZ., R. *Reference Model for Service Oriented Architecture 1.0*. [S.l.]: OASIS Standard, 2006.

MURALI, T.; PAWLAK, R.; YOUNESSI, H. Applying aspect orientation to j2ee business tier patterns. *Aspects, Components and Patterns for Infrastructure Software Workshop, AOSD2004*, 2004.

POWERLOGIC. *jCompany Developer Suite*. 2009. Acessado em 25 de junho de 2009. Disponível em: <http://www.powerlogic.com.br>.

RYMAN, A.; CHINNICI, R.; WEERAWARANA, S.; MOREAU, J.-J. Web Services Description Language (WSDL) Version 2.0,W3C Recommendation. 2007.

SAMPAIO, C. *SOA e Web Services em Java*. [S.l.]: Editora Brasport, 2006.

Softwell Solutions. *O Maker*. 2009. Acessado em 25 de junho de 2009. Disponível em: <http://www.softwell.com.br/PaginaAction?pagina=OMaker>.

WOJCIECHOWSKI, J.; SAKOWICZ, B.; DURA, K.; NAPIERALSKI, A. Mvc model, struts framework and file upload issues in web applications based on j2ee platform. *Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004. Proceedings of the International Conference*, 2004.

YIN, R. *Estudo de caso: planejamento e métodos. 3. ed.* [S.l.]: São Paulo: Artmed, 2005.